MANJULA B. WALDRON
KENNETH J. WALDRON
EDITORS

MECHANICAL DESIGN

THEORY & METHODOLOGY

Springer

Mechanical Design: Theory and Methodology

Springer Science+Business Media, LLC

Manjula B. Waldron
Kenneth J. Waldron
Editors

Mechanical Design: Theory and Methodology

With 109 Illustrations

Springer

Manjula B. Waldron
Biomedical Engineering Center
Ohio State University
Columbus, OH 43210
USA

Kenneth J. Waldron
Department of Mechanical Engineering
Ohio State University
Columbus, OH 43210
USA

Library of Congress Cataloging-in-Publication Data

Mechanical design : theory and methodology / edited by Manjula B.
Waldron, Kenneth J. Waldron.
p. cm.
Includes bibliographical references
ISBN 978-1-4757-2563-6 ISBN 978-1-4757-2561-2 (eBook)
DOI 10.1007/978-1-4757-2561-2

1. Engineering design—Data processing. 2. Computer-aided design.
I. Waldron, Manjula B. II. Waldron, Kenneth J.
TA174.M384 1996
620′.0042′0285—dc20 95-37685
CIP

Printed on acid-free paper.

Production managed by Laura Carlson; manufacturing supervised by Joseph Quatela.
Typeset by Asco Trade Typesetting Ltd., Hong Kong

9 8 7 6 5 4 3 2 1

ISBN 978-1-4757-2563-6 SPIN 10034603

Preface

This volume, *Mechanical Design: Theory and Methodology*, has been put together over the past four years. Most of the work is ongoing as can be ascertained easily from the text. One can argue that this is so for any text or monograph. Any such book is only a snapshot in time, giving information about the state of knowledge of the authors when the book was compiled. The chapters have been updated and are representative of the state of the art in the field of design theory and methodology.

It is barely over a decade that design as an area of study was revived, mostly at the behest of industry, government, and academic leaders. Professor Nam Suh, then the head of the Engineering Directorate at the National Science Foundation, provided much of the impetus for the needed effort. The results of early work of researchers, many of whom have authored chapters in this book, were fundamental in conceiving the ideas behind Design for X or DFX and concurrent engineering issues. The artificial intelligence community had a strong influence in developing the required computer tools mainly because the field had a history of interdisciplinary work. Psychologists, computer scientists, and engineers worked together to understand what support tools will improve the design process. While this influence continues today, there is an increased awareness that a much broader community needs to be involved.

This volume is a small step. It compiles information currently available in the field of design theory and methodology. The information provided addresses process and product issues. Most of the authors emerged from, or are associated with, mechanical engineering design, hence the title contains "mechanical design." This is to accommodate the current discipline-specific culture and provides this volume with a disciplinary home. However, the information contained easily extends to any other engineering discipline as well.

The aim of this book is to provide the reader with both the theory and the applications of design methodology. It captures current research results in the field and provides a compendium on which design educators can base their design teaching philosophies. Chapters from this book were successfully used in teaching an integrated product design course.

All of the contributors to this volume are active researchers in the area of design theory and methodology and have each made significant contributions to this field. Every chapter is self-contained and is readable without assistance from any other chapter. The organization of the book requires explanation. As with any organization, one can arrange material in many different ways. Our organization is somewhat unique for we have included both product- and process-related issues. The structure of this volume reflects these dimensions. The chapters span theory to application, process and product tools, and information flow, from specific domain knowledge representation to more general analogical reasoning, from single concept design to life cycle design and quality issues.

The completion of this volume is largely due to the timely contributions and revisions by the authors. We would like to acknowledge the patience of the contributors and their willingness to revise their work as they waited for the publication of this volume. Thanks are also due to Myrtis Smith, Soo Won Kim, and Debbie Wong for their assistance in contacting authors, formatting of chapters, and various required tasks which they performed. Thanks are also due to our children, Andrew, Lalitha, and Paul, who did not complain while we spent long hours in the evenings and on weekends completing this volume rather than spending time with them.

MANJULA B. WALDRON
KENNETH J. WALDRON

Contents

Preface v
Contributors ix

1. Introduction 1
Kenneth J. Waldron and Manjula B. Waldron

2. The Influence of the Designer's Expertise on the Design Process 5
Manjula B. Waldron and Kenneth J. Waldron

3. Methods of Studying Mechanical Design 21
Manjula B. Waldron and Kenneth J. Waldron

4. Design Characterizations 35
Manjula B. Waldron and Kenneth J. Waldron

5. An Observational Methodology for Studying Group Design Activity 52
John C. Tang and Larry J. Leifer

6. Representation of Conceptual Mechanical Design Knowledge 71
Albert Esterline, Megan Arnold, Donald R. Riley, and Arthur G. Erdman

7. Configuring Systems Using Available Assets: A Conceptual, Decision-Based Perspective 127
P. N. Koch, J. D. Peplinski, F. Mistree, and J. K. Allen

8. Group Decision Making in Design 161
Deborah L. Thurston

9. Routineness Revisited 195
Dave Brown

10. A Comparative Analysis of Techniques in Engineering Design 209
Srikanth M. Kannapan and Kurt M. Marshek

11. A Data Representation for Collaborative Mechanical Design 237
Richard L. Nagy, David G. Ullman, and Thomas Dietterich

12. Characterizing Human Analogical Reasoning 254
Beth Adelson

13. Entropy Measures in Engineering Design 275
Ronald S. LaFleur

14. Design Education 299
Kenneth J. Waldron and Manjula B. Waldron

15. Life-cycle Design 312
K. Ishii

16. Support for Workflow Process Collaboration 329
Jay Ramanathan

17. Improved Total Development Process: Overcoming the Ten Cash Drains 365
Don Clausing

Contributors

Beth Adelson
Rutgers University
Camden, New Jersey

Janet K. Allen
Systems Realization Laboratory
The Woodruff School of Mechanical Engineering
Georgia Institute of Technology
Atlanta, Georgia

Megan Arnold
Department of Civil Engineering
University of Minnesota
Minneapolis, Minnesota

David C. Brown
Professor
Artificial Intelligence Research Group
Computer Science Department
Worcester Polytechnic Institute
Worcester, Massachusetts

Don Clausing
Professor of Mechanical Engineering
Massachusetts Institute of Technology
Cambridge, Massachusetts

Thomas G. Dietterich
Professor
Department of Computer Science
Oregon State University
Corvallis, Oregon

Arthur G. Erdman
Professor
Department of Mechanical Engineering
University of Minnesota
Minneapolis, Minnesota

Albert Esterline
Department of Computer Science
North Carolina A&T State University
Greensboro, North Carolina

K. Ishii
Associate Professor
Department of Mechanical Engineering
Stanford University
Stanford, California

Srikanth M. Kannapan
Xerox Corporation
Design Research Institute
Cornell University
Ithaca, New York

Patrick Nathan Koch
Systems Realization Laboratory
The Woodruff School of Mechanical Engineering
Georgia Institute of Technology
Atlanta, Georgia

Ronald S. LaFleur
Department of Mechanical and Aeronautical Engineering
Clarkson University
Potsdam, New York

Larry J. Leifer
Center for Design Research
Mechanical Engineering Department
Stanford University
Stanford, California

Kurt M. Marshek
Department of Mechanical Engineering
The University of Texas at Austin
Austin, Texas

Farrokh Mistree
Professor of Mechanical Engineering
Systems Realization Laboratory
The Woodruff School of Mechanical Engineering
Georgia Institute of Technology
Atlanta, Georgia

Richard L. Nagy
Energy Investment Inc.
Fremont, California

Jesse David Peplinski
Systems Realization Laboratory
The Woodruff School of Mechanical Engineering
Georgia Institute of Technology
Atlanta, Georgia

Jay Ramanathan, PhD
Director of Knowledge Integration Center
UES, Inc.
Dublin, Ohio

Donald R. Riley
Professor
Department of Mechanical Engineering
Associate Vice President for Academic Affairs and Information Technology
University of Minnesota
Minneapolis, Minnesota

John C. Tang
Sun Microsystems, Inc.
Mountain View, California

Deborah L. Thurston
Associate Professor
Department of General Engineering
University of Illinois Urbana
Urbana, Illinois

David G. Ullman
Professor of Machine Design
Department of Mechanical Engineering
Oregon State University
Corvallis, Oregon

Kenneth J. Waldron, PhD, PE
The John B. Nordholt Professor and Chairman
Department of Mechanical Engineering
The Ohio State University
Columbus, Ohio

Manjula B. Waldron, PhD
Professor
Biomedical Engineering Center
The Ohio State University
Columbus, Ohio

1 Introduction

KENNETH J. WALDRON AND MANJULA B. WALDRON

The distinction between an engineer and a scientist is often forgotten. The difference is that the engineer is occupied with the creation of new artifacts, technologies, or systems, while the scientist is focused on understanding the physical world. To be sure, that understanding is essential to the creation of new technologies, but it is only the foundation upon which the engineer must build.

The creative or synthetic parts of engineering activity are mostly embodied in design and manufacture. These activities are inextricably bound together. Engineering design can be viewed as planning for manufacture. Manufacture is the act of turning that design into an artifact. History has repeatedly shown that attempts to isolate these as two separate activities have always led to inferior results. Strenuous efforts to improve manufacturing processes and manufacturing systems, which can be observed in many industries at the present time, will not lead to world-class products without world-class design.

The Challenges of the Design Process in Modern Engineering

The technical and professional environment in which a design engineer must operate is very different from that of even a few years ago. Technologies have advanced rapidly, and current engineering practice features a high level of integration of technologies that were once regarded as separate technical domains. This means that the designs of many products require skills and knowledge that cannot be encompassed by a single individual, or even a small group of individuals. Engineering design has become a team activity.

Communication has become one of the most important elements of the design process. This is ironic because engineering has traditionally been regarded as an appropriate career for those who are mathematically gifted but who score poorly in communication subjects. Traditionally, design engineers communicated their designs to manufacturers graphically by sets of working drawings. Verbal communication was limited to writing specifications and part lists. The complete inadequacy of this mode of operation has become abundantly clear. Not only is it essential for engineers from different technical domains to communicate freely throughout the design process, but it is necessary that communication with the manufacturers be bidirectional and take place throughout the design process. Communication with other members of the organization, particularly those who will be responsible for selling and servicing the product, is equally vital.

The impact of the social environment is also important. A feeling has developed in society that technology must be controlled. Regulations imposed on industries are society's attempt to achieve control. At the same time, tort litigation against manufacturing industries has had a great impact on the economics of those industries. This has created an increasingly complex legal and regulatory environment within which the design team must operate. It mandates effective communication between the designers and the legal and regulatory experts.

At the same time, competitive pressures are impelling companies continually to seek to shorten product cycle times. It is no longer possible to wait until one stage of the design process is completed before starting the next. As far as possible, design and manufacturing processes must proceed in parallel. Considerable ingenuity is involved in designing process plans to accomplish projects in a highly parallel fashion when parts of the project are dependent on information developed in other parts of the company. Sometimes risks are taken and design features and ranges of values of specifications are assumed in order to allow progress in other areas before those features have, in fact, been developed.

All of the above impacts the engineering design process and the designers who perform the process. Careful planning of the process has become essential. Experience has shown that time spent in planning the process is a very good investment, even though that time subtracts from the time available to actually execute the design. A much more structured process than in the past becomes a necessity because of the time pressure and the large number of interactions to be managed. Protocols are needed to ensure that the necessary communication with experts in other departments as diverse as marketing, warranty service, and legal affairs actually occurs. Modern computational tools can greatly facilitate the process, but they also have their limitations and they can be misapplied or misused by those who do not thoroughly understand them.

As teachers and academic researchers in engineering design, we face substantial challenges. These challenges include understanding the changing design process as it is practiced in industry and proposing improvements and developing tools to enhance its effectiveness. The challenges also include the preparation of engineering graduates to operate in this environment.

It is unrealistic to suppose that a four-year curriculum alone can equip anyone to be a professional engineer in today's industrial environment. Experience has always been a necessary component of the designer's makeup, and is even more necessary in the diverse technical and social environment of the global marketplace. While cooperative education or internship experiences are to be encouraged to provide some awareness of the industrial environment, these experiences fall far short of the level of practice and accomplishment needed to produce a fully productive engineering designer.

In this light, the challenge is to determine how much design experience should be in an engineering curriculum, and how should that component be structured. There will always be pressure from the engineering science courses that compete for curriculum time. What is the relative value of time spent on design experiences early in the curriculum as opposed to time spent in the senior year when most engineering science courses have been completed? To what extent should small design projects be distributed throughout engineering science courses as opposed to concentrating synthetic activity in a capstone experience?

In the American Society of Mechanical Engineers Publication *Innovations in Engineering Design Education*, Fisher et al. (1993) have written a white paper entitled, "Design Methodologies and New Paradigms for Design." In this paper the authors claim that "in our rush to reinstate design as a legitimate engineering activity we have often confused the process or methodology with its tools." The tools are different from the methodologies. There are many common elements in various design methodologies used in practice today, such as Quality Function Deployment (QFD), Pugh's Total Design, Taguchi's robust design, and Pahl and Beitz's prescriptive design methodologies. It is important that practitioners be familiar with these design methodologies.

Any design methodology is a prescribed sequence of actions, and it is important to know in what context a given methodology is applicable and for what problem types it is useful. It is important to understand that design is more than functional design. It is a part of an overall product realization process.

Book Organization

In this volume we have collected current contributions from many of the leading figures of the academic engineering design field. These contributions include discussions of design methodology, design practice, computational tools and assistants, and design education. The information contained in the chapters that follow will provide the reader with current research results relating to both product and process issues. Each chapter is complete and self-contained. The reader can choose the material in any sequence desired. The information will be useful to educators and practitioners alike in understanding the entire product realization process and the tools that are available for creating quality products.

This volume is unique for it contains discussion of both "soft," people-oriented issues and "hard" development of the tools and techniques necessary for making group decisions. The first five chapters are concerned with process study. Chapter 2 provides information on how the designer's level of expertise affects the design process and the resulting designs. Chapters 3, 4, and 5 discuss the methods used for studying the design process. Several process characterization models for single designers and groups of designers are discussed. Chapters 6, 7, and 8 then explore the representation of design knowledge, decision support systems for catalog designs, and group decision making with concern for information flow and its errors. Chapters 9 and 10 discus routine design and comparative analysis techniques for engineering design. Data representation models for group design activity are the topic of Chapter 11. In Chapter 12 analogical reasoning is discussed. Chapter 13, on entropy measures, is unique as it provides the concept of an energy balance principle that can provide a universal measure for efficiency of design solutions and methods. The demand that the new knowledge available for design methodology places on design education is discussed in Chapter 14. In Chapter 15 designing for the entire life cycle of the product from manufacture to retirement is discussed. The overall value of the information from several persons, as is the case in concurrent design, can be computed through compatibility measures. In Chapter 16 a workflow management tool that facilitates timely information management and decision making in the product realization process is discussed. In Chapter 17 total quality improvement principles applied to product design are discussed.

Reference

Fisher, C. A. et al. "Design Methodologies and New Paradigms for Design." In *Innovations in Engineering Design Education.* New York: American Society of Mechanical Engineers, 1993. pp. 81–84.

2
The Influence of the Designer's Expertise on the Design Process

Manjula B. Waldron and Kenneth J. Waldron

Abstract. Designers bring their prior experience and expertise to the design process whenever they read drawings, draw, or design or when they observe. It is for this reason that one cannot separate the process from the expertise of the designer. What is designed is integrally tied to the designer. The question then arises: "Are there notable differences between experienced and not-so-experienced designers, and if there are, what form do they take?" In this chapter we summarize our previous work on identifying the differences between the approaches of designers with different levels of expertise. These include the differences in recall of drawings, in reasoning about motion, in reasoning about drawings, and in designing. We present the implications of this work on the design process.

Introduction

There is a widely accepted conjecture that visualization is an important component of the conceptual mechanical design process and that humans need to visually interpret drawings and blueprints in order to recognize features and to evaluate designs for manufacturability (Luby, Dixon, & Simmons, 1986). The impact of experience with mechanical design on the speed and accuracy of recognizing and identifying appropriate design functions, features, and other factors that contribute to the interpretation of the drawings has not been studied exhaustively. The papers summarized in this chapter examine the manner in which experienced and inexperienced mechanical designers differ in the utilization, reasoning, and abstraction of information about mechanical design. Before we discuss these papers it is pertinent to examine the nature of expertise as reported in the literature.

The differences between experts and novices within a variety of knowledge domains have been reported. Experts solve problems in their domains more quickly and accurately than novices (Newell & Simon, 1972). The reasons for this are that experts have more extensive knowledge, and they have a representation of the domain in their memory. They construct an abstract and organized representation of the knowledge domain, which allows them to make inferences about component relations. Early studies suggested that the information units formed by experts correspond to abstract functional relations between items. For example, Chase and Simon (1973), found that the information units (i.e., chunks) formed by expert chess players correspond to abstract structures, such as attack and defense relations on the board. Novices do not exhibit this structure in their recall. By examining engineering graphics drawings, Cooper (1983) found that spatial information representation is sensitive to changes in information processing demands and levels of expertise in problem-solving skills. Adelson (1981) concludes that experts develop cognitive structures based on abstract functional principles of their area of expertise. These structures guide perception and recall of material in their domain. One implication of the difference in knowledge representation between experts and novices is that experts can process information from their domain more efficiently than novices. However, if experts perform a task that is incompatible with the underlying structure of their memory representation, performance deteriorates. For programmers, Adelson (1984) found that expert performance was superior when the questions asked were functional in nature, but novice performance was better when the questions focused on the concrete knowledge of the code itself.

Simulation is important in the development of mechanical designers' ability to engage in visual and spatial reasoning. For human designers, recognizing the functioning of a device may involve mental models of the device. Hegarty, Just, & Morrison (1988) asked subjects to judge which of two pulley systems required more force to lift a weight. In the series of problems, the alternative pulleys differed in mechanical advantage along one or more dimensions (e.g., number of load-bearing ropes). The researchers collected the subjects' verbal protocols and the number of accurate responses during task performance. The subjects in this experiment employed different mental models to determine the mechanical advantages of the pulley systems. Low-scoring subjects considered all relevant attributes and had no preference among the rules for the mechanical device, whereas high-scoring subjects used rules for judgment based on the configuration of the device and preferred rules based on determining attributes. While they did not relate their studies to the expertise of the subjects, they did establish that the accuracy of identifying mechanical device functioning depends on noticing relevant features and on the mental model used to simulate the device operation. Clearly, one would expect that experience within the domain should facilitate this process. Formal course work, as well as actual design experience, should augment a person's ability to engage in mental simulation of a device.

This chapter is organized to first present the visual recall differences, and differences in reasoning about motion of mechanical devices and mechanical engineering drawings, between experts and novices. We also look at the differences in their ability to form mental models and to design conceptually. Finally, a discussion on the implications for the design process is given.

Visual Recall Differences

The purpose of the research summarized in this section was to look at short-term recall differences between expert and novice mechanical designers when they were viewing mechanical engineering drawings. These differences were studied by tracking the errors they made while copying the drawings from memory, and the number of times they needed to reference the drawings when engaged in this task. By analyzing their video protocols, as they performed the tasks, we were also able to track their recall strategies. Portions of the work reported in this section were presented earlier in a conference paper by Waldron et al. (1987).

Method

The format of the study was as follows: six mechanical engineering drawings were presented to three groups of six subjects in each group. The subjects in the first group were engineering professionals considered by their peers to be expert mechanical designers. The subjects in the second group were considered to be semiexperts. This group largely consisted of graduate students in mechanical design. The third group consisted of engineering students who had had a one quarter duration course in engineering graphics. This group was considered to be novices in terms of engineering design experience. The six drawings that were chosen spanned a range of detail and complexity so as to be sufficiently challenging to subjects with different levels of experience. The level of complexity of each drawing was judged by an experienced mechanical designer. He determined that the drawing number 1 was the simplest, drawing numbers 2 and 3 were a level more complex, and drawing numbers 4, 5 and 6 were the most complex.

Subjects were asked to look at a drawing and to reproduce it from memory on a sheet of paper. They were also requested to name the part and to think aloud while completing the task. The total number of references to the drawing, and the average time between references were recorded. Furthermore, a detailed study of the first reference to the drawing was made by retaining a carbon copy of the work done up to that point. Subjects were not allowed to erase. Rather they used a different colored pencil after each viewing. The order of the colors was the same for each subject. Thus, a record of the number and types of errors made could be recorded. A verbal description of the think-aloud work, by each subject, was also recorded.

Results and Discussions

Three-way analyses of variance (ANOVA) against group and drawing numbers were performed on the following measures: total duration of references, duration of the drawing period after the first reference, number of errors made, number of references, drawing duration, duration of the longest reference, and duration of the longest drawing period. Significant overall differences among the groups were found. Overall, the expert group displayed the best performance as was to be expected. This was also evidenced by their performance and by the significant group main effect differences in the analysis.

The drawing times increased with the assessed complexity. In both the most complex and the simplest drawings, the mean number of references of the experts was significantly lower than that of the other groups. Furthermore, the mean reference time was also lower; that is, they needed significantly less time for the information to be available to them for recall. The mean duration of the drawing time after the first reference showed that the expert designers consistently drew for a longer period before referring to the original drawing than the novice designers. There was a continual increase from novice to semiexpert to expert for each drawing. This suggests that *information specification and utilization are directly related to the level of expertise of the designers.* In both the simplest and the most complicated drawings, the ratio of the drawing durations of the experts to those of the novices was roughly 2 : 1.

The number of errors made by the subjects was significantly lower for the experts than for the novice subjects. The ratio was again 1 : 2. If one takes errors to imply a misutilization or misordering of visual information, then this, again, suggests an order in the organization of the visual information that is dependent on the experience of the designer; that is, the more experienced the designer, the better is his/her organization of the visual information, leading to fewer errors. The experts made significantly fewer references to the original than the novice subjects. The ratio between the number of references made by the novice subjects to that of the experts, was once again, roughly 2 : 1 for the simplest and the most complex drawings. This suggests *a higher level of "chunking" of information during the information pickup and maintenance by the experts when compared with the novice designers.*

The names assigned to the parts by the subjects were scored by an expert mechanical designer on a 3-point scale: 0 = incorrect, 1 = partially correct, and 2 = totally correct. One-way analyses of variance were performed to test for significant differences among groups in accuracy of naming the drawings. Significant differences were found between the groups for the mean score for all drawings as well as their overall group score. The mean, overall group score for novice subjects was 0.2, for semiexperts it was 1.0, and for experts it was 1.5. This is shown in Figure 2.1; that is, the experts named the drawings significantly more correctly than the other groups. As is to be expected, naming of the parts is directly dependent on the domain knowledge, which would explain the performance gradient from novice to expert. This suggests that *the expert designers carry with them a large symbolic–visual associative knowledge that assists them in information pickup, maintenance, and utilization of this knowledge.*

One of the problems in using the verbal protocol was that the expert designers chose to use fewer verbal utterances in their protocols, but drew

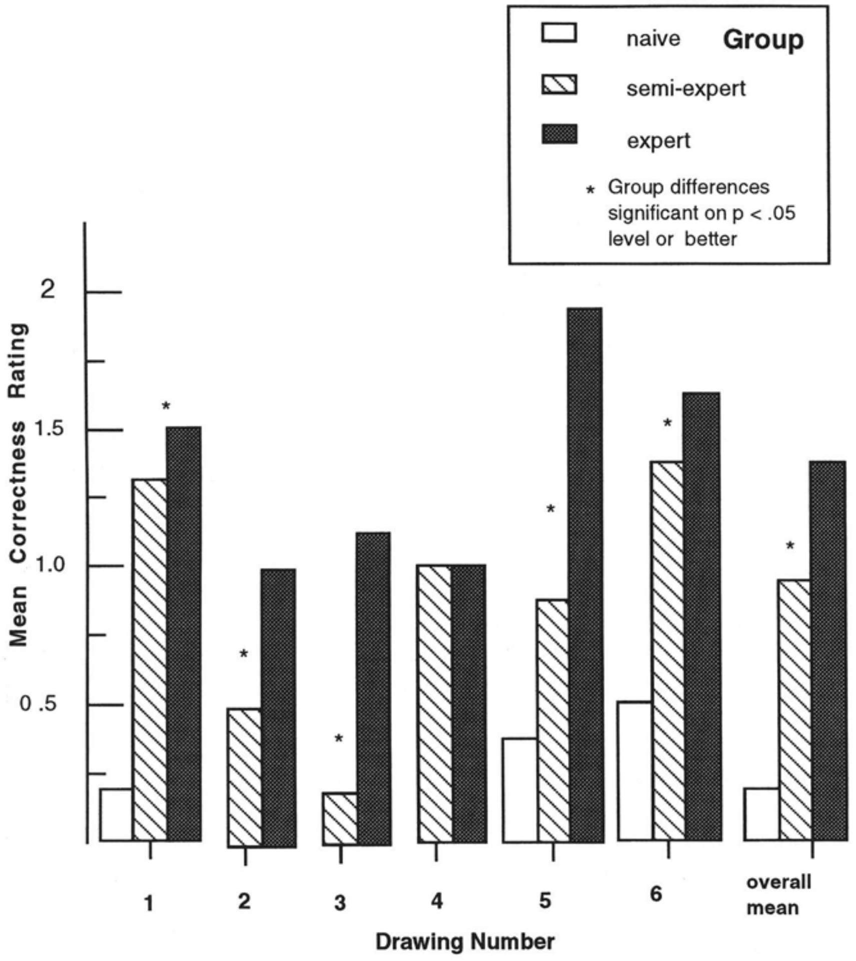


FIGURE 2.1. Mean correctness rating of names assigned by the subjects of each group by drawing number and the overall drawings.

more than the novice subjects, despite the experimenter's prompting. As one expert put it, "It is not easy to draw and talk at the same time, which would you prefer we do?"

The results of this study support the original hypothesis that the efficiency of information handling by mechanical designers is dependent on their level of expertise in design. Both reference times and drawing durations support this. Furthermore, higher-order information is dealt with at a higher symbolic level by experts than by novice designers. The latter seem to focus more on lines and sizes, whereas experts focus on features, resulting in fewer errors, fewer references to the original, and longer drawing duration between references. That is, *as experience in mechanical engineering design increased, the efficiency of copying and naming the drawings also increased.*

Differences in Reasoning About Motion

In this experiment, subjects with four levels of relevant experience attempted to judge whether an animated mechanical device corresponds to a static presentation of another mechanical device. In each problem, the animated and static devices presented either were or were not the same device. Both the static and animated devices were presented on the computer screen at the same time, using the same scale for representation. One would expect expert mechanical engineers to more accurately identify the correspondence between mechanical devices and their operation. That is, compared to novices, experts should more frequently indicate that the two devices match when the devices are the same, and should more frequently indicate that the two devices do not match when the devices are different. In addition, during perception they should attend more to functional features of the mechanical devices than novices. This section is a summary of an unpublished manuscript by Waldron and Herren (1994).

Method

The format of the experiment was as follows:

Three different types of device representing different principles of operation—*rotation*, *oscillation*, and *push rods*—were presented to each of the subjects. There were four devices of each type. Each of the devices differed from the others in its direction and/or its speed of motion. Figure 2.2 shows a schematic representation of one such device. For example, in rotational devices, the output wheel turned in the same direction as the input device, but at a slower or faster speed, or in the opposite direction to the input at a slower or faster speed. Similarly for oscillating and push rod devices.

On each trial, an animated device, with its internal mechanisms hidden, appeared on the left side of a computer screen. On the right-hand side of the screen a static device, with its internal mechanism visible, was shown. The subjects had to decide whether or not the two devices were identical. Thus, chance level was at 50%. The computer recorded two performance variables on each trial: (1) the response scored as correct or incorrect and (2) the reaction time from the appearance of the problem on the screen to the initiation of a choice response. The design was a 4 (subject category) × 2 (problem set—first versus second set) × 3 (device type—rotating, oscillating, and push rod) × 2 (problem type—match versus nonmatch) × 12 (devices, four nested within each device type) mixed factorial. Subjects completed a total of 144 trials during the experiment.

Forty-four males with four levels of mechanical engineering experience participated in the study. Group 0 consisted of 11 undergraduate students with no engineering background, Group 1 contained 15 undergraduate engineering majors, Group 2 consisted of 9 graduate students in mechanical engineering, and Group 3 had 9 expert professional mechanical designers.

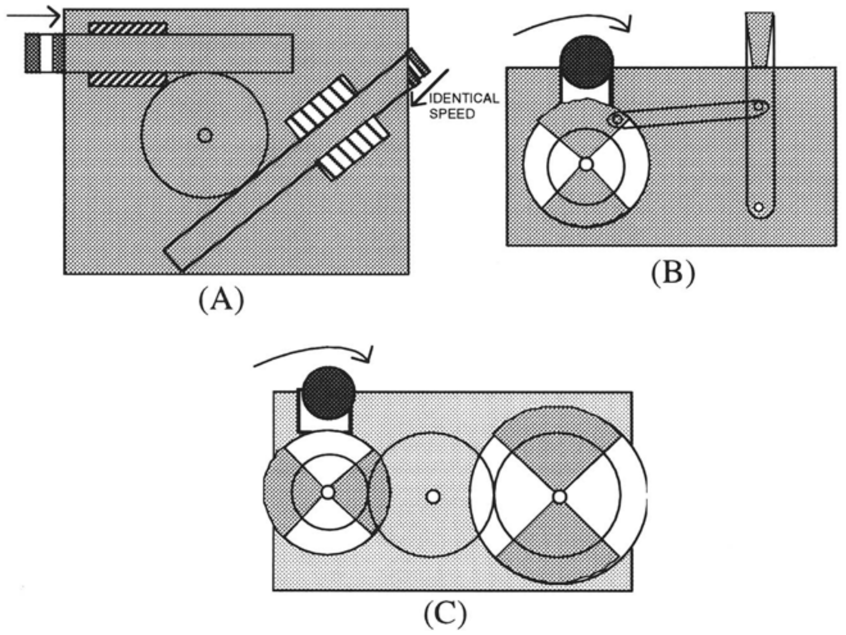


FIGURE 2.2. (A) Push-rod device; (B) rotary device; (C) osciliatory device.

Results and Discussions

Accuracy

The regression analysis was performed with the subject percent correct score as the dependent variable. An inspection of type III sums of squares revealed that all main effects were significant ($p < 0.0001$). More important, for the subject categories, the overall accuracy for all device types increased linearly from 56% correct for novice subjects to 76% correct for the experts, as is shown in Figure 2.3. This confirms the prediction that as expertise increases so does the ability to indicate accurately whether the static device is the same as the animated device. Furthermore, it was found that rotating devices were easiest (80% correct), oscillating devices were of intermediate difficulty (63% correct), and push rods were the most difficult (55% correct). This effect may be due to the relative frequency of use of the device types in natural settings (i.e., rotating devices are more frequently employed in mechanisms than either oscillating members or push-rod devices). Nonmatching problems were more difficult (60% correct) than matching problems (73% correct). The subject category by device indicated that undergraduate and graduate mechanical engineering students performed equally well on oscillating devices. However, in the other two device categories, accuracy increased monotonically with increasing expertise.

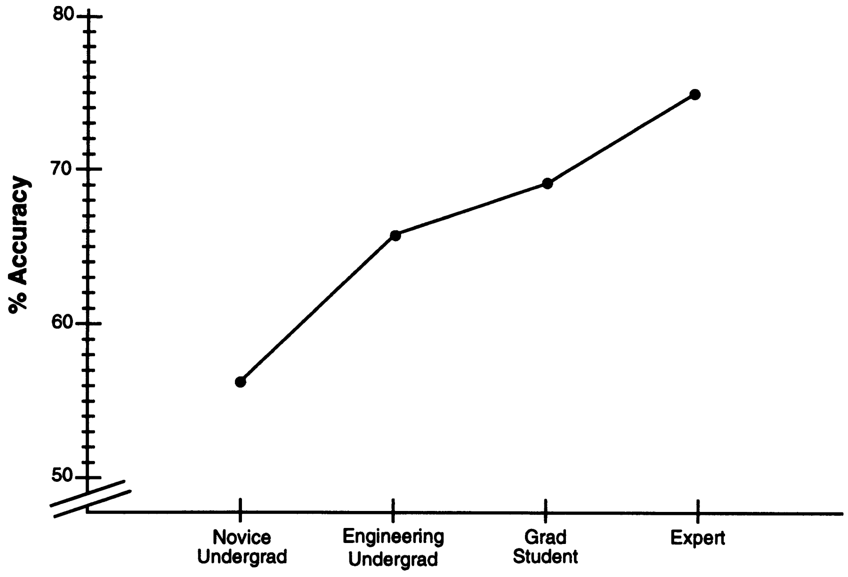


FIGURE 2.3. Accuracy by subject category.

Reaction Times

Reaction times averaged 16.75 s overall, which is relatively long, indicating that the subjects took the task seriously and that the task was reasonably difficult. All main effects in the regression analysis were significant. The reaction time increased from novices to graduate students, but dropped off somewhat for experts. This can be explained by examining the accuracy and reaction time in conjunction. Novices responded more quickly, but nearly at chance level (56%) indicating that guessing was used. Experts responded more quickly (20 s vs. 22 s) and accurately (76% vs. 69%) than graduate students, thereby showing a greater engagement in the process.

Multidimensional Scaling Analyses

In order to determine the features of the devices to which members of the different subject groups attended in making their judgments, matrices representing dissimilarities among the devices within each device type were entered into a nonmetric multidimensional scaling. This technique produces a spatial solution representing psychological differences as distances between points, where the points represent devices.

Overall, the multidimensional scaling solutions indicated that subjects *attend to the relative speed* (range of oscillation indicating how quickly the device oscillates) *and direction of motion of the devices in order to make their judgments*. In the task used in this study these features were most prominent as indicators of the match between the static and animated devices. The subjects discriminated more accurately between devices on the basis of phase relations. Correct differentiation between devices because of phase difference is correlated with expertise in mechanical engineering. Notice that speed and direction are functional, rather than spatial, features of the devices, and thus experts should be more sensitive to these features in their judgments than novices. This was tested by using regression analysis. For both rotating and oscillating devices it was found that the accuracy increased with expertise as predicted (means of 46.2, 58.3, 66.6, and 76.9 across levels of expertise), indicating that for these device types experts attended more to the relative speed (hence function) of the input and output components than novices. Whereas, when the difference in the appearance of the internal mechanisms was the best indication that two devices differed, the novices performed more accurately, thus indicating that they were more attentive to the form of the device than experts.

The level of accuracy in the task is positively correlated with the amount of experience with the domain. Not only does accuracy increase as expertise increases, but it also increases as a function of device type. The more experience one has with the devices the better one performs. Hence all subjects performed better with rotating and oscillatory devices than with push-rod devices. The results of this study also shed light on the development of expertise. Highly accurate performance on this perceptual task is directly related to the amount of mechanical design experience. Subjects with little or no experience often resort to what appears to be a guessing strategy for task performance. Novice undergraduates only perform better than chance on rotating devices. Engineering undergraduates and graduate students perform more accurately than chance on both rotating and oscillatory devices, but not on push-rod devices. Experts perform better than chance on all devices. *Experience with mechanical design and mechanical drawings has a direct correlation to the skills of visualizing and simulating the internal workings of 2D drawings.*

The results of analysis also suggest that experts attend less to the form of the device to discriminate than novices. The internal components of the third oscillatory device look very different than those of any of the other oscillatory devices. Novice undergraduates performed better on judgments involving the static version of this device than on any other type of judgment. In fact, they performed better than experts on these judgments. While novices correctly use the configuration of the internal components of this device to differentiate it from other oscillatory devices, they perform more poorly than experts, when form information does not discriminate between devices.

The question of how experience influences this task has a complex answer. Clearly, none of the subjects had ever been exposed to the specific 2D devices included in this task. Experts have examined a wider variety of devices and therefore can find an appropriate analogy to the device and use that information to simulate the operation of the device in order to make a judgment. It appears that the perceptual units upon which they infer an analog are the speed and phase relations of the devices.

Differences in Reasoning About Drawings

In this section we discuss the differences in the knowledge and reasoning used by designers with different expertise in interpreting mechanical engineering drawings. Engineering drawings provide a coded, schematic representation of a mechanical design. These established codes are intended to communicate to others the spatial and functional aspects of the conceived design. It is from these drawings that important manufacturing information is derived. This interpretation issue was studied by presenting schematic drawings to mechanical designers with different levels of expertise. The following work is largely drawn from the papers by Waldron et al. (1989) and Chovan and Waldron (1990).

Method

Thirteen drawings of various mechanical assemblies were used in this study. The drawings were presented to the subjects in three forms: (1) the line drawing alone; (2) the line drawing with a descriptive title; and (3) the line drawing with a descriptive title and a brief written explanation. The drawings were carefully selected with different levels of complexity. These drawings included several types of couplings, valves, motors, and other mechanisms.

All 13 drawings were presented to subjects in three groups, namely, expert (six professional designers), semiexpert (seven graduate students, specializing in mechanical design) and novice (six undergraduate non–mechanical-engineering students). Verbal reports were collected while the subjects were viewing the drawings. The subjects were asked to describe what they saw in each drawing, and if they knew the function of the device. Their responses were videotaped. Each subject's videotaped protocol was transcribed and analyzed for every drawing. Those design features, subassembly functions, and overall function of the mechanism represented in the drawings that were either included or excluded by the subjects in their responses were tallied by drawing for each protocol. Two types of reasoning methods were identified, namely, *inverse* reasoning (where the subject starts by recognizing the overall function and, thereby, forming a hypothesis that can be tested by examination of component parts and their functional relationships) and *forward* reasoning (where the subject starts by recognizing one or more components and proceeds by reasoning about the formal or functional relationships of the components, thereby deducing the function). The identification was taken to be correct if the subject explicitly or implicitly identified the components and an overall function that was similar to the one intended as identified by the title and description.

Results and Discussion

Analysis of variance showed significant differences ($p < 0.0001$) between the groups with the mean recognition rates being 67% for experts, 47% for semiexperts, and 8% for the novice subjects. There were significant differences in the reasoning methods used ($p < 0.0001$) with inverse reasoning used by experts 75% of the time, by graduate students 33% of the time, and by novice subjects only 27% of the time. This is shown in Figure 2.4. All subjects were more successful at recognition when they used inverse reasoning. Furthermore, experts used all the component functions and subfunctions present in the diagram in their verbal reasoning, whereas novice subjects used few or none. Experts initially perceived the overall function, semiexperts perceived the features in the drawing, whereas the novice focused on the geometry of the drawing.

Each diagram was analyzed to enumerate all the functions and subfunctions used by each subject to further their reasoning, and the energy sources used were identified. The results were subjected to discriminant function analysis to uncover by which aspects (function, subfunction, reasoning, design feature, etc.) the groups could be distinguished. We were thus able to identify the variables that contributed significantly to the distinction between the groups. We were able to establish, through this analysis, that for those drawings that represented devices more likely to have been experienced by the subjects, including functions such as valves and differentials, the semiexperts behaved more like the experts. They used inverse reasoning and the functions and the subfunctions in their analysis. For the drawings of less-often-encountered functions, such as cams, special couplings, and a spring

Recognition Rate (%)

	a	b	c
Drawing type	Fig. only	Fig. with label	And with explanation
Experts	67	80	85
Graduates	47	54	62
Novices	8	38	58

$p = 0.0001$

Inverse Reasoning

	Experts	Graduates	Novices
Use rate (%)	75	33	27
Success rate (%)	80	50	21

Forward Reasoning

	Experts	Graduates	Novices
Use rate (%)	25	67	73
Success rate (%)	30	25	6

$p = 0.0001$

FIGURE 2.4. Recognition rate and the reasoning types for the three groups.

motor razor, the semiexperts behaved more like the novice subjects. Whenever reasoning was a significant contributing factor, the experts always used backward reasoning, whereas the other groups did not. This study strongly suggests that the meaning of the drawing varies significantly with the design experience of the subject. To the experienced designer, the drawing indicates its intended function but not to the inexperienced person. The function is recalled when the drawing is viewed, suggesting that memory is organized to retain the function in association with the drawing. Inverse reasoning suggests efficiency of storage.

Differences in Designing

The purpose of this study was to perform a systematic study of designers with different expertise engaged in a robotic manipulator arm design activity. Based on the analysis, it was proposed that the expertise of the designer can be decomposed into a knowledge (knowing) and a skill component (doing). The skill is responsible for providing the designers with a feel for their design and is used extensively in error checking. This section forms a summary of the paper by Waldron (1988).

Method

A descriptive statement of a problem requiring subjects to design an industrial robot manipulator arm was provided to subjects. They were asked to carry out the design and to give verbal explanations while they designed. The depositional method (Chapter 3) was used for data collection. The designers were videotaped while they designed and talked about their designs thus providing information about the design process they followed. No time limit was set for the task. Subjects were free to take as long as they wished. The subjects consisted of eight male designers with different levels of experience designing manipulator arms. Five had more than 10 years of industrial experience; two of the five had recently designed a similar manipulator. Three subjects had less than 2 years of experience, and one of these had no practical experience.

The videotaped design protocol was transcribed. The utterances, gestures, and the items the designers used during design were annotated. From this we attempted to answer the question, how does the designer perceive what is required of him/her?

Results and Discussions

The data analysis showed that the experienced designers relied more on commercial catalogs, whereas the inexperienced designers relied on texts and handbooks. Recency of designing similar objects decreased the initial problem set-up time considerably (75% in fact). All designers attempted to find the total peak torque requirement—a principle taught in robot design courses. The average time for obtaining the torque values was 6.8 s for experienced designers, which was significantly different from 48.3 s for the less experienced designers. This difference was largely due to the simplifying heuristics employed by the experts to arrive at static and dynamic torque.

The experienced designers bring to the design heuristics that are based on seemingly "simple," yet powerful, analytical models in the early stages of concept development to set basic parameters and make configurational decisions. The more experienced the designer, the "simpler" is the model he used. This supports the earlier observations of Waldron and Waldron (1988) that the inexperienced designers have difficulty in applying the appropriate analytical model to the design in the conceptual stage.

Furthermore, all designers but one (who had no practical experience) spread their work out on the table and referred to it spatially while working.

Experience lends itself to opportunistic behavior in which the designers were willing to commit to a procedure and patch if they ran into difficulty. The less experienced designers relied on a systematic textbook approach that required major changes in the procedures when difficulties were encountered. Designers with practical experience tended to set design time as a major design constraint, and physical analogs were important to them when conceptualizing a solution.

Implications for the Design Process

In Developing CAD Tools

The results summarized above strongly indicate that the interpretation, and the manner of interpretation, of the drawings vary significantly with design experience. Hence, drawings carry more than structural information to the experienced designer. If the hypothesis that engineering drawings are the most common mode of communication of information between the design and manufacturing communities is accepted, then this study indicates that any intelligent computer-aided design (CAD) system that stores the designer's drawings must also carry with it some information about the level of the designer's experience as well as the functional decomposition and function in order to structure mapping of the particular design. CAD systems that adapt to the expertise of the designers may thereby be of use to both experienced and inexperienced designers.

In designing CAD tools, it is important to keep in mind the visual–spatial orientation of designers by providing them with spatial as well as sequential access to their designs. Since the initial interpretation is important in determining how quickly and well the designs will get done, it is important that the design heuristics of very experienced designers, in as wide a variety of design situations as possible, be archived in order for successive design times to be reduced and the quality of the designs improved.

Such smart CAD tools could carry out routine computation for designs of devices with which the designer is already familiar. These systems would have certain functional and subfunction hierarchies as well as subfunction to structure mapping, so that the designer can guide the system in functional decomposition. Likewise, such decompositions can provide large relational databases for searches for new designs to accommodate varying degrees of expertise possessed by designers. These systems would be of great benefit in teaching design. The design students could gain experience in designing by experiencing different designs.

In Education

The above studies indicate a strong correlation between the designers' experience and the information to which they attend, and how they reason about this information. The results presented above have implications for education and research. Since the efficiency of information handling by mechanical designers is dependent on their level of expertise, these types of perceptual tasks potentially could be used to evaluate the adequacy of an educational curriculum. A perceptual task could be used to identify skill within a design domain. Visual–spatial recognition and reasoning skills are very valuable skills for designers. The above studies link these skills to the designer's experience in designing. Currently, no college or professional entry examination tests these skills. Exams presently used exclusively test analytical and verbal comprehension and reasoning. While these skills are important, the above studies show conclusively that design potential and ability are highly related to visual–spatial perception. This important skill is ignored both in curriculum development and in testing.

Design is learned by designing. No verbal or analytical presentation by itself can teach the perceptual skills necessary to design. Apprentice mechanical designers need to attempt and encounter a wide variety of designs in order to increase their knowledge of artifacts and to enhance their visual–spatial skill. Experienced designers combine appropriate analogies of previously encountered designs and simulation to obtain new designs. Inexperienced designers need to increase their repertoire. The adaptive CAD system proposed could provide the information for an intelligent CAD instructional system, which could be used by design education programs to enhance design education.

In Industry

Designers are visually and spatially oriented and like to be able to review their work as they proceed through their design. The procedural sequence per se is not of great significance, but the results obtained are. Time is an important parameter through which designers organize and plan their design process.

The experience in a particular type of design has a direct impact upon reducing the time used to set up the initial analytical model to be used. The experience is further directly related to the sophistication of the model, which manifests itself in the form of strategic, but physically simple, models, based on powerful heuristics. Initially, exact computed numbers are not crucial, but a ballpark figure is required to find the class and type of components that would be directly applicable. Mechanical designers seem to depend on visual and haptic associations (their own arms in the manipulator design example) to see if the design will be successful or not. Hence, such simulation tools that allow visualization and heuristic computations are necessary for designers, and clearly mandated time limits for the design need to be specified.

In selecting teams to carry out new designs, it is important to include a designer experienced in the particular design domain. The intent is that this designer will initially establish the heuristic models from which the less experienced designers can work. The availability of product catalogs is very important to experienced designers in the conceptual design stage.

References

Adelson, B. (1981). Problem solving and the development of abstract categories in programming languages. *Memory and Cognition, 9*, 422–433.

Adelson, B. (1984). When novices surpass experts: The difficulty of a task may increase with expertise. *Journal of Experimental Psychology: Learning, Memory, and Cognition, 10*, 483–495.

Akin, O. (1978). How do architects design. In Latombe (Ed.), *Artificial Intelligence and Pattern Recognition in Computer Aided Design*, IFIP North Holland.

Chase, W. G., and Simon, H. A. (1973). Perception in chess. *Cognitive Psychology, 4*, 55–81.

Chase, W. G., and Ericsson, K. A. (1981). Skilled memory. In J. R. Anderson (Ed.), *Cognitive Skills and Their Acquisition*. Hillsdale, NJ: Erlbaum.

Chovan, J. C., and Waldron, M. B. (1990). The use of function and form when reading mechanical engineering drawings by experts and non-experts. *Proceedings of the Second International ASME Conference on Design Theory and Methodology. Chicago*, pp. 137–143.

Cooper, L. (1983). Flexibility in representational systems. In J. Beck, B. Hope, and A. Rozenfeld (Eds.), *Human and Machine Vision*. Orlando, Fl: Academic Press.

Earle, J. (1985). *Engineering Design Graphics*. New York: Wiley.

Cunningham, J. J., and Dixon, J. R. (1988). Designing with features: The origin of features. Proceedings 1988 ASME Computers in Engineering Conf. pp. 237–243.

Cutkosky, M. R., Tenenbaum, J. M., and Muller, D. (1988). Features in process-based design. Proceedings 1988 ASME Computers in Engineering Conf. pp. 557–562.

Dixon, J. R., Cunningham, J. J., and Simmons, M. K. (1987). Research on designing with features. IFIP WG 5.2 First International Workshop on Intelligent CAD. Cambridge, MA, Oct. 6–8.

Hegarty, M., Just, M. A., and Morrison, I. R. (1988). Mental models of mechanical systems: Individual differences in qualitative and quantitative reasoning. *Cognitive Psychology, 20*, 191–236.

Luby, S. C., Dixon, J. R., and Simmons, M. K. (1986). Designing with features: Creating and using a features data base for evaluation of manufacturability of castings. Proceedings 1986 ASME Computers in Engineering Conf. pp. 285–292.

Neves, D. M., and Anderson, J. R. (1981). Knowledge compilation: Mechanisms for the automatization of cognitive skills. In J. R. Anderson (Ed.). *Cognitive skills and their acquisition*. Hillsdale, NJ: Erlbaum.

Newell, A., and Simon, H. A. (1972). *Human problem solving*. Englewood Cliffs, NJ: Prentice-Hall.

Reitman, J. S. (1976). Skilled perception in go: Deducing memory structures from inter-response times. *Cognitive Psychology*, *8*, 336–356.

Stauffer, L. A. and Ullman, D. G. (1988). A comparison of the results of empirical studies into the mechanical design process, *Design Studies*, *9*(2), 107–114.

Waldron, M. B. (1989). Observations on management of initial design specifications in conceptual mechanical design. *Engineering Design*, Vol. 1, Proceedings ICED 89, Harrogate, pp. 189–201.

Waldron, M. B., and Herren, L. T. (1994). The effect of expertise on judgments requiring spatial reasoning. Unpublished paper, The Ohio State University, Columbus, OH.

Waldron, M. B., Jelinek, W., Owen, D. H., and Waldron, K. J. (1987). Strategy differences between expert and naive mechanical designers. In *ICED Proceedings 1987*, Boston, MA, pp. 86–94.

Waldron, M. B., and Waldron, K. J. (1988). A time sequence study of a complex mechanical system design, *Design Studies*, *9*(2), 95–106.

Waldron, M. B., Waldron, K. J., and Abdelhamied, K. (1989). The differences in reading schematic drawings of mechanisms by expert and naive mechanical designers. *Proceedings of the First International ASME Conference on Design Theory and Methodology*. Montreal, pp. 15–20.

3 Methods of Studying Mechanical Design

MANJULA B. WALDRON AND KENNETH J. WALDRON

Abstract. In this chapter we present different methods used for studying mechanical design, including the process of creating designed artifacts. If we wish to understand fully the manner in which successful products are created and design specifications are converted into information for manufacturing and use, then we need to comprehend better the design process that creates them. While successful engineering products have been designed for centuries, understanding the process of this conversion has only begun. This has been largely motivated by a need to create a more efficient process to decrease time to market, requiring better design tools to assist designers in coping with the increasing informational demands placed upon them. The methods of study naturally involve interactions with designers or design teams, be it through case studies, interviews, or observations. In this chapter we survey different methods currently in use to study this process and provide the context in which each method can be used.

Introduction

The development of design theories, or the creation of effective tools and techniques for design engineers, requires one to examine not only the characteristics of the designer but also of the design process that created the designed artifacts. One must understand what knowledge is brought to the process by the designer, how it is applied, and in what manner the design process progresses from initial specifications to the final artifact. Designers are an integral part of the process and are the source of information about the knowledge and the rules that are applied to arrive at the designed artifact. Thus, the output of the study of the design process is information that must, of necessity, be obtained from the designers who actually produced the design. The goal of this chapter is therefore to discuss the methods in use to put the designer in the driver's seat of creating computer-aided design (CAD) tools. The chapter does not discuss the techniques or the computer tools used for the representation of the designed artifacts. These aspects are dealt well in other works, for example, Tong and Sriram (1992), Sriram and Adey (1987).

The design process consists of the evolution of the artifact in time including the plans for manufacture, use, and (of late) disposition of the artifact. It has an initial input in the form of functional descriptions, specifications, etc., which form the starting point of any design process. This input may be technically vague, in the form of goals rather than specifications, or very tightly defined in terms of physical constraints and materials to be used, etc., providing very little flexibility of approach. The final product consists of the designed artifact that is capable of performing the originally specified functions or attaining the stated goals. Everything which occurs between the specification and the production of the artifact is then part of the process. The development of the design is dependent on physical laws, the environment in which the design must function, the designer's experience in the domain, and the collective knowledge of all the persons active in the process. In other words the design process takes the required *function* of the design in the environment in which it must work, and transforms it into the *form* of the designed artifact utilizing the knowledge of the designers in the process (Waldron, 1989).

The only successful designers available for study are humans, and a considerable part of the information they use is not readily accessible. The study of the design process therefore, of necessity, requires some study of human behavior. The tools and techniques available for this must be borrowed from observational and social psychology. These tools are often subjective and require techniques different from those usually encountered by engineering students and faculty. Because of this, the study of the design process is considered to be more of an art, and traditionally trained engineers may find these techniques difficult to accept. However, their acceptance by the artificial intelligence and expert systems communities has provided credibility to their application to engineering design. In fact, there is a definite recognition that engineering education must move back from engineering science to engineering practice, which must include the qualitative methods developed by social scientists in addition to the quantitative analysis of physical sciences (Vest, 1994).

There are two basic types of methods used to study the design process: one in which the researcher is in control, the other in which the designer is in control. The first type includes methods based on interviews. In the second type the researcher works with the designer. This type includes process tracing methods.

Process tracing methods allow development of an understanding of the problem-solving processes of designers. The procedure is concerned with the type of information used and its function in the design process while arriving at the design. A representative, or sample design, is studied to obtain information about the process, so that tools for assisting designers to design more efficiently and naturally can be developed (Waldron, 1985). Thus, this methodology has the potential to allow the consumers (designers) to drive the CAD development process in a manner that is natural to them. Examples of process tracing methods are protocols, depositions, retrospective studies (including case studies), and observational studies. These methods are described in detail in the following sections.

Interviews

The methods discussed in this section are more suited for knowledge elicitation from experts than for the chronological study of the design process per se, because these methods provide information about functional relationships and rules, and about the rationale behind different rules, but may not provide accurate chronological information about how the design may have proceeded. Nevertheless, these are important techniques traditionally used to study human behavior and develop case studies and expert systems. We feel that it is important to include these techniques. In addition to ensuring the historical completeness of this chapter, the interviewing techniques provide a powerful combination when used with protocols, as is noted in the discussion of the depositional method.

Structured interviews have a fixed schedule of questions. Therefore, the researcher must first develop an expertise in design and the design process through prior review of pertinent literature (Boose, 1989). The questions can be closed, open, or probing. Care must be taken to make the questions clear and unambiguous and not to make them leading or loaded. The designer can also be asked to scale responses. McCracken (1990) found the use of unstructured exploratory interviews with experts in the design process to be very important in facilitating the process of information gathering. He argues that for large projects, a series of interviews, both structured and unstructured, may be useful. The researcher must understand the design sufficiently to ask relevant questions of the designer. The advantage of this method is that the researcher can focus on direct acquisition of the information of interest to him or her, thus reducing the time needed for data analysis. Following preliminary informal study and knowledge gathering, the researcher sets up the fact finding interviews (Hart, 1986; Waldron, 1986). The interviews are often used in initial sessions with the designer, and are of three types—descriptive, problem-oriented, or directed.

Descriptive interviews represent the earliest method and predate the convenient video recording techniques available today. Initially, the designer gives a lecture to familiarize the researcher with the domain. Then the designer completes a worksheet describing the important characteristics of the design and a step-by-step description (as opposed to a chronological ordering) of the design process. This method provides only an idealized, textbook form of design and may omit critical strategies for decision making.

In the problem-oriented interviews, the researcher presents a design problem to the designer and asks a predetermined set of questions at each step of design process. These are structured by the researcher to follow a predetermined plan of questioning, such as how and why certain decisions were made, and verifying the context in which these were made for a given design. The structure of knowledge the designer uses can be identified, but it is still limited by the researcher's questions.

In the directed interviews, the interviewing technique depends on the type of knowledge representation to be used. The frame representation is a common knowledge representation scheme. Each frame consists of a series of slots, each of which represents a standard property or attribute of the element represented by the frame. A slot is identified by a name associated with the attribute. It can have default values specified or can include a range of values for the attribute. Each slot can have procedural information attached to it, indicating an action to be taken if certain values of the attribute are obtained. The researcher creates a set of frames for the knowledge domain and presents them to the designer to help fill in the slot values and procedures associated with the information in the slot. In this method, the familiarity of the researcher with the domain limits the knowledge that the designer provides and critical information may be missed.

Both of these methods have the advantage of allowing the researcher to direct the information acquisition process, so that the data are easy to analyze. However, they are both sensitive to the capabilities and the time availability of the researcher and the designer. These methods are dependent on the researcher's grasp of the subject, and may not be a true representation of the design process. If the researcher is ill prepared, or does not have good communication skills, then the whole study is in jeopardy. While authors such as Hart (1986) have provided general guidelines for conducting good interviews, descriptive interviews still suffer from being subjective, haphazard, and inefficient. Nevertheless, interviews provide a powerful fact-finding tool for obtaining information from designers regarding their reasons for taking a particular course.

Protocol Analysis

A protocol is defined as a description of the activities (ordered in time) in which a subject engages while performing a task. The power of a protocol lies in the richness of its data. Even though protocols may be incomplete in themselves, they allow an investigator to see the cognitive processes by which the task is performed. The manner in which this process is ordered in time and the cognitive behaviors exhibited in performing the intended task can also be examined (Ericsson and Simon, 1984). There are two types of protocol methods proposed, namely, verbal or think-aloud protocols and discussion protocols.

Verbal or Think-Aloud Protocols

This method is described in great detail by Ericsson and Simon (1984). Several researchers have used this method to study engineering design process (Ullman et al., 1988; Waldron, 1989; Christiaans and Dorst, 1992). In this method, the designer is asked to speak aloud while designing. The expert's solution process is recorded on an audio- or video-tape recorder. The role of the researcher during data collection is to ensure that the designers continue to think aloud if they should lapse into silence. Ericsson and Simon (1984) suggest several techniques including training for increasing verbalization. The researchers are not allowed to interrupt the problem-solving process, and should be as unobtrusive as possible. The recorded session is analyzed by the researcher for facts and procedural and judgmental knowledge.

The basic assumptions of protocol analysis are that the subject's behavior provides information about his or her design knowledge. Each step observed provides information about the task-relevant operators and the knowledge available to the subject. The verbal reports are a reflection of the information the subject holds in his or her short-term memory, including information about the goals and subgoals, and the operators that bring new knowledge to the short-term memory.

The analysis procedures proposed by Ericsson and Simon (1984) include transcribing the tapes and the breaking the transcription into meaningful episodes or segments. An episode is an aggregate level of problem solving introduced by Newell and Simon (1972), while a segment is at a finer level and is a statement (Ericsson and Simon, 1984). Each episode or segment is coded, with or without a context, to extract the knowledge, relationships and reasoning used by designers. The protocol contains words and expressions that are indicative of what may be occurring. Coding of the data may be done by identifying following types of statements:

Intentions, goals, or ideas, which can be recognized by verbs such as "must" or "have to."

Cognitions, or the current situation, which are recognized by sentences indicating immediacy.

Planning, or sequences of possibilities are explored and include constructions like "If *x*, then."

Evaluations, or comparison of alternatives, which are recognized by words like "yes," "no," "fine," or expletives.

For mechanical design, researchers have identified other items such as drawings, gestures, simulation, constraints, and process and design decisions (Ullman et al., 1987; Waldron et al., 1989).

The advantage of this method is that the designer guides the design and the information collected provides information for a more normal task environment. The researcher must have acquired some background knowledge of design to organize the data into meaningful episodes and relationships.

This method of generating and analyzing protocols has the distinct advantage of obtaining a designer's solution as it occurs. It helps in the collection of information that is accurate and that truly reflects the designer's chronological approach to the design. The researcher can study the protocol and arrive at the information structure at leisure.

This technique nevertheless has disadvantages: Verbalizing knowledge during designing may interfere with the task, or it may alter the expert's usual approach. Protocol analysis is also difficult and time consuming. Analysis requires a theoretical framework and a coding scheme for categories and types of information. The researcher should be skilled in the area of protocol analysis. Tracing the entire design process may not be feasible, particularly if the expert must think about the solution, and the solution may take days or months, as is the case in practical design tasks.

Discussion Protocols

Discussion protocols are more commonly used in the analysis of group design activity. In this method, two or more designers engage in design through discussion. These discussions are recorded for use in analysis. The discussion provides the researcher with extra information about the design from two perspectives: the decision-making alternatives and the strategies for resolution of conflicts during design. The analysis procedures are similar to those described above for use in think-aloud protocols. While this method has advantages, such as that of obtaining more general and diverse designs, it also has disadvantages. The designers discussing the problem may have a mutual vocabulary and shared knowledge that may not be verbalized during the discussion, and basic information may not surface in the discussion since it is assumed in their higher-order understanding. These techniques have been used in studying mechanical engineering design by Tang and Leifer (Chapter 5) and Nagy et al. (1992).

Depositional Method

In some areas, such as the study of conceptual mechanical design, the think-aloud methodology may actively impede the designer's thinking. The designer may be literally unable to think and talk at the same time, thereby forgetting to verbalize. If the designer forgets to verbalize this information, then it is lost in the verbal think-aloud protocol. The researcher's prompting may only produce partial information utterances.

In the depositional method proposed by Waldron (Waldron et al., 1989), the best parts of the protocol and interview methods are combined. Here the designers still have control of the process, but must provide the researcher with a rationale for their actions and describe what they have done at conveniently chosen intervals. The researcher is free to interrupt if the designers should forget to make their deposition. Thus, the interviews are structured by the design activity. At each step in a design the decisions and strategies used to arrive at the result are theoretically available. The analysis of the depositions (see example below) provides information about the design goals, strategies (St) used, process (Pd) and design decisions (Dd) made, the actions (A) carried out, the given and/or generated constraints (GC), negotiated constraints (NC), and the skills (S), knowledge, and metaknowledge (Mk) used. This method was used to acquire conceptual design strategies and provided a means to study expert novice differences in conceptual design (Waldron and Herren, 1992).

For example, a depositional protocol from a person designing a robot manipulator arm was analyzed. The result of the analysis is shown below. It can be seen that a goal is followed by a strategy, which is followed by a process decision, which is followed by an action based on skill or knowledge. This temporal sequence was observed in the depositions analyzed from seven designers of varying degrees of expertise.

A sample depositional protocol analysis of data from the manipulator arm design problem:

```
A: Read problem
  Goal 1: Conceptualize the problem
      St: integrate the available information
      Pd: relate text to diagram
          A: look at diagram
          A: reread problem
          A: write DIMENSION-RECALL
      Pd: seek additional information
          Mk: doesn't understand the meaning of outer arm is a square tube
          A: uses diagram to resolve
      Pd: list questions about problem
          S: metal, plastic, or alloy materials possible
          S: mechanical, hydraulic power available
          A: turns back and forth between diagram and description
      Pd: try to figure out where to start
          Mk: mind flips between range of motion, speed it has to operate, and
              actuation and sensing systems
          Mk: not sure what "motion of the joint should be resolved to 2 min of
              arc" means
          GC: BC has to be able to swing
          Mk: doesn't know why elbow joint needs sensing system
          A: looks at diagram
          S: it would seem the sensing system would be allocated at the gripper
      Pd: read problem for info. about sensing system
          A: reads problem
```

This technique has the advantage of traditional verbal protocols in that it captures detailed information about the designer's strategies and decisions and information about the procedures used. It provides information on design histories (sequence of choices and alternatives) and the corresponding rationale. In addition, it has the advantage of being able to track designs that take place over a period of several days. Furthermore it allows for design-oriented interviews in which the designer can clarify how, when, and why he or she made a particular decision. It can capture the strategies the designer uses to constrain the design process and information about the contexts of design decisions by encouraging the designer to report available alternatives. In this way, the researcher may identify decision alternatives that may not be verbalized during verbal protocols or discussed in structured interviews, because the designer is not in control and is therefore not actively seeking alternatives.

The disadvantage of the method is similar to that of verbal protocols; that is, the large amount of data that must be analyzed. Furthermore, interrupting the designer during the design may alter the activity. Rationalizations about decisions or strategies may not be those used originally, since the goal of the study is not to document the exact cognition of the designer but to capture information about the design process, so this is not a serious flaw. Nevertheless, when studying design, the advantage of this method in obtaining information that is otherwise unobtainable far outweighs this disadvantage.

Case Studies

This method is the earliest retrospective method. Case studies of designs were compiled by Fuchs at Stanford in the 1970s. These were used to allow design students to learn about designs, but were limited in value as far as obtaining information about the design process because of a lack of established analysis techniques. These are more useful for their archival value.

Case studies such as those by Fuchs were designs that were documented and compiled, and are now accessible to the researcher studying the design process. They often contain detailed information about final overall design, but the process information is sketchy. Likewise, the final selections are documented, but seldom do they contain the unsuccessful designs that were considered. Finally, exact chronological development of the design is not available through the examination of case studies. Hence, the sequence of relevant information and how it was used is absent. The researcher can look at the case study and analyze it based on the information contained in the document about the process. However, this can be ad hoc at best. Because case studies were often idiosyncratically documented, it is difficult to develop analysis techniques that will apply to all of them. Process observation techniques have been more successful in obtaining meaningful information about the design process.

More recently, case studies have employed three types of data sources, namely, observations, interviews, and archival documents. Researchers record the design protocol of designers while working, either by videotaping or by sitting in on meetings and observing how people interact and taking notes. These techniques, although ubiquitous in other domains, have not been used extensively in the study of the engineering design process. This may be because engineering design research began in the 1980s, when protocol analysis was more in use to develop expert systems.

In obtaining data for case studies, researchers may conduct face-to-face interviews with designers engaged in the design process and other members who are relevant to the particular design exercise. In addition, the archival data sources, such as catalogs, papers, books, and other sources used during the design process, are also gleaned for the information used in developing a particular design. Case studies are considered to be an appropriate research tool for looking at complex design processes and are therefore considered exploratory and descriptive in nature.

In a complex and long-term design process, case studies allow researchers to look at more complex questions and to answer the *how* and the *why* of the design process, particularly when they have little control over the manner in which the design will progress who the players might be, and when the designing activity may take place.

Yin (1989) lists the steps that necessary to carry out a case study. The first step is to have a theory that is being tested, and then select a design that embodies the theory under scrutiny. Next develop a design data collection strategy prior to actually conducting the research. During the research, the data are collected via observation, interviews, and archival sources. These data are then analyzed using the techniques that are usually employed in analyzing such data. Naturally, the data are qualitative, and will usually be in a text form, and may contain drawings that were used to describe the design to other team members. These may come from field notes collected during observations, as well as the transcripts of interviews which may have been structured or unstructured, that were obtained from team members involved in the design process. The qualitative data analysis techniques are based on content analysis (Weber, 1985), and domain analysis (Spradley, 1979).

A grounded theory is also sometimes used in analyzing data obtained in qualitative research (Glaser and Strauss, 1967). This analysis involves obtaining data categories of context-specific knowledge that are important and meaningful to designers involved in the design process, and allowing one to obtain relationships between the categories and the data collected through an inductive process. These relationships or hypotheses can then be tested against new design data, which in turn allows one to refine one's categories and hypotheses further.

Content analysis, on the other hand, allows one to make inferences from the data collected through using specialized procedures for processing such data. These procedures include partitioning the transcribed data into sets of units that can be clustered to define categories. These can then be examined to determine if they are indeed mutually exclusive. A coding system can be devised based on a sample of the text that can then be used to code the entire transcript. These categories are often devised to explain relationships that exist among similar data in the protocol or the transcript.

Retrospective Method

In order to overcome the noninteractive nature of case studies and yet capitalize on the wealth of already completed designs, Waldron and Waldron (1988) proposed the retrospective method in which the designers provide a detailed record of the design process through recall and reference to written records. This approach is similar to case studies, but with the addition of chronological information about the design process, and when and how major decisions were made. The designers provide the written trace suitably coded, showing the major events of the design, including the design decisions and completion of subphases, etc., which directly impact the design process. These traces properly register the times and indicate the major information pathways utilized directly and indirectly to show the factors that influence the design process including the environment. The researcher, after reviewing the material, then interacts with the designer to obtain the details of the process information. The difference between this and the descriptive interview method is the control. In this process the designer is in control and the researcher is organizing the information about the process in a manner useful for analysis and computer simulation. It does require the researcher to learn something about the domain and have good interviewing skills. Since the designers are documenting their own solution, the data will be more accurate than in the descriptive interview method.

The advantage of this method over the protocol method is that larger problems that take longer time periods to solve can be accommodated. This is possible because the real time records can be supplemented by the retrospective information from the designers or the group leaders who may be able to provide chronological records of the process. Good design practice includes the use of designer's notebooks containing and documenting all written work performed. It is common practice to date each page as it is used. The structural elements identified could be used to guide the interpretation of individual or group protocols. In content-intensive areas such as mechanical design, the interpolation of knowledge and solution may take considerable time. The integration of this information is not visible in the protocol or depositional methods, but may be identified through retrospective examination. The disadvantage of this method includes loss of memory or incomplete written records. The designer may forget to describe some aspect of the process or may rationalize it to be unnecessary. The errors and dead end paths may not be recounted. The designer may not think it necessary to provide small but important details, because these may have been negotiated automatically, with little conscious thought by the designer.

Waldron and Waldron (1988) used this technique to trace the design process of a complex design project (the Adaptive Suspension Vehicle). Using this technique they were able to identify complex decision making and information structures.

Process Observation

In this method, the researcher observes the designer(s) engaged in designing with no interference to the normal activities. He or she simply records all the interactions relating to the design. This technique provides information not obtained by other techniques, such as each person's role in the design process. The method is very useful in large industrial settings where multiple persons are involved in the design process. The researcher can set up a context for each decision and create partitioned knowledge bases for individual designers. This method may help to see how these designers interact to make decisions and solve problems.

This technique is obviously an extremely time-consuming method of process information elicitation. The transcripts require extensive analysis to glean useful information. In addition, it is a more intrusive technique, since the designer's conversation and movements related to the design must be recorded at all times, by either the researcher or the designer. This adds an element of uncertainty and may make consent harder to get from the designers.

Hales (1987) used this technique to study the design of a system for evaluation of materials in a coal gassification environment. He observed the project for 2.8 years. He recorded 1373 interchanges from 37 people covering 2368 h of work effort with resolution down to 0.1 h. His field data consisted of 76 h of audio tapes, 1180 pages of diary notes, 116 weekly notes, and 6 design reports. He reduced these data by color coding the notes according to the participants. These were entered into a computer database for analysis by indexing, sorting, and grouping of information. He found that the obvious problem in objective analysis was that many of the human exchanges do not lend themselves to objective analysis other than for time and work effort. For this reason he added the notion of context to his work effort data. By context he meant what the work was, with whom, where and when it was done, and the techniques and the tools that were used at that time. He was thus able to track the project costs with the number of persons involved, and was able to identify the types of activities carried out in addition to designing, such as planning, social contact, information retrieval, reviewing, helping others, etc., and the percent of times these were done. He identified communicating techniques to be important and found that the "mood" of the designer contributed to the design process. This method allowed him to represent this design process in the context of environment, marketing, management, and the project. He found an iterative relationship among these.

Waldron and Brooks (1994) used the sense-making methodology proposed by Dervin (1989, 1992) to analyze data obtained from a collaborative, concurrent group design through process observation techniques. They transcribed the observational protocols and partitioned the design process into a series of situations about which the designers were trying to obtain information from each other. They used a two-step coding scheme to identify inter- and intragroup information exchanges among designers. The sense-making methodology allows one to identify situations in which the designers were stopped, that is, faced a gap and sought resources or help in order to proceed. This analysis allowed them to trace the inter- and intragroup gaps, help, and decisions made. The results showed an underlying structure to the basic information exchange in the design process. The gaps faced were much larger than the help received, which were much larger than the decisions made by the groups.

Conclusions

In this chapter we have described a number of methods that can be and have been used in the study of the design process. Because of the nature of the study of the human behavior, analysis of the data collected follows a general description rather than a prescription. There are general analysis techniques, but the data in each situation will need to be partitioned in an ad hoc manner, reducing their scientific validity. However, they provide information about the design process that is not available by any other means.

Protocol analysis is the most commonly reported technique in the literature reporting the study of the design process. However each researcher has had to adapt the analysis technique proposed by Ericsson and Simon (1984) somewhat, because of the audio-visual nature of the data to be collected. However, the data collection procedure was followed as originally proposed. The depositional method was a further modification to obtain strategic knowledge from the designers. This method allows better collection of the decision-making process. In order to study large cooperative design projects, the retrospective and process observation techniques may be of use.

References

Boose, J. H. (1989). A survey of knowledge acquisition techniques and tools. *Knowledge Acquisition*, *1*, 3–37.

Christiaans, H., and Dorst, K. (1992). Cognitive models in industrial design engineering: A protocol study. *ASME Proceedings Design Theory and Methodology DE 42*, pp. 131–142.

Dervin, B. (1989). Users as research inventions: How research categories perpetuate myths. *Journal of Communication*, *39*(3), 216–232.

Dervin, B. (1992). From the mind's eye of the user: The sense-making qualitative-quantitative methodology. In Glazier, J. D. and Powell, R. R. *Qualitative Research in Information Management*. Englewood, CO: Libraries, Unlimited, Inc. pp. 61–84.

Dixon, J. R. (1988). On research methodology towards a scientific theory of engineering design. *Artificial Intelligence for Engineering Design, Analysis and Manufacturing, 1*(3).

Ericsson, K. A., and Simon, H. A. (1984). *Protocol Analysis: Verbal Reports as Data*, Cambridge, MA: The MIT Press.

Glaser, B. G., and Strauss, A. L. (1967). *The Discovery of the Grounded Theory: Strategies for Qualitative Research*. New York: Aldine Publishing.

Hales, C. (1987). *Analysis of the Engineering Design Process in an Industrial Context*. Hampshire, England: Gants Hill Publication.

Hart, A. (1986). *Knowledge Acquisition for Expert Systems*. New York: McGraw Hill.

McCracken, J. R. (1990). Questions: Assessing the structure of knowledge and the use of information in design problem-solving. Ph.D. Dissertation. Ohio State University. UMI Dissertation Information Service.

Nagy, R., Ullman, D., Dietterich, T. (1992). A data representation for collaborative mechanical design. *Research in Engineering Design 3*, 233–242.

Spradley, J. P. (1979). *The Ethnographic Interview*, New York: Harcourt, Brace, Jovanovich College Publishers.

Sriram, D., and Adey, R. (1987). *Knowledge based expert systems in engineering: Planning and design*. Surrey, UK: Computational Mechanics.

Tong, C., and Sriram, D., eds. (1992). *Artificial Intelligence in Engineering Design*, Vol. III. New York: Academic Press.

Ullman, D. G., Stauffer, L. A., and Dietterich, T. G. (1988). A model of the mechanical design process based on empirical data. *Artificial Intelligence for Engineering Design, Analysis and Manufacturing, 2*, pp. 33–42.

Ullman, D. G., Stauffer, L. A., and Dietterich, T. G. (1987). Toward expert CAD. *Computers in Mechanical Engineering*, 56–70.

Ullman, D. G., Wood, S., and Craig, D. The importance of drawing in the mechanical design process. *Proceedings 1989 NSF Engineering Design Research Conference*, 31–52.

Vest, C. (1994). Our revolution. *ASEE Prism*, 40.

Waldron, M. B. (1989). Observations on management of initial design specifications in mechanical design. In *ICED 89 Proceedings*, August, pp. 157–166.

Waldron, M. B. (1990). Understanding design. In H. Yoshikawa and T. Holden (Eds.). *Intelligent CAD II*. North Holland, pp. 73–88.

Waldron, M. B., and Brooks, R. L. (1994). Analyzing inter and intra group information exchanges in conceptual collaborative design. In *Proceedings of ASME Design Theory and Methodology Conference*, DE-68, American Society of Mechanical Engineers, New York, pp. 241–248.

Waldron, M. B., and Herren, L. T. (1992). Using depositions to acquire conceptual design strategies. Working paper, The Ohio State University.

Waldron, M. B., and Waldron, K. J. (1988). A time sequence study of a complex mechanical systems design. *Design Studies, 9*, 95–106.

Waldron, M. B., and Waldron, K. J. (1992). Knowledge acquisition methodology in selection of components in mechanical design. In Tong, C., and Sriram, D., (Eds).

Artificial Intelligence in Engineering Design, Vol. III. New York: Academic Press, pp. 57–77.

Waldron, M. B., Waldron, K. J., and Herren, L. T. (1989). Empirical study on generation of constraints which direct design decisions in conceptual mechanical design. In *Proceedings 1989 NSF Engineering Design Conference*, 15–30.

Waldron, V. R. (1985). Process tracing as a method for initial knowledge acquisition. *Proceedings of 2nd IEEE AI Applications Conference*, Miami, pp. 661–665.

Waldron, V. R. (1986). Interviewing for knowledge. *IEEE Transactions on Professional Communication. PC-29* (*2*), 31–34.

Waterman, D. A., and Newell, A. (1971). Protocol analysis as a task for artificial intelligence. *Artificial Intelligence 2*, 285–318.

Weber, R. P. (1985). *Basic Content Analysis.* Newbury Park, CA: Sage Publications.

Yin, R. K. (1989). *Case Study Research Design and Methods.* Newbury Park, CA: Sage Publications.

4
Design Characterizations

Manjula B. Waldron and Kenneth J. Waldron

Abstract. In this chapter we present a discussion of engineering design and the differences between the design as an artifact and the design as an activity. Hence, different taxonomies of design, as well as different models of the design process, are discussed. Furthermore, their relationships to intelligent computer-aided design are also discussed. The purpose of this chapter is to present different facets of design characterization as discussed in the literature.

Introduction

In characterizing mechanical design it is important to make a distinction between what the designers create, that is, the designed artifact, and the design process. The design process can be viewed as a sequence of steps, such as clarification of the specifications and the environment in which the design will function, understanding the behavior, and establishing the operational constraints, including manufacture, servicing, marketability, usability, and disposability. Or the design process may be viewed as creating conceptual design for the artifact, testing and evaluating the designed artifact, and, based on the results, refining and optimizing the design, until some satisfying criteria is reached.

The models of these processes can be created from a theory, or a set of axioms, or can be based on observations. One could define *the designer* as "somebody who creates something that will be used (practically) by others"; that is, the designer is concerned with how things ought to be in order to attain certain goals and functionality. The designer is an integral part of the design process, and the artifact to a large extent depends on the designer. Designers may work by themselves or in groups. The designs may be totally new, that is, created from scratch, or, as is more usually the case, they may be an improvement on the performance of or the modification of existing artifacts.

Today's designer faces the challenge of integrating information from many domains. There is increasing need for designers to consider the life-cycle issues during design. This requires that they have information available on the knowledge and the methods used in diverse fields. In order to transfer knowledge, so that units from different domains can access it, one needs a consistent model along which such knowledge can be transferred and the appropriate methods that can be applied in a timely fashion. These models should be capable of collective learning so that the information gained from one design process can be used again during successive design processes. The models through their predictive powers enable the units engaged in the design process to foresee problems that may arise from different possible candidate designs. In Chapter 15, Ishii discusses this aspect in greater detail.

In this chapter we describe engineering design and the classification of mechanical design, and then present characterizations of the design process by different researchers.

Engineering Design

Dym and Levitt (1991) define engineering design as a "systematic, intelligent generation and evaluation of specifications for artifacts whose form and function achieve stated objectives and satisfy stated constraints." Dym (1994) states this definition incorporates many assumptions such as that the designed representations include form and function and can be translated from these representations into a set of fabrication specifications for the production of the designed artifact. The problem-solving, design, and/or fabrication process incorporates the evaluation criteria for design.

The process of designing is different from the *design as a product* (artifact). The process is something that depends heavily on time and the units involved in creating the artifact, whereas the artifact may or may not depend on time. The process has an initial input in the form of functional descriptions and specifications, however vague, that form the starting point of any design process. The final output consists of the designed artifact, which is capable of performing the initial specified functions or attaining the stated goals. Everything in the middle is then the process that is dependent on the physical laws, the environment in which the design must function, the designer's experience in the domain, and the collective knowledge of all the persons engaged in the design process.

The artifact can be simple or complex. A simple artifact stands alone, and may have relatively simple geometry. A complex artifact may have complex geometry or it may consist of assemblies, which may in turn consist of simple artifacts. Thus, one could describe artifacts through enumerated strings and an associated grammar (Fitzhorn, 1989).

A design process has an underlying direction; it is more than trial and error. During design the designer formulates an idea, compares it to competing ideas, or tests for constraint satisfaction. Based on the evaluations, new ideas may be generated and the designer learns about the design situation, which will be actively used in the next design process. Any model of the design process must accommodate this underlying learning for it to be successful (Bucciarelli & Schon, 1987).

Models of the design process are necessary as a basis for interpreting the observations, for prescribing a design procedure, or for designing a computer system to perform the design. The models constructed require the power to describe what is observed and the what might be observed, given certain conditions. These models, if based on provisional theories, may prove faulty; nevertheless, they serve an important function by allowing the research to proceed. Modeling the design process provides us with a systematic description that can fill gaps in knowledge and enumerate design strategies, decisions and knowledge, etc. It can provide us with an understanding of how current computer-aided design (CAD) tools are being used, which can assist in the removal of inadequacies in the current CAD tools by improving the human/machine interface and providing computer aids to different stages in the design process. One can learn about the logical and spatial reasoning necessary in design and create better representation and presentation methods for designed artifacts.

Dixon (1987) described three theories of design: prescriptive, descriptive (cognitive), and computational. The study of the human designer leads to the descriptive or cognitive theories. The computational theories, on the other hand, need not depend on anything a human does.

The sequence from process to model to theory is a continuum. One can start from an abstract notion of a theory and move in the direction of the process or vice versa. No matter how one starts, the two must converge, if we want to avoid the situation where we have a useless theoretical structure, or a large amount of process knowledge without a theoretical basis.

Provisional theories could be built on empirical observation or on self-observations, although introspection on cognitive activity can be misleading. The study of the designer's behavior as a part of the design process can lead to the creation of tools that can augment the designer's thinking. Such data can provide information on the types of environment that will be useful to make the designer more efficient. Based on these theories models that represent the design process can be created. These models assist in creating better CAD tools. These tools could be purely for drawing, or creating the geometric or form representation of the artifact, or, as more knowledge-based tools have become available in recent years, they can represent the function of the artifact, which is often termed intelligent CAD (ICAD). As more persons become involved in creation of the design, the CAD models become increasingly complex and the information management becomes crucial. Thus, ICAD tools need the support of work flow management tools as described by Ramanathan in Chapter 16.

Classification of Engineering Design

Dixon (1987) argued that a classification (called taxonomy) of design may lead to relationships among different design models due to certain shared characteristics. This may facilitate the manner in which underlying knowledge and its representation may be organized to create a scientific theory of design.

Classification of Mechanical Design Problems

Dixon et al. (1989) proposed that at the highest level of abstraction one could view the design as consisting of an initial and a final state of knowledge. Furthermore, each state could be characterized by knowledge types, such as perceived need (for the initial state that is the motivation), function (of what it does), physical phenomenon (or underlying principles), embodiment (concept), artifact type (attribute), artifact instance (value,) and feasibility (for the final state). As an example, choose the initial state of knowledge of design of a plug. The need is to fill a hole. The function is to stop a leak. The phenomenon is friction. Embodiment might be a wood peg in the hole. The artifact type may be a cylindrical piece of wood, and its instance may be of dimensions $\frac{1}{4}$ in. diameter, 1 in. length.

Ullman (1992a) extended this classification to include the design environment and the design process in addition to the artifact design. The process transforms the initial state into the final state, and the environment stipulates the constraints on those doing the design. Hence, the environment consists of the participants, resources, and design specifications. The problem consists of the initial and the final state representations and the satisfaction criteria. The process consists of the plans, the action, the effect, and the evaluation or the failure criteria.

In addition, Ullman proposed the representation languages to be graphic (drawings), numerical (computations), textual (writing), and physical (prototypes). For the process the plans and processing actions could be fixed, selected from a list, parametrized, or searched. The effects could be refinement (improvement on the initial state), decomposition (breaking the design into smaller designs), or patching (modification of the design based on some failure criteria).

Routine, Innovative, Creative, Conceptual, and Selection Design

Brown and Chandrasekaran (1985), in examining problem solving, observed that the knowledge acquired from previous experiences simplifies the task of designing. These experiences may make the design fairly routine, where no new knowledge is involved, but new configurations of existing designs may be sought. They proposed three classes of design. Class 1 designs are those that require substantially new knowledge or are creative in nature. In Class 2 or variant designs some knowledge preexists, while Class 3 designs are routine. This type of design classification will be further described.

In *routine design*, all possible solution types are enumerated (that is, all the attributes, applicable useful methods, and the strategies are known *a priori*). The goals are fixed; hence, no overt planning on the part of the designer is required. The values of the individual variables may change, but the ranges and types of variables do not. Hence, these types of designs lend themselves well to the creation of knowledge-based expert systems. Brown and Chandrasekaran (1989) report an expert system to design an air-cylinder using the concept of routine design. In Chapter 9, Brown discusses routine design in greater detail.

In *innovative design*, the knowledge base is already known and available to the designer. The requirements of the design are the same, but its application may become different; hence, the solutions are novel, but no new knowledge is added. This kind of design requires planning, since new strategies may be employed and possible solution types may be novel; hence, all attributes of this design may not be known *a priori* and some of the values of the variables may be outside of the normal range.

In *creative design*, on the other hand, neither the attributes nor the strategies are known ahead of time. Once these are known, the design becomes "innovative" or "routine." If, in a design, new variables and new knowledge bases are added to the system, then this design becomes *creative* design. As soon as the design is completed, it becomes *innovative* or *routine*.

Based on individual experience, a designer may be able to go directly from function to some structure because of having encountered similar conditions previously; hence, the design is routine to that designer. Another designer, who is not experienced with this function-to-structure mapping, may have to go more deeply into the decomposition before a structure can be identified. For that designer, then, this design is not "routine." Whether or not a design is accepted as "creative" is a social activity. However, for the person doing the design, if they have done it without having prior knowledge, and if they have added new variables, then to that person the design is "innovative" or "creative," whereas to society it may be just "reinventing the wheel," and is therefore a routine design. For example, the design of the first automobile was creative, or a new automobile design may be creative, but the millionth automobile of the same kind is not. Nevertheless, for an individual, for whom this is a new project, the process of creating the millionth automobile may be creative. Hence, there is a tremendous difference between social and personal creativity and it may be dependent on the environment. For this reason this classification scheme has limitations.

To clarify this terminology further, it is worth noting that the *conceptual design* is the initial design that is different from routine and creative design. The conceptual design can be studied and modeled. One could have a very routine conceptual design when the specifications are tightly defined, which may lead to very creative artifacts! One can gather information about conceptual but not creative designs, since the creativity may be in the process or the artifact. Conceptual design is in a continuum with detailed design, production, distribution, and recovery. Hence, it is traceable in the design process. *Selection design*, on the other hand, deals with selection of alternatives, for example, from catalogs.

The problem and process types of taxonomies were used to classify conceptual and selection types of design by Ullman (1992a).

Process Representation

One model may not capture all the unique models of each individual designer. Rather than developing a single model of the design process, it is more useful to have a framework in which multiple models of different researchers can be incorporated. Conceptually, a design can be thought of as moving from function (specifications) to structure (form or artifact) along the axis of a cylinder. Each designer's experience is the spiral bound around the axis with creativity as the radius of the cylinder.

Figure 4.1 shows a possible framework to accommodate design process models. There are a given set of functions that the designed artifact must perform. The designer's task is to attach the appropriate structures that will perform the specified functions. Designers with less experience may require functional decomposition until they know of an associated structure for each decomposed function that may accomplish that function. The mapping to the structure is then performed at the subfunctional level in the form of

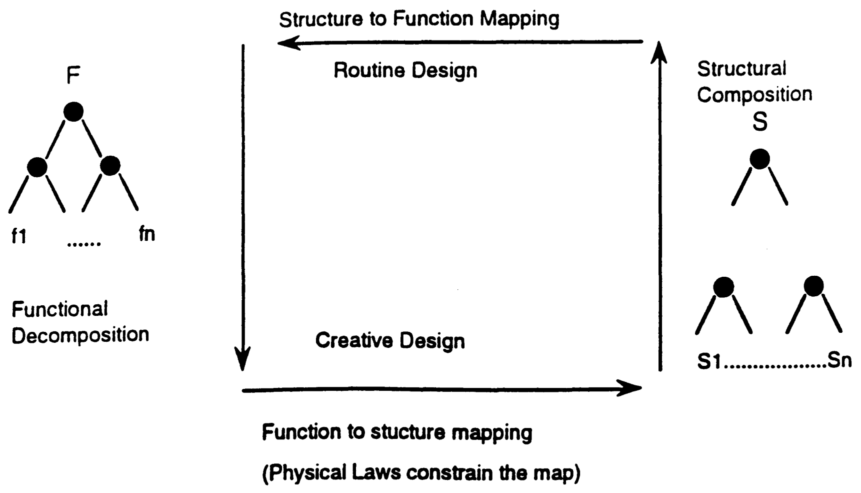


FIGURE 4.1. A possible design framework to accommodate design models.

substructures, which must be recombined to achieve the overall structure. This is the synthetic aspect of design. Analysis of this structure ensures that the overall function would meet the original specifications. Designers with extensive experience in a particular design may already know the associated structure that could accomplish the intended function. Therefore, they may proceed opportunistically to select the appropriate structure(s) to attain the function. This is more of a routine design. The depth of functional decomposition and structural recomposition on which the designer must rely would be a measure of the creativity in the design process. Hence, for each designer there is a unique process model for each design.

It is important to note that the function-to-structure mapping contains both the form and the design knowledge that can be used in future. In specific representations and models one would need to retain the design history and the design rationale (Ullman, 1992b). The design history is defined as a sequence of choices, the alternatives, and the descriptions of these alternatives that are available to designers because of previous encounters. The design rationale is the reason behind these choices. Furthermore, the design is complete only when someone declares it to be so or when all the stated specifications are achieved. The latter does not assume satisfaction but only guarantees completeness.

The output of the design process is the knowledge that a plan for creating the artifact exists, either in the form of prototypes or working models or drawings that guarantee such production. This output also contains the resultant change in the process as a result of traversing through the steps of the design process. The representation of this learning is important and as yet not satisfactorily studied or represented. Further analogical reasoning plays an important role in the process. This too is rarely studied and modeled in design.

Specific Models

Finger and Dixon (1989) provide an excellent review of the research in mechanical engineering design. In Part I of their paper, they discuss the descriptive, prescriptive, and computer-based models of design. The reader is referred to this excellent reference for both an extensive review and brief descriptions of each type of models. In this chapter we present specific models for each type of design in detail.

Descriptive Models

These models are based on the observation of designers designing. Protocol or depositional methods are used for collecting information on the process (Chapter 3). The analyzed results provide information on what is important to designers, and permit construction of models that can describe the pro-

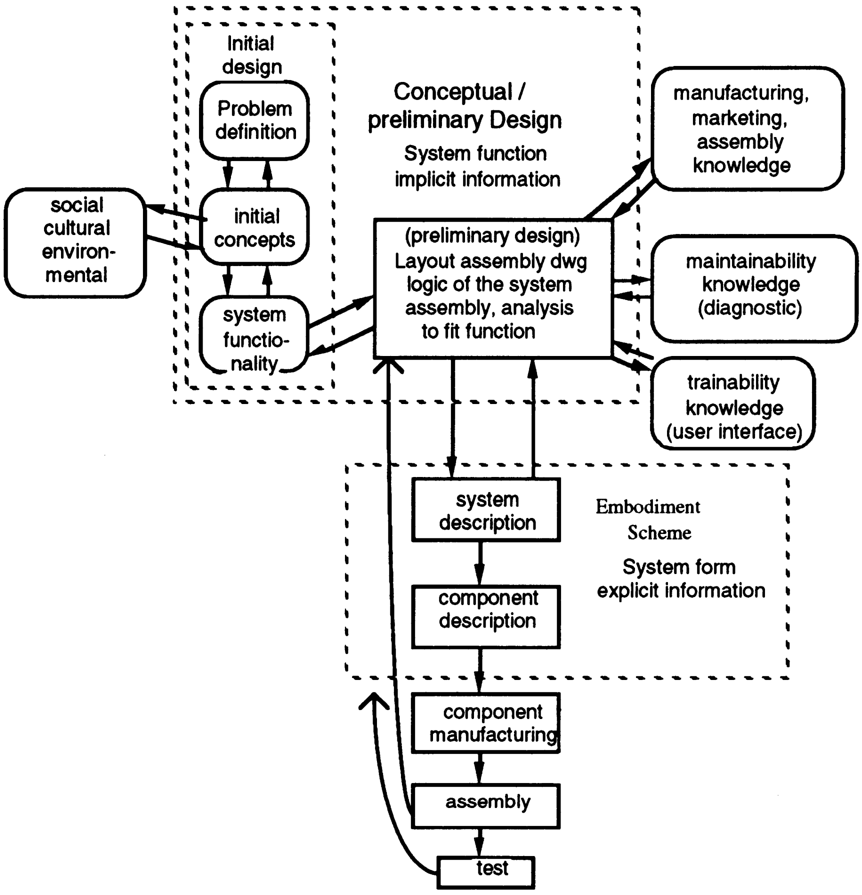

Initial design
Problem definition
initial concepts
system functio-nality
social cultural environ-mental
Conceptual / preliminary Design
System function implicit information
(preliminary design) Layout assembly dwg logic of the system assembly, analysis to fit function
manufacturing, marketing, assembly knowledge
maintainability knowledge (diagnostic)
trainability knowledge (user interface)
system description
component description
Embodiment Scheme
System form explicit information
component manufacturing
assembly
test

FIGURE 4.2. A functional model of the design process.

cess. Most of these studies have been conducted on the conceptual phases of design (Ullman et al., 1988; Waldron et al., 1989)

One such model is the *knowledge flow model* described by Waldron and Vohnout (1989). Figure 4.2 shows this design process model based on the flow of information among different units engaged in the design process and accounts for the knowledge used and the realization of the functionality. The conceptual, preliminary, and detailed design stages interact so as to reflect design–evaluate–redesign. In addition, this model takes into account the knowledge designers must use to ensure the success of designs, such as, manufacturing, marketing, assembly, maintainability, disposability, and usability. It accounts for the knowledge that the designer is using to create the design, which can be readily communicated to the manufacturing environment. The design knowledge organization strongly influences the way in which the process planners may use the information contained in the knowledge base.

The approach is to examine the designer's knowledge and its organization in the mode of communication used in the design process. In Figure 4.2, the conceptual design block represents the beginning of the design based on the initial problem definition, which also forms the transactional part of the conceptual design as well as the driving force for it. The initial design phase takes the problem definition and converts it into functional specifications. Based on the designer's social, cultural, and environmental knowledge, the problem statement and the functional specifications may be negotiated until a set of functional specifications are found that satisfy the designer and allow him or her to proceed to formulate embodiments of design schemes which can be used to create detailed and working drawings (French, 1992).

In the preliminary design process the artifact is conceived so that it can carry out the prescribed functions. The designer ensures that the logic of the design is satisfactory in the layout drawings. These drawings, once detailed, will be used by the manufacturer of the artifact. In this phase, in order to produce successful and efficient design, the manufacturing knowledge that will be used to produce the artifacts is helpful. The layout and detailed drawings, after performing the associated analysis, contain the information about the shape, geometry, dimensions, and tolerances so that the manufactured artifact will indeed perform as intended. In creating the assembly drawings, the designer must also have the assembly knowledge to ensure that the drawings contain enough information. This will ensure that the manufactured artifact, when assembled, will achieve its intended function. Paying attention to the servicing (maintenance) during design will also ensure proper and easy service of the product during its use. Knowledge of its intended use and the customer needs will ensure that the product will be used to accomplish the intended function. Finally, incorporating the knowledge of product disposal during the design phase will ensure a minimal impact on the environment after the useful life of the product is over.

This additional knowledge may be part of the existing knowledge the designer possesses, or it may be that it is acquired through interactions with the appropriate persons in the design team. Hence, this model incorporates concurrent engineering practices. The actual manufacturing, assembly, use, maintenance and disposal of the product is its explicit manifestation. The evaluative feedback knowledge, to the designer, from the manufacturers, users, field technicians, and people dealing with waste disposal is important to contribute to the improvements of future designs.

The separation of the model into blocks is necessary for representational purposes so that the construction of computational models is possible. In observing designers working one may not see a clear separation of these blocks. Rather this information is inferred from the observed data. Human designers, through their integrated, associative, experiential knowledge, may flow fluently from the problem statement to the formulation of functional specifications and then to using their accumulated knowledge to arrive at the drawings and specifications that are used for the manufacture of the product.

The designer at the conceptual level cycles back and forth between the system, subsystem hierarchy until all the constraints are satisfied and a solution is specified. The choices are made based on the context of the design, which depends on the negotiable and the nonnegotiable constraints. The model allows the design process to be represented by three levels based on context. The highest level or the semantic level allows the goodness of the design to be assessed. The next or the syntactic level allows the logic of the design to be evaluated. The lowest level allows the selection of the shapes and dimensions so that the functional constraints specified in the other two levels can be achieved. This paradigmatic representation allows a computer system to be developed which can be orchestrated by the designer.

Williams (Waldron, 1990) proposed a model based on the reasoning cycle, which is employed by the designers using three frameworks of knowledge. His model was derived from the results of protocol studies done by others, and is based on the premise that designers have a desire to do something and a cognitive ability to do it. In designing, the designer faces two types of situations that depend on complexity and uncertainty. Complexity is when the designer has too much knowledge and uncertainty when the designer has too little knowledge. These are interrelated. The standard observations were:

1. That chunking is ubiquitous in the design process and the designer organizes the elements of the problem into a hierarchy.
2. Knowledge is used to guide the designer; that is, it is not a blind search.
3. The designer needs feedback from the results of a prototype or a design simulation, in order to improve the design further.
4. Conjectures are initially made, by designers, based on abductive reasoning, and then deductive reasoning may be used to from hypotheses. Once they have a clear hypothesis designers act to arrive at a problem solution and then proceed to a more generalized model.

The three frameworks of knowledge are the *generic knowledge*, the *specific knowledge* and *information* or *data about the knowledge*. There is an interaction between these frameworks of knowledge, which directly affects the human reasoning process in the design activity. The consequences of these frameworks allow the computer systems to be isolated from designers and allow them to use their personal effective knowledge, and not be constrained by what is provided by the system.

Another model, task-episode-accumulate (TEA) model, was proposed by Ullman et al. (1990). This model uses the information processing during the design process as its basis. Here they argue that the design information may be stored in the mind of the designer (internal environment), who uses short-term memory and long-term memory to retrieve the information and make it available to the design state. Alternatively the information may be stored in an external environment such as books, papers, drawings, or in the minds of team members. The evolution of the design is incorporated as a design state in the TEA model. A design state contains the accepted design proposals and the constraints that these designs must satisfy. The design states are modified by design operators such as select, calculate, simulate, etc.

Prescriptive Models

Prescriptive models, as the name implies, allow one to prescribe the sequences of activities that constitute the design process. This is distinguished from the axiomatic prescription for the attributes of the designed artifacts as proposed by Suh (1990). In this chapter we present the design process model proposed by Pahl and Beitz (1984), which formalizes the design process into four systematic phases: clarification of the task, conceptual design, embodiment of the design, and detailed design.

In the clarification stage the specifications are elaborated and working specifications result. During the conceptual design phase, the problems are identified, functional structures are established, concept variables are firmed, which are evaluated against the technical constraints. The embodiment phase of the design results in design layouts through design refinements and optimization. In the detailed design phase the design is finalized and detailed drawings are produced complete with design documentation.

The assumption underlying this design process model is that the design process can be decomposed into levels of abstraction and the design proceeds in a sequence from the definition of subfunctions to selection of physical principles, to embodiment of the design, to detail design, to production planning. At this point, a new problem is defined and the cycle repeats until a solution is reached. It is assumed that both the designers and the design process go through the same cycle of design activities.

These activities consist of solving a series of problems into which the design has been decomposed. A solution is found either from intuition or from a known method and related information. This solution is analyzed by simulation and calculation. The constraints are evaluated and a decision is made whether to repeat this process or go on to the next problem identified. Hence, each solution leads to another problem. Until this solution is arrived at, one does not proceed to the next problem.

In this theory, the design solution is assumed to be a puzzle defined at two levels, namely, the abstract and the solution level. The abstract level is assumed to be mapped functionally onto a solution level. It is assumed that the physical principles that will bear on a particular function are known *a priori*, and the design is functionally well defined and decomposable. The problem with the underlying theory is that it requires the process to be independent of the designer. The theory does not allow for the use of analogy, spatial reasoning, etc., which seem to be very important when humans design. Furthermore, the theory assumes that a general problem-solving model exists, which is not necessarily so.

Computer-based Models

In computer-based models, the information about the design process is entirely in the computer program. This is different from the cognitive model where the human designer plays an important role and the model describes what the human designer does. The computer-based models describe the methods by which the computers themselves can accomplish the designing task. These models do not necessarily have to be derived from human behavior although they may. Likewise the successful computer-based design process may or may not be used by humans. These may be entirely embedded in another design problem. These reciprocal relations, between human design activity and the computer programs are neither mandated nor essential (Finger & Dixon, 1989).

Computer-based models have been developed on an ad hoc basis, although such models are being developed in research laboratories and have been developed to solve specific design problems. There is as of yet no single theoretical approach that can handle all aspects of design. The computer-based models largely describe computer programs that carry out, or assist in specific design tasks. Since these models are still being developed and no established procedures exist for the overall design process in industry, the design process models really do not exist except in the form described below.

In parametric design the structure of the object to be designed is known and in the design process appropriate values need to be assigned to the parametric design variables. Constraints and criterion functions when expressed numerically allow optimization techniques to be used (Finger & Dixon, 1989). Dominic (Dixon et al., 1989) is one of the first computer systems for knowledge-based parametric design of components; it uses iterative redesign.

A computer model for "routine" design was created by Brown and Chandrasekaran (1989). They devised a Design Specialists and Plans Language (DSPL) in which the hierarchy of alternative design plans are contained. It is a computer model for routine design consisting of cooperating agents such as specialists, plans, tasks, steps, constraints, redesigners, failure handlers, etc. (Spillane & Brown, 1992). These agents are hierarchically organized and cooperate to configure the design. DSPL has been used for design of air-cylinders and gear pairs.

There are computer-based models for configuration design. Finger and Dixon (1989) describe it as the transformation of physical concept into a configuration with a defined set of attributes without specific assigned values. Mittal et al. (1986) developed a configuration design software called PRIDE (Pinch Roll Idler Design Expert), for designing paper handling systems inside copiers. This computer program assists in the design of a paper transport mechanism for copiers by using a knowledge base to generate, evaluate, and redesign the configuration of rollers. Figure 4.3 shows such a computer-based configuration design model. The configuration is defined in terms of

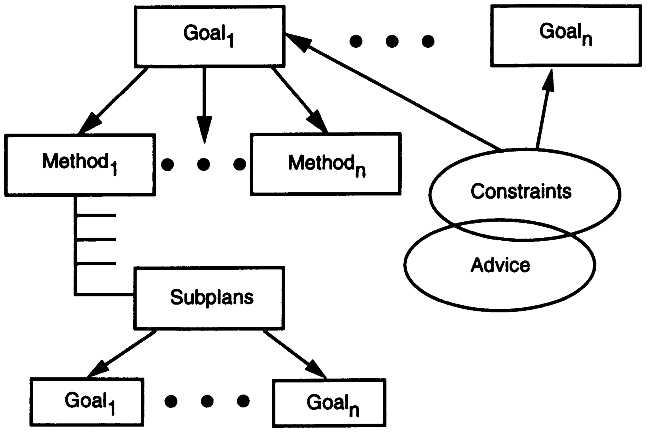


FIGURE 4.3. Computer-based model for configuration design.

goals and each goal has its own method and subplans (which could have goals of their own) with associated constraints. In addition the "advice" feature guides the possible redesign paths (Waldron, 1990).

Computer-based programs for conceptual design are capable of synthesizing structural components from the functional or behavioral requirements (Ulrich & Seering, 1989) or by reasoning from first principles (Cagan & Agogino, 1987; Williams, 1992). However, these models are still in research stages; how effective they will be in influencing the design process in future is yet to be determined.

Modeling and the ICAD Systems

How do models of the design process affect what is produced? The interaction between the model and the product is important. What is the motivation behind the study of the process and its relation to ICAD systems? What are the issues that are important in design? Should one study the issues associated with the design process such as the methodological issues or the issues associated with building computer systems that support design? These questions are all important and bear investigation.

On one end, one may study the designers and find out what the designers do, while, at the other end, one may make guesses as to what the designer does, and create the computer system based on the assumptions and then find out if this assists the designer. Perhaps the answer is somewhere in the middle. One may want to get some information from protocol or depositional studies, to obtain sufficient information on which to base valid hypotheses, models, and conjectures from which the systems could be developed and expanded upon through educated guesses.

The model and framework of the design process presented earlier, while useful in tracking the design process, is too abstract to be very useful in creating practical ICAD systems. In order to guide what the implementers of the ICAD systems should do, we need to consider the separate environments for design, and develop a specific model or models that help in the development of ICAD systems. The important aspect in this representation is not a search for a proper representation of the process in the machine. Rather, the search is for the information structure associated with the design process that contains the strategies and the manner in which the process changes and the design evolves. In the ICAD systems it is more important to consider the control issue rather than the representation issue, so that a distinction can be made between the artifact and the process in this system.

There is certainly a strong connection between the process and the object. The model of the design process may be related closely to the artifact rather than having a generic process for all artifacts. But one must study this interaction and find out what actually takes place. How would the designer interact with the system to facilitate the interaction? Should the system be made more general rather than specific? How would the learning consequences of the designers be incorporated in the ICAD system? These are important questions that must be tackled before reliable computer-based models which are effective in directing the design process can be envisaged.

Comparison with Other Disciplines

When one considers the mechanical design process in relation to other designs, one sees some analogies. For example, in architecture, one could view the architect more as a product designer, a structural engineer as a mechanical engineer, and a contractor more like a manufacturer, a person who must physically realize the design created.

The input-process-output model poses the problem of how to characterize inputs and outputs. In some domains, functionality may be the input and the defined attributes of the object are the output, whereas in other domains, for example architecture, this may be reversed because the vocabulary for the function and form are often the same and the designers think of these interchangeably. For example, the word "coupling" may mean both the function of coupling two shafts or the form this coupling takes. Furthermore, the design specifications of goals may be loose and may change during the course of the design process. In some fields, artifacts may be defined in the specifications, while in others these may evolve. Furthermore, one object or artifact may perform many functions, and one function could be performed by many artifacts.

The function-to-form mapping could therefore be one to many or many to one depending on the situation. By making many arbitrary or ad hoc assumptions, the designers overcome this problem. The assumptions may be based on acceptability of the product in the culture in which it will be placed; hence, acceptability becomes a functional requirement. But the artifact has to be realizable, which becomes a physical problem; hence, the input to the design process may take the form of acceptability, realizability, or feasibility, which are really qualifications on the result. To achieve this, one must have the knowledge of both the process and the domain. In that case, one needs a language to describe the models with "hard" and not "soft" vocabulary. This is difficult to achieve in design because the designer's language is inherently "soft" or ambiguous. Acceptability is therefore harder to define than realizability and feasibility (Waldron, 1989).

The following is a list of some domain-independent characteristics of the design process along which the relative levels of difficulty for each domain could be discussed.

Functional Specification

1. Difficulty of description	E > M > S > A
2. Static and dynamic relations for temporal analysis and simulation	A > M > S > E
3. Ambiguity of environment	E > M > A > S
4. Function to form confusion	S > E > M > A

Artifact

1. Geometrical form is the main attribute	S > E > M > A
2. Tolerance and manufacturing as part of design	S > E > A > M
3. Assembly of many parts working together	E > A > S > M
4. Difficulty in standardizing	E > A > M > S
5. Not symbolic	S > E > M > A
6. Multifunctional component	S > E > M > A

Designing

Non-isomorphism between functionality and artifact may be intuitive, concurrent	S > E > M > A

where S = software; E = electrical, including control; M = mechanical; and A = architecture.

Conclusions

In this chapter characterizations of engineering design as described by different researchers was described. Further, a taxonomy of engineering design was discussed. The characterization of the design process was described in detail. Different types of models based on various theories of design were presented and compared. Furthermore, the use of models in creation of

CAD tools was discussed and the difficulty in design characterization in different design domains was compared. From the discussions presented it is apparent that the design process models are still evolving and there is, as yet, no single, well-developed theory for design. While considerable effort is being expended toward the establishment of such a theory, the progress is still slow and somewhat tenuous.

References

Brown, D. C., & Chandrasekaran B. (1985). Expert systems for a class of mechanical design activity. In J. S. Gero (editor) *Knowledge Engineering in Computer Aided Design*. Amsterdam: North Holland.

Brown, D. C., & Chandrasekaran, B. (1989). *Design Problem Solving*: *Knowledge Structures and Control Strategies*. Los Altos, CA: Morgan Kaufman.

Bucciarelli, L. L., & Schon, D. S. (1987). Generic design process in architecture and engineering: A dialog concerning at least two design worlds. In M. Waldron (Ed.). *Proceedings from the NSF Workshop on the Design Process*, Oakland, CA. 43–67.

Cagan, J., & Agogino, A. M. (1987). Innovative design of mechanical structures from first principles. *Artificial Intelligence for Engineering Design*, *Analysis & Manufacturing*, *1*(3), 169–190.

Dixon, J. R. (1987). On research methodology towards a scientific theory of engineering design. *Artificial Intelligence for Engineering Design*, *Analysis & Manufacturing*, *1*(3), 145–158.

Dixon, J. R., Guenette, M. J., Irani, R. K., Nielsen, E. H., Orelup, M. F., & Welch, R. V. (1989). Computer-based models of design processes: The evaluation of designs for redesign. In *Preprints of the National Science Foundation* (*NSF*) *Engineering Design Research Conference*, University of Amherst. MA. 491–506.

Dym, C. L. (1994). *Engineering Design*: *A Synthesis of Views*. Cambridge University Press, NY.

Dym, C. L., & Levitt, R. E. (1991). *Knowledge Based Systems in Engineering*. McGraw Hill, NY.

Finger, S., & Dixon, J. R. (1989). A review of research in mechanical engineering design. Part I: descriptive, prescriptive and computer based models of design process. *Research in Engineering Design 1*(1), 51–67.

Fitzhorn, P. (1989). In *Preprints of the National Science Foundation* (*NSF*) *Engineering Design Research Conference* University of Amherst. MA. 221–231.

French, M. E. (1992). *Form*, *Structure and Mechanisms*. London, UK: MacMillan.

Mastow, J. (1985). Towards better models of the design process. *AI Magazine Spring*, pp. 44–45.

Mittal, S. M., Dym, C. L., & Morjaria, M. (1986). PRIDE: An expert system for the design of paper handling system *IEEE Computer*, *19*(7), 102–111.

Pahl, G. & Beitz, W. (1984). *Engineering Design*. London, UK: The Design Council.

Simon H. (1969). *The Sciences of the Artificial*. Cambridge, MA: The MIT press.

Spillane, M. & Brown, D. (1992). Evaluating design knowledge compilation mechanisms. In D. Brown et al. (Eds.). *Intelligent Computer Aided Design*. Amsterdam: North Holland. 351–374.

Suh, N. (1990). *The Principles of Design*. Oxford, UK: Oxford University Press.

Ullman, D. G., Dietterich, T. G., & Stauffer, L. A. (1988). A model of the mechanical design process based on empirical data: A summary. In J. S. Gero, (Ed.). *AI in Engineering Design*. New York: Elsevier, 193–215.

Ullman, D. G., Wood, S., & Craig, D. (1990). The importance of drawing in the mechanical design process. *Computers and Graphics*, *14*(2), 263–274.

Ullman, D. G. (1992a). A taxonomy for mechanical design. *Research in Engineering Design*, *4*(3), 172–187.

Ullman, D. G. (1992b). *The Mechanical Design Process*. New York: McGraw Hill.

Ulrich, K., & Seering, W. (1989). Conceptual design: synthesis of schematic descriptions in mechanical design. *Research in Engineering Design*, *1*(1), 3–18.

Wallace, K. N., & Hales C. (1987). The application and evaluation of formal design engineering methods. *Proceedings of the International Conference on Engineering Design*, *ICED*, Boston, MA: 94–101.

Waldron, M. B. (1988). Tristratal level of design: A foundation for intelligent CAD in mechanical engineering. *Proceedings IFIPWG 5.2 Workshop on Intelligent CAD Preprints*, 468–475.

Waldron, M. B. (1989). Modeling of the design process. In H. Yoshikawa & D. Gossard (Eds.). *Intelligent CAD I*. Amsterdam: North Holland, 13–29.

Waldron, M. B. (1990). Understanding design. In H. Yoshikawa & T. Holden (Eds.). *Intelligent CAD II*. New York: North Holland, 73–88.

Waldron, M. (1991). Design processes and intelligent CAD. In F. Arbab & H. Yoshikawa (Eds.). *Intelligent CAD III*. Amsterdam: North Holland, 56–75.

Waldron, M. B., & Waldron, K. J. (1988). Position paper on conceptual CAD for mechanical designers. *Proceedings of Computers in Engineering*, August, 2. 203–211.

Waldron, M. B., & Waldron, K. J. (1988). Time study of the design of complex mechanical systems. *Design Studies* April *9*(2), 95–106.

Waldron, M. B., & Waldron, K. J. (1989). Empirical study on generation of constraints which direct design decisions in conceptual mechanical design. 1989 NSF Engineering Design Conference U Mass. June 15–30.

Waldron, M. B., Waldron, K. J., & D. H. Owen (1989). Use of systemic theory to represent the conceptual mechanical design process. In Newsome et al. (Eds.). *Design Theory 88*. New York: Springer Verlag, 36–48.

Waldron, M. B. & Vohnout, V. J. (1989). Formalizing knowledge in design for CAD/CAM integration. In A. Samuel (Ed.). *Managing Design and Manufacturing*. Amsterdam: North Holland. 42–64.

Williams, B. C. (1992). Interaction-based design: constructing novel devices from first principles. In D. Brown et al. (Eds.). *Intelligent Computer Aided Design*. Amsterdam: North Holland. 255–275.

5
An Observational Methodology for Studying Group Design Activity

JOHN C. TANG AND LARRY J. LEIFER

Abstract. A methodology for observing and analyzing group design activity is presented. This methodology is based on ethnographic and interaction analysis methods from the social sciences. Using it to study collaborative design activity leads to a descriptive analysis that identifies what resources the designers use and what obstacles they must overcome to accomplish their work. Based on this analysis, a better understanding of the needs of designers can be used to guide the design of tools to support group design activity. For example, this analysis led to an understanding of the role of hand gestures in collaborative design activity. Gestures are used to help demonstrate actions and establish shared reference. Hand gestures are often conducted in relation to sketches and other objects in the shared workspace. Descriptions of how to record group activity on videotape, represent and analyze the data (using a hypertext system), and abstract general observations from the data are presented.

5.1. Introduction

The design process is a complex and creative activity that has long been the subject of study. Several different methodologies have been applied to study design activity, as reported in the overview papers of Bessant (1979), Wallace (1987), and Finger and Dixon (1989). To name a few, Thomas and Carroll (1979) conducted psychological experiments probing design activity, Ullman et al. (1987) applied protocol analysis on individual designers "thinking aloud," and Wallace and Hales (1987) used participant observation to study an engineering design project for almost three years.

The research presented in this paper draws upon an exisiting methodology, known as interaction analysis, to study group design activity. Videotape

This paper first appeared in *Research in Engineering Design*, Vol. 2. 1991, pp. 209–219. Reprinted with permission.

records of actual design activity are analyzed to identify how the designers accomplish their work and what problems they encounter along the way. This qualitative description of design activity leads to a deeper understanding of the design process and raises implications for the development of technology to support it. This methodology was used in recent research to study small group, conceptual design activity (Tang, 1989), leading to design implications for tools to support that activity. In applying this methodology to *study* a particular design activity, we also discovered ways in which it could be used *as part of* any design process to understand the needs of the end user.

This chapter presents a methodology for studying group design activity based on interaction analysis methods, which are introduced in Section 5.2. Detailed descriptions of how to observe and analyze group design activity are presented in Sections 5.3 and 5.4. As an example of the kind of findings that this methodology yields, Section 5.5 discusses observations on the role of hand gestures in collaborative design activity. The advantages and constraints of this methodology are discussed in Section 5.6, and applying it as part of the design process is discussed in Section 5.7.

5.2. In Introduction to Interaction Analysis

The observational methodology presented in this paper is based on interaction analysis, a qualitative analysis method used in the social sciences. In the fields of anthropology and sociology, qualitative methods are used to investigate human activity. Since group design activity is a complex social activity, it is appropriate to apply these methods to study it. Other design researchers [Bessant & McMahon, 1979; Darke, 1979; Wallace, 1987] have also advocated applying social science methods to study design activity.

In the field of anthropology, ethnographic studies observe the activities of a culture by participating in it through an extended period of time. The daily life of the culture is studied in its natural setting with minimal disruption to that activity. The resulting ethnography is a description of the common practices of that culture, as experienced by the observer. Recently, ethnographic methods have been used to study not only foreign cultures, but professional subcultures in developed countries (Latour & Woolgar, 1979; Lynch, 1985), and design activity in particular (Bucciarrelli, 1988).

Interaction analysis is a recent development in anthropology and qualitative sociology that integrates an ethnographic perspective with fine-grained analysis of human interaction. This methodology involves analyzing records of human activity in order to understand how that activity is accomplished through the interactions among the participants and the artifacts in their environment. Ideally, the participants should be observed in their natural working environment addressing a real task. Logistics sometimes dictate that the situation be structured to the extent that a realistic task is given to the participants in an environment where they can be easily observed. A crucial element of this approach is that the researcher not intervene in the group's activity once they have begun working on the task. The participants are free to organize their work as they wish, and it is the observer's responsibility to record and analyze the activity that subsequently unfolds. The goal is to capture samples of human activity in contexts in which they would naturally occur.

The activity is typically recorded on videotape, which is analyzed to identify patterns in how the participants accomplish or are hindered from accomplishing their work. By collecting and comparing among examples from the data, specific resources that the participants use to help them accomplish their work or obstacles that hinder their work can be identified.

Conversation analysis is a prominent form of this kind of analysis that studies how people interact through conversation (Levinson, 1983; Sacks et al., 1974). Interaction analysis extends beyond focusing only on the conversation of the participants to include other aspects of how people interact with each other and their environment. Examples of video-based interaction analysis include the study of the accompanying nonverbal behavior in conversation (Goodwin, 1981; Heath, 1986) and the interaction between humans and technology (Suchman, 1987). Our research extends the use of interaction analysis to study group design activity.

This approach contrasts with experimental methods where tightly controlled situations are constructed to test a preformulated hypothesis. Rather, interaction analysis explores naturally occurring activity to identify and understand what parameters and relationships are important to the interaction. This approach also contrasts with participant observation, which relies solely on the accuracy, completeness, and objectivity of notes collected by the participant observers. Rather, the activity is recorded on videotape, which can be reviewed again and again from a variety of perspectives. These underlying tenets of interaction analysis are described in Suchman's (1987) study of human–machine interaction:

> This study proceeded, therefore, in a setting where video technology could be used in a sort of uncontrolled experimentation. On the one hand, the situation was constructed so as to make certain issues observable.... On the other hand, once given those tasks, the subjects were left entirely on their own. In the analysis, by the same token, the goal was to construct a characterization of the "interaction" that ensued, rather than to apply a predetermined coding scheme. Both predetermined coding schemes and controlled experiments presuppose a characterization of the phenomonenon studied, varying only certain parameters to test the characterization. Application of that methodology to the problem of human–machine interaction would be at the least premature. The point of departure for the study was the assumption that we *lack* a description of the structure of situated action. And because the hunch is that the structure lies in a relation between action and its circumstances that we have yet to uncover, we do not want to *presuppose* what are the relevant conditions, or their relationship to the structure of the action. We need to begin, therefore, at the beginning, with observations that capture as much of the phenomenon, and presuppose as little, as possible (Suchman, 1987, p. 114, original emphasis).

Our research is premised on the need to observe and understand what design teams actually do in order to guide the design, development, and introduction of tools to support their activity. Our research applies the interaction analysis methodology to study the activity of design teams.

5.3. Observing Group Design Activity

In our studies, eight different sessions of small groups (3–4 people each) working on conceptual design tasks were observed. The groups consisted of peer participants who were not in the context of any formal authority hierarchy (i.e., no supervisors with people who report to them). The observed sessions were the first time that the participants worked together as a group on the task, thus capturing the earlier, more conceptual stages of the design process. All of the tasks that the groups worked on were human–machine interface design problems (see sample problem statement in Appendix). The groups typically worked on the task for about $1\frac{1}{2}$ hours, deciding on their own when to end their session.

Videotape was used to record the design activity for later analysis. The final configuration for the observational equipment used in our research is depicted in Figure 5.1. Two video cameras were mounted on tripods: one aimed at the shared workspace of the group, while the other captured a wide angle view of the group as a whole. The cameras were "passive" in that they

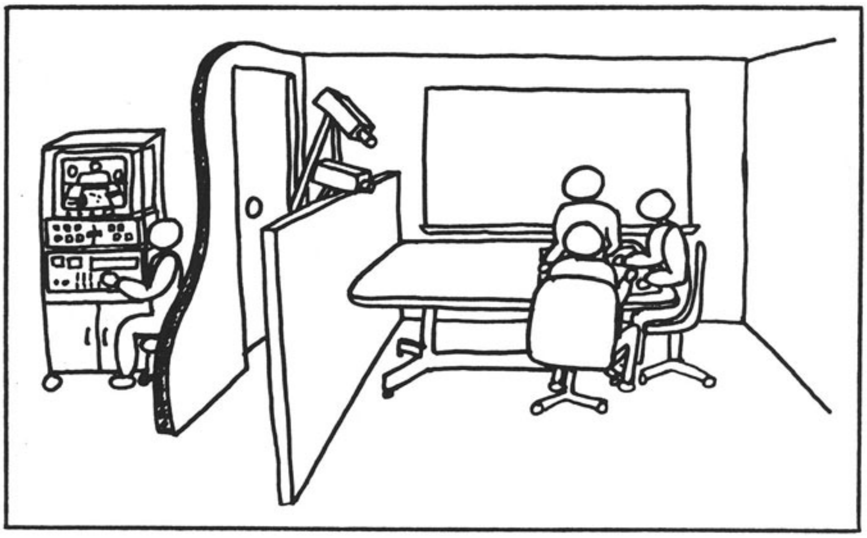


FIGURE 5.1. Observational configuration. The participants were separated from the observational equipment and the experimenter. One camera is aimed at the shared workspace of the group, another captures a wide angle view of the group as a whole.

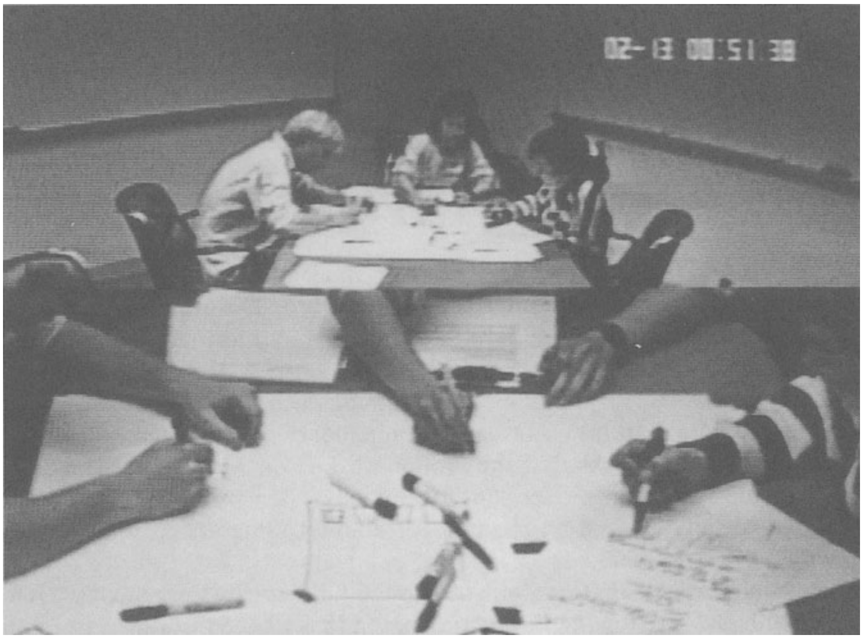


FIGURE 5.2. The recorded video image. The split screen image combines the wide angle view of the group (top) with a close-up view of the group's shared workspace activity (bottom).

were not moved or re-aimed during the session. This arrangement is considered less distracting than having an active camera person in the room aiming and focusing the cameras. The cameras were partially obscured from the participants by a partition, and the experimenter and recording equipment were located in a neighboring room.

The signals from the two video cameras were combined into one split screen video image, shown in Figure 5.2. A time stamp that displays the date and elapsed time in hours, minutes, and seconds was included in the video image. This time stamp was used to index the contents of the videotape. The split screen image with time stamp and the accompanying audio were recorded on videotape. An additional audio tape recording was made as a back-up and for use in transcribing equipment to help make a transcript of the verbal dialog.

5.4. Analyzing the Data

Videotape records of design activity contain a wealth of data for analysis, which can initially be overwhelming. Reviewing the videotape data itself quickly suggests more specific foci for analysis. Analyzing the video data involves:

- becoming familiar with the data
- developing a workable representation of the data for analysis
- abstracting patterns and general observations from the data

Although it is clearer to introduce this process of analysis as if these activities occurred in a three step sequence, the actual analysis was much more complex. The three activities occurred concurrently and were informed by each other. It was often the case that representing the data or identifying patterns in the data led to a new perspective on it, prompting a re-familiarization with the data or a modified representation for the data for further analysis. For clarity of presentation, this section presents an idealized, three step framework for the analysis. However, the examples drawn from our study of group design activity will indicate that the analysis that actually occurred was a much more interrelated process.

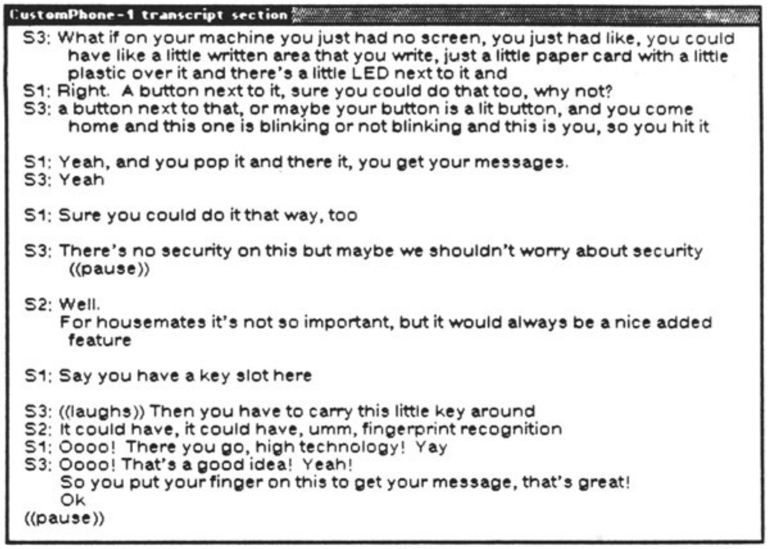
5.4.1. Becoming Familiar with the Data

After videotaping the design sessions, the initial task in analyzing the data is to review the tapes to become acquainted with the sequence of events in the session and to note incidents for closer examination. A good exercise for becoming familiar with the data is to make a transcript of the verbal dialog of the session. Deciphering who said what and in what order is a prerequisite for deeper understanding of the activity.

Figure 5.3 shows a section of transcript from a design session. The speaker associated with the text is designated by the 'S' labels. Some indication for the pacing of the speech is given through the punctuation and line formatting. Turns of talk from different speakers with no line space between them indicate overlapping talk.

In our study of group design activity, making a transcript of the verbal dialog not only helped us become familiar with the data, but also revealed that the transcript by itself did not adequately represent the recorded design activity. Understanding the transcript often required attending to the accompanying drawing and gesturing activity that was observable on the videotape. This initial familiarization exercise led to the development of a representation that included these nonverbal activities as will be described later.

Another technique that is helpful for developing an overall perspective on the data is to bring several different viewpoints to bear on the video data. At the time of this research, a working group of designers, anthropologists, and computer scientists (called the Interaction Analysis Lab) met weekly at the Xerox Palo Alto Research Center to review videotapes of human activity. These meetings brought together insights on the data from the different perspectives of these disciplines. Since the researchers came from different academic disciplines, they each brought different sensitivities to bear on analyzing the video data. Furthermore, they were each forced to demonstrate their claims about the activity by observable evidence from the video data, rather than relying on any single discipline's characterization of human


CustomPhone-1 transcript section

S3: What if on your machine you just had no screen, you just had like, you could
have like a little written area that you write, just a little paper card with a little
plastic over it and there's a little LED next to it and
S1: Right. A button next to it, sure you could do that too, why not?
S3: a button next to that, or maybe your button is a lit button, and you come
home and this one is blinking or not blinking and this is you, so you hit it

S1: Yeah, and you pop it and there it, you get your messages.
S3: Yeah

S1: Sure you could do it that way, too

S3: There's no security on this but maybe we shouldn't worry about security
((pause))

S2: Well.
For housemates it's not so important, but it would always be a nice added
feature

S1: Say you have a key slot here

S3: ((laughs)) Then you have to carry this little key around
S2: It could have, it could have, umm, fingerprint recognition
S1: Oooo! There you go, high technology! Yay
S3: Oooo! That's a good idea! Yeah!
So you put your finger on this to get your message, that's great!
Ok
((pause))

FIGURE 5.3. Sample transcript section. This sample section of a transcript illustrates how the verbal dialog and its pacing are represented. The speaker is designated by "S" labels.

activity. This emphasis on understanding human activity through the directly observable interactions among people and their environment is a distinctive characteristic of interaction analysis. This approach contrasts with cognitive orientations that account for human activity by mental activity that is not directly observable. Multidisciplinary group analysis is a practical technique for assuring that the resulting observations are based on observable evidence from the data.

Selected segments of our video data on group design activity were reviewed in the Interaction Analysis Lab. One issue that emerged from these analysis sessions is the variety of activities that could be observed in the recorded design activity and their interrelationships: talking, writing text, drawing graphics, and gesturing. These sessions helped identify some of the patterns of activity (e.g., instances of using hand gestures, classifying the various uses of gestures, quick alternation among writing, drawing, and gesturing) that we focused on in our research, as will be discussed later.

The videotapes can also be reviewed with the participants themselves to elicit their perspectives and help focus the analysis of the video data. In our research, the participants were invited individually to review the videotape. This technique was modeled on the work of Frankel and Beckman (1982) in their analysis of doctor-patient interactions. The participants were encouraged to comment freely on what they saw; they could stop the tape at any time to interject their thoughts. These sessions also provided us an opportunity to ask the participants specific questions about issues that arose in our prior examinations of the tape. We believe that reviewing the actual data with the participants elicits more detailed recollections than if they were asked in an interview to recall their thoughts from memory. These review sessions were audio taped to record their comments.

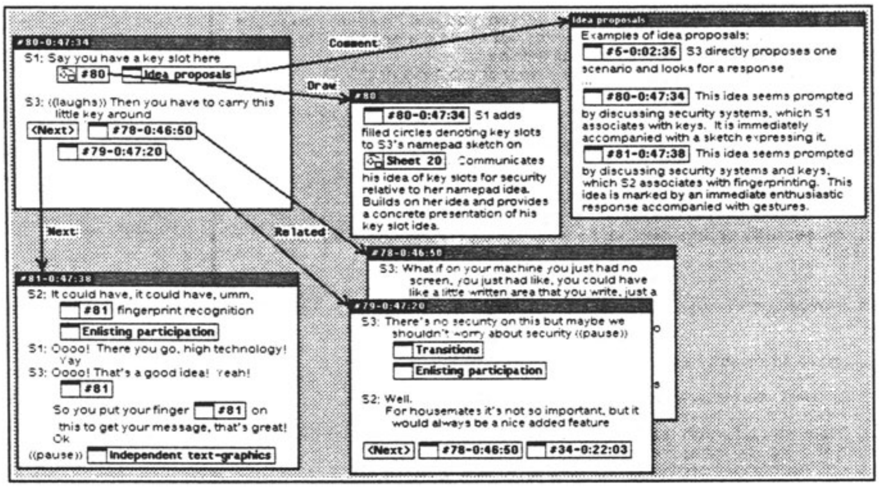
5.4.2. Representation of the Activity for Analysis

Developing a representation for relating the verbal transcript, notes on non-verbal activity, and comments from other researchers and participants is a major methodological issue. In our studies of group design activity, the NoteCards software system was used to help manage this wealth of data and organize its analysis. NoteCards is a hypertext system that runs in the Xerox Lisp environment (Halasz et al., 1987). It is analogous to index cards, in that it encourages breaking data down into small units, called cards. These cards can be pieces of text, graphics, or other information representable in the Lisp environment. NoteCards provides mechanisms for linking and grouping these cards to facilitate organizing them. The cards can be connected by links, which can be designated by type (e.g., comment, related, next). Cards can also be grouped together into fileboxes. NoteCards offers several mechanisms for structuring, displaying, and navigating through large networks of cards and links. It also allows users to program functions to execute customized operations on the data.

An example will demonstrate both how NoteCards was used to develop a representation of the activity and how that representation was used in this analysis. After creating a transcript of the verbal dialog, the transcript was divided into segments. Each segment consisted of an interactional exchange over a particular focus of attention. When the group's attention shifted to a new focus, a division between segments was marked. The segments averaged less than a minute in length and typically comprised 3–7 turns of talk. No claims in the analysis are based on the definition of these segments. This segmentation was done to facilitate the analysis—to be able to distinguish, identify, and group together different segments of the activity. While many of the segment boundaries were clear-cut, some were rather arbitrary. An alternative method proposed by Fish (1988) divided the data into segments of fixed time intervals (i.e., 30-second segments) without attending to the content of the activity.

Each segment of transcript was placed on a separate card, and linked to the segment that followed it, creating a chronological chain of links through all the cards. Each segment was linked to other segments dealing with a related topic, or grouped together into fileboxes that collected segments exhibiting a common pattern of activity. Segments were also linked to comments by the researchers or participants that refer to some part of the activity included in the segment.

As mentioned earlier, our initial work on making a transcript and ana-



#80-0:47:34
S1: Say you have a key slot here
#80 Idea proposals
S3: ((laughs)) Then you have to carry this
little key around
<Next> #78-0:46:50
#79-0:47:20

#80
#80-0:47:34 S1 adds
filled circles denoting key slots
to S3's namepad sketch on
Sheet 20 Communicates
his idea of key slots for security
relative to her namepad idea.
Builds on her idea and provides
a concrete presentation of his
key slot idea.

Idea proposals
Examples of idea proposals:
#6-0:02:36 S3 directly proposes one
scenario and looks for a response
...
#80-0:47:34 This idea seems prompted
by discussing security systems, which S1
associates with keys. It is immediately
accompanied with a sketch expressing it.
#81-0:47:38 This idea seems prompted
by discussing security systems and keys,
which S2 associates with fingerprinting. This
idea is marked by an immediate enthusiastic
response accompanied with gestures.

#81-0:47:38
S2: It could have, it could have, umm,
#81 fingerprint recognition
Enlisting participation
S1: Oooo! There you go, high technology!
Yay
S3: Oooo! That's a good idea! Yeah!
#81
So you put your finger #81 on
this to get your message, that's great!
Ok
((pause)) Independent text-graphics

#78-0:46:50
S3: What if on your machine you just had no
screen, you just had like, you could have
like a little written area that you write, just a

#79-0:47:20
S3: There's no security on this but maybe we
shouldn't worry about security ((pause))
Transitions
Enlisting participation
S2: Well.
For housemates it's not so important, but it
would always be a nice added feature
<Next> #78-0:46:50 #34-0:22:03

FIGURE 5.4. How segments are linked to other objects. Arrows from the link icons indicate how a segment from a design session transcript is linked to other segments, notes on the workspace activity, and comments of analysis.

lyzing the data as a group led to a focus on the listing, drawing, and gesturing activity that occurs in collaborative design work. Portions of the videotaped data were selected to investigate these drawing space activities more intensively. For one entire $1\frac{1}{2}$ hour design session and a 10-min section of a second design session (where the group specified a design for one of their ideas), each instance of listing, drawing, and gesturing was described on an individual card. Each transcript segment was annotated by links to those cards noting any instances of listing, drawing, or gesturing that occurred during that segment. A sample segment from the transcript of Figure 5.3 and the cards that it is linked to are shown in Figure 5.4. In this way, NoteCards was used to manage and keep track of a variety of information, comments, and relationships among the empirical data.

5.4.3. Abstracting Observations from the Data

The goal of this analysis is to identify generalizable observations about design activity from the videotaped data. One strategy in this analytical process is to look for "collectibles"—recurring patterns of activity that can be collected throughout a session or across a variety of sessions. This strategy is a common technique in conversation analysis [see for example Levinson (1983)] that has been extended to interaction analysis (Tatar, 1989). Patterns of activity were identified and other examples of that pattern were collected. Comparing and contrasting among several different examples (while being sensitive to the contexts in which they were situated) leads to a better understanding of that activity.

In particular, our interest in analyzing group design activity was to identify implications for the design of tools to support that activity. We focused on identifying collectibles that led to an understanding of what resources the designers used or what obstacles they encountered in accomplishing their work. Analyzing these collectibles led to an understanding of specific resources and obstacles for the designers.

For example, one pattern of activity identified as a collectible in our study of group design activity was the use of hand gestures. Many instances of the use of hand gestures were collected from the recorded design sessions. This collection of data raised several research questions:

- What did these hand gestures accomplish?
- What relationship did these gestures have with the group's other ongoing activity (e.g., talking, drawing)?
- What problems arose from the use of these hand gestures?

Comparing and contrasting among this collection of data led to an understanding of what resources and obstacles are associated with gestures. For example, the relationship of gestures to the drawing space is a resource for interpreting them, since gestures often refer to marks in the drawing space. On the other hand, visual obstructions that prevent collaborators from sharing a view of their gestures can be an obstacle. These observations are discussed in more detail in the next section.

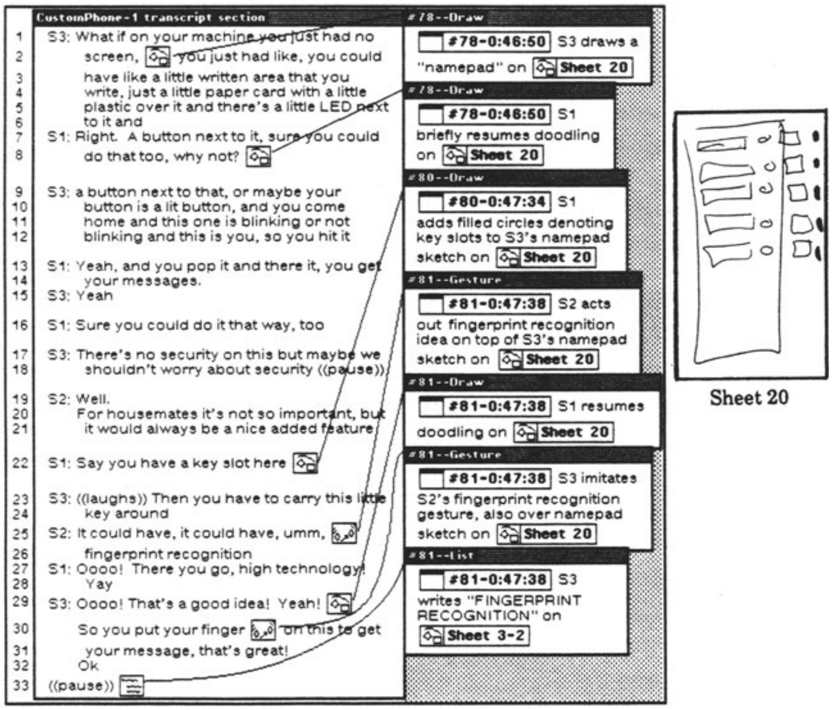
In summary, the analysis consists of:

- identifying specific patterns of activity of interest
- collecting instances of that activity in a variety of situations
- comparing and contrasting among the collected instances to explain the activity and its variation across different situations

The advantage of this approach is that the resulting observations are closely tied to the empirical data. It is the data that initially suggest the collectibles and groupings, rather than hypothesized groupings being imposed onto the data. The disadvantage of this approach is that it is very time consuming. Careful attention is required to identify collectibles and to collect enough relevant instances of each collectible for analysis. Much qualitative analysis is needed to compare and contrast among the collected instances in order to gain an understanding of the activity that leads to generalizable observations. Tang (1991) describes observations that were raised in using this methodology to study group design activity.

5.5. Findings: The Uses of Hand Gestures

One issue that emerged from analyzing the data was understanding the use of hand gestures. There is a long history of studying gestures in human interaction [see for example Goodwin (1986) and Kendon (1986)] and the prevalence of gestures in collaborative design activity is obvious. Our re-



1 S3: What if on your machine you just had no
2 screen, you just had like, you could
3 have like a little written area that you
4 write, just a little paper card with a little
5 plastic over it and there's a little LED next
6 to it and
7 S1: Right. A button next to it, sure you could
8 do that too, why not?
9 S3: a button next to that, or maybe your
10 button is a lit button, and you come
11 home and this one is blinking or not
12 blinking and this is you, so you hit it
13 S1: Yeah, and you pop it and there it, you get
14 your messages.
15 S3: Yeah
16 S1: Sure you could do it that way, too
17 S3: There's no security on this but maybe we
18 shouldn't worry about security ((pause))
19 S2: Well.
20 For housemates it's not so important, but
21 it would always be a nice added feature
22 S1: Say you have a key slot here
23 S3: ((laughs)) Then you have to carry this little
24 key around
25 S2: It could have, it could have, umm,
26 fingerprint recognition
27 S1: Oooo! There you go, high technology!
28 Yay
29 S3: Oooo! That's a good idea! Yeah!
30 So you put your finger on this to get
31 your message, that's great!
32 Ok
33 ((pause))

#78--Draw
#78-0:46:50 S3 draws a "namepad" on Sheet 20

#78--Draw
#78-0:46:50 S1 briefly resumes doodling on Sheet 20

#80--Draw
#80-0:47:34 S1 adds filled circles denoting key slots to S3's namepad sketch on Sheet 20

#81--Gesture
#81-0:47:38 S2 acts out fingerprint recognition idea on top of S3's namepad sketch on Sheet 20

#81--Draw
#81-0:47:38 S1 resumes doodling on Sheet 20

#81--Gesture
#81-0:47:38 S3 imitates S2's fingerprint recognition gesture, also over namepad sketch on Sheet 20

#81--List
#81-0:47:38 S3 writes "FINGERPRINT RECOGNITION" on Sheet 3-2

FIGURE 5.5. Annotated transcript section from design session. A section of transcript from a design session, linked to notes on the instances of listing, drawing, and gesturing that occurred. The area of the paper being worked on during this section is shown at the right.

search focused on what gestures accomplish in group design and how they could be supported by collaborative tools. We observed that gestures can be used to: enact a simulation of an idea; help mediate the group's interaction; and possibly help store information. An important feature of gestures is their relationship to drawings and other objects in the drawing space. These observations are illustrated with an annotated transcript representing a scene from the video data, shown in Figure 5.5.

5.5.1. *Scene from the Video Data*

The section of transcript shown in Figure 5.5 is annotated with brief descriptions of every instance of listing, drawing, and gesturing that occurred during the section. An icon is placed in the text of the transcript at approximately the point where the listing, drawing, or gesturing activity begins. That icon is linked to a note describing the activity. The line numbers along the left margin are used throughout this section to index locations in the transcript.

The region of the paper where the participants are making their marks and sketches is included to the right of the transcript.

The designers have chosen to design a custom phone answering machine to service a household that has several different inhabitants (see Appendix for complete problem statement). At this stage of the session, they have established that the answering machine routes incoming phone messages to particular recipients in the household. In this section, they talk about how those recipients retrieve their phone messages, and especially how they could prevent their own messages from being accessed by others.

In this section, S3 first proposes a "namepad" configuration where each recipient has a slot and can select to hear their own phone messages when a flashing LED indicates that their slot has messages. However, S3 realizes that this solution does not prevent other people from accessing the phone messages directed to a particular person, a security issue that the group had previously raised. S1 proposes that each button could be locked with a key. Then S2 proposes that the machine sense the person's fingerprint when pressing the button to access messages, and recognize whether to grant access to them or not. This idea gets an enthusiastic response, culminating in S3 imitating the fingerprint recognition gesture and documenting it.

5.5.2. *Observations on the Use of Hand Gestures*

One observed use of hand gestures is to enact ideas that involve a dynamic sequence of actions. Hand gestures can be an effective way to express these ideas to other group members. For example, in the gesture noted in line 25 of the transcript in Figure 5.5, one designer acts out the fingerprint recognition idea. This gesture is shown in Figure 5.6. By holding her finger over a button on the sketch of the phone machine, S2 demonstrates how she imagines the phone machine recognizing her fingerprint and subsequently playing her phone messages. Enacting a sequence of actions through gestures is a convenient way of demonstrating behavior, especially how people will interact with the design. These gestures range from abstract motions to more detailed enactments, often done in relation to existing sketches or other objects in the drawing space.

Hand gestures are also commonly used to mediate the interaction of a group, such as raising a hand to indicate wanting the next turn in the conversation. As part of the gesture marked in line 25 of the transcript, S2's hand moves deliberately toward the namepad sketch, effectively commanding a turn in preparation for her acting out the fingerprint recognition idea. Gestures are also used to direct the group's attention by pointing to or otherwise referring to drawings or areas in the drawing space.

Gestures are not typically thought of as a medium for storing information because they do not leave behind any persistent record. However, the data showed some evidence that information can be chunked and preserved effectively through gestures, especially if the gesture is imitated by others and

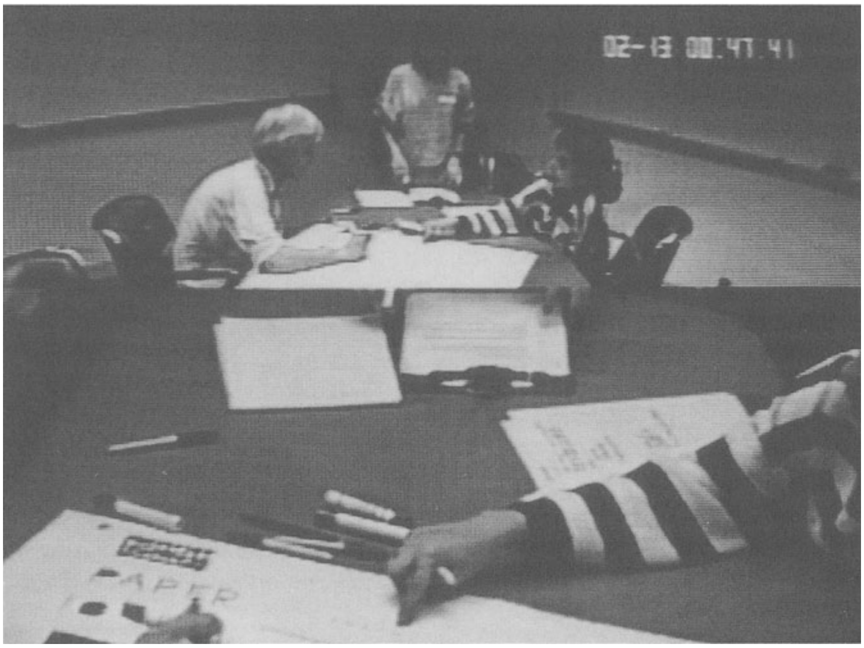


FIGURE 5.6. Gesture example. S2, on the far right, enacts the fingerprint recognition idea by pressing her finger on a sketch of a button.

labeled in text or graphics. For example, on line 30 of the transcript, S3 imitates S2's gesture of the fingerprint recognition idea from line 25. The idea is later written down by S3, as noted in line 33, but the essence of the idea is encoded in the gesture, which is not otherwise persistently documented. The fact that the fingerprint recognition idea is not readily apparent just by looking at the marks made in the workspace is evidence that much of the idea is not preserved except through the gesture.

A most important characteristic of hand gestures is that they are typically made in relation to existing objects in the drawing space. Gestures that enact an idea are often acted out in the context of a sketch or other object in the drawing space (e.g., the fingerprint recognition gestures over the namepad sketch on lines 25 and 30). Gestures are often used to direct the group's attention by referring to sketches or other objects (e.g., pointing to another group member) in the drawing space. These observations indicate that it is important to not only see the gesture, but also to see it in relation to the workspace and the other participants.

One observed problem concerning gestures is that they are sometimes not perceived by other team members, because their attention is focused elsewhere. Being able to clearly view gestures can be difficult, especially in meetings with many participants. Meetings in computer-augmented rooms [e.g., Colab (Stefik et al., 1987)] that are cluttered with computer equipment, or meetings involving participants in physically remote locations present greater challenges in sharing gestures.

Tools could be applied to convey gestures so that all of the participants can share in viewing them. Such tools should also preserve the relationship between gestures and their referents in the shared drawing space. VideoDraw (Tang and Minneman, 1991) is an example of a prototype tool that uses video to convey gestures in support of collaborative drawing activity. Hand gestures are captured by a video camera aimed at the drawing surface. This video image is presented as part of the shared drawing surface that the other collaborators view, so that everyone can see those gestures and see them in relation to the marks that they refer to on the drawing surface.

5.6. Advantages and Constraints of the Methodology

Video-based interaction analysis is a useful methology for studying human activity. Studying how people actually accomplish an activity leads to a better understanding of the resources and hindrances that exist for the participants and suggests the design of tools to augment those resources while eliminating obstacles in their work. This methodology results in an analysis that is strongly tied to examples from realistic work activity.

Interaction analysis enables a new understanding of design activity that cannot be obtained by the previously discussed methods that have been applied to study it. For example, with respect to studying hand gestures, interaction analysis has enabled an understanding of how gestures are used in the context of collaborative design, leading to specific design implications for tools to support that activity. Psychological experiments would have studied gestures in isolation, possibly missing the importance of the relationship between gestures and their referent sketches. Protocol analysis would depend on people being sufficiently aware of their use of gesture to report on it in their thinking aloud. Yet, it is because gestures are so naturally and effortlessly used that they are an effective resource for designers in collaboration. The time scale of participant observation studies would not lend themselves to focusing on the role of hand gestures in the design process.

However, video-based interaction analysis has some constraints that suggest when it is and is not appropriate to use. Interaction analysis is limited to observing a tractable time period of activity (typically hours, rather than weeks or months). This may seem like a limited amount of observed activity, yet it contains a wealth of data that requires a large amount of time to analyze. Consequently, only a limited sample of activity can be studied using this fine grained analysis.

A related concern is how the observations gained from this methodology can be generalized. Certainly, other kinds of activity might occur under different situations than those observed. Thus, it is important to present the findings in terms of the context in which they were observed. Those findings that are based on evidence that goes beyond that particular context (such as the observations reported here on the use of hand gestures) can be more broadly generalized. However, some findings will be more dependent on the specific context (e.g., that only one person tends to work at the chalkboard at a time), and can only be generalized to certain similar contexts.

A concern that is often raised in observational studies such as these is that observing the activity may affect the activity itself. There is evidence in the psychology literature that the initial effects of being observed fade quickly with time (Kelley & Thibaut, 1969, p. 6). There is no rigorous test that can determine the effects of being observed. We assert that the passive observational method presented in this paper is less disrupting than the controlled experimental and protocol analysis methods used in other design studies. In the sessions that we have observed, there were only isolated references to the fact that the participants were being videotaped ("Don't mind the 'explosive' television cameras", "Oh I did that on TV"); otherwise the activity was focused on the design task. Besides these isolated references, there was no visible evidence that the observation affected the group's activity.

5.7. Applying this Methodology in the Design Process

In applying interaction analysis to study group design activity, we discovered that it could be used not only to *study* the design process but also *as part of* the design process. In the research reported in this paper, the work activity of design teams is studied in order to help develop tools to better support group design activity. This research models a design process where the designers first understand the needs of their end users (which in this case are designers engaged in group work) before building tools to support the users' work. Applying interaction analysis to study the activity of the target end users could be used in *any* design process to understand the users' needs and guide the design of tools to support their work activity.

While designers are often encouraged to understand the users' needs and design technology that meets those needs, the designers are typically not equipped with any methodologies to help them accomplish this need-finding. Interaction analysis could be applied to study the work activity of target end users in order to help designers identify what resources are used and what hindrances are encountered by their target users. This understanding could help guide the designers in designing technology that augments resources while eliminating hindrances in users' work. In this way, interaction analysis can be an integral part of the design process.

When applying this methodology *as part of* the design process, a troublesome concern arises. Since this methodology depends on observing actual interaction with an artifact, it is difficult to apply it to the design of future technology that does not yet exist. The participants must have an artifact of some form to interact with in order to use this methodology to observe their interaction with it. A starting point is to study a related work activity in order to understand where to begin intervening with new technology. The research presented in this chapter an example of that approach: collaborative design activity using conventional tools (paper, pen, chalkboard) was studied in order to guide the design of new tools to support that activity.

Additionally, a rapid prototyping design approach that functionally prototypes or simulates the imagined new technology can give some indication of how the users will interact with it. Vertelney (1989) describes some techniques using computers and video to quickly prototype user interfaces. By iterating between observing prototypes in use and developing new prototypes, a new technology can emerge that is designed to fit the needs and capabilities of its users. Early experiences in applying the observational methodology as part of the design process to understand the needs of users are reported by Tatar (1989), Tang et al. (1990), and Suchman and Trigg (1990).

5.8. Conclusions

Video-based interaction analysis is a qualitative methodology that can be used to study group design activity. This methodology results in a descriptive analysis of the activity, leading to an explanation and understanding of how the group accomplished their work. It has been applied to study the collaborative drawing activity of design teams (Tang, 1989). In this research, the methodology identified prominent features of group workspace activity (e.g., gestures, the process of creating drawings) and a better understanding of specific aspects of those features (e.g., the relationship of gestures to the workspace, the use of the drawing process to mediate interaction). Using this methodology to study design activity leads to a better understanding of the design process.

This methodology can also be used as part of the design process, to understand the needs of the users and guide the design of technology to meet those needs. In our studies of collaborative design activity, this methodology helped identify specific implications for the design of tools to support that activity. Using this methodology as part of the design process leads to the design of better artifacts that fulfill users' needs.

Acknowledgments

We would like to acknowledge Lucy Suchman for her consistent guidance and support, and for introducing us to a new perspective on studying design activity. Austin Henderson provided extensive comments on an earlier draft of the paper. We also thank Sara Bly, Chuck Goodwin, Scott Minneman, and Deborah Tatar for their insights and discussions that they shared in this research. We acknowledge the Xerox Palo Alto Research Center (PARC) for funding this research, and the members of the Interaction Analysis Lab there for their comments and help in analyzing the videotapes. The Center for Design Research at Stanford University and the System Sciences Laboratory at Xerox PARC provided environments that fostered this research.

Appendix

Problem statement for the design session:

In teams of three or four, design a custom multifunction telephone for the user and environment of your choice.

It should have at least three of the following functions: auto-dial and redial, answering machine, calendar and clock, log or diary, call waiting and forwarding, hold and transfer, conferencing, call-back, speaker-phone or any other you might think of relevant to your particular user(s).

The goal of this project is for you to be able to design complex computer-based products which are easy, efficient, safe and satisfying to use. You should be able to use scenarios to describe users and environments, task analysis to determine information needs, keystroke models to predict efficiency and simple prototypes and storyboards to check learning.

References

Bessant, J. R., & McMahon, B. J. (1979). Participant observation of a major design decision in industry. *Design Studies*, *1*(1), 21–26.

Bessant, J. R. (1979). Preparing for design studies: ways of watching, *Design Studies*, *1*(2), 77–83.

Bucciarelli, L. L. (1988). An ethnographic perspective on engineering design. *Design Studies*, *9*, 159–168.

Darke, J. (1979). The primary generator and the design process. *Design Studies*, *1*, (1) 36–44.

Finger, S., & Dixon, J. R. (1989). A review of research in mechanical engineering design. Part I: Descriptive, prescriptive, and computer-based models of design processes. *Research in Engineering Design*, *1*(1), 51–67.

Fish, R. S. (1988). "Comparison of remote and standard collaborations. *Conference on Technology and Cooperative Work*, Tucson, AZ, pp. 1–11.

Frankel, R. M., & Beckman, H. B. (1982). IMPACT: An interaction-based method for preserving and analyzing clinical transactions. In L. Pettigrew (Ed.), *Explorations in Provider and Patient Interactions*, Nasvhille: Humana, Inc.

Goodwin, C. (1981). *Conversational Organization*: *Interaction between Speakers and Hearers*. New York: Academic Press.

Goodwin, C. (1986). Gestures as a resource for the organization of mutual orientation, *Semiotica*, *62*(1/2), 29–49.

Halasz, F. G., Moran, T. P., & Trigg, R. H. (1987). NoteCards in a nutshell. *Proceedings of the Conference on Computer and Human Interaction and Graphics Interface* (*CHI* + *GI*). Toronto, pp. 45–52.

Heath, C. (1986). *Body Movement and Speech in Medical Interaction*, Cambridge, MA: Cambridge University Press.

Kelley, H. H., & Thibaut, J. W. (1969). Chapter 29: Group problem solving. In G. Lindzley and E. Aronson (Eds.), *The Handbook of Social Psychology*; *Volume Four*: *Group Psychology and Phenomena of Interaction*. Reading, MA: Addison-Wesley Publishing Company, pp. 1–101.

Kendon, A. (1986). Current issues in the study of gesture. In Jean-Luc Nespoulous, Paul Perron, and Andre Roch Lecours (Eds.), *The Biological Foundations of Gestures*: *Motor and Semiotic Aspects*, Hillsdale, NJ: Lawrence Erlbaum Associates, pp. 23–47.

Latour, B., & Woolgar, S. (1979). *Laboratory Life*: *The Social Construction of Scientific Facts*, Beverly Hills, CA: Sage Publications.

Levinson, S. C. (1983). Conversational structure, *Pragmatics*, Cambridge, Cambridge University Press, pp. 284–370.

Lynch, M. (1985). *Art and Artifact in Laboratory Science*: *A Study of Shop Work and Shop Talk in a Research Laboratory*. London: Routledge & Kegan Paul, 1985.

Sacks, H., Schegloff, E., & Jefferson, G. (1974). A simplest systematics for the organization of turn-taking for conversation, *Language*, *50*, 696–735.

Stefik, M., Foster, G., Bobrow, D. G., Kahn, K., Lanning, S., & Suchman, L. (1987). "Beyond the chalkboard: Computer support for collaboration and problem solving in meetings. *Communications of the ACM*, *30*(1), 32–47.

Suchman, L. A. (1987). *Plans and situated actions*: *The problem of human-machine communication*. Cambridge, MA: Cambridge University Press.

Suchman, L. A., & Trigg, R. H. (1990). Understanding practice: Video as a medium for reflection and design. In J. Greenbaum and M. Kyng (Eds.), *Design at Work*, Hillsdale, NJ: Lawrence Erlbaum Associates, pp. 65–89.

Tang, J. C. (1989). *Listing*, *drawing*, *and gesturing in design*: *A study of the use of shared workspaces by design teams*, Xerox PARC Technical Report SSL-89-3 (Ph.D. Dissertation, Stanford University).

Tang, J. C. (1991). Findings from observational studies of collaborative work." *International Journal of Man-Machine Studies*, *34*(2), 143–160.

Tang, J. C., & Minneman, S. L. (1991). VideoDraw: A video interface for collaborative drawing. *ACM Transactions on Information Systems*, *9*(2), 170–184.

Tang, J. et al. (1990). Observations on the use of shared drawing spaces. Videotape, Xerox Corporation, Palo Alto Research Center, 1990.

Tatar, D. (1989). Using video-based observation to shape the design of a new technology. *SIGCHI Bulletin*, *21*(2), 108–111.

Thomas, J. C., & Carroll, J. M. (1979). The psychological study of design. *Design Studies*, *1*(1), 5–11.

Ullman, D. G., Stauffer, L. A., & Dietterich, T. G. (1987). Toward expert CAD, *Computers in Mechanical Engineering*, *6*, 56–70.

Vertelney, L. (1989). "Using video to prototype user interfaces. *SIGCHI Bulletin*, *21*(2), 57–61.

Wallace, K. M. (1987). Studying the engineering design process in practice, In Nadler G. (Ed.), *International Congress on Planning and Design Theory: Plenary and Interdisciplinary Lectures*, Boston, MA, pp. 29–34.

Wallace, K. M., & Hales, C. (1987). Detailed analysis of an engineering design project. In Eder W. E. (Ed.), *Proceedings of the 1987 International Conference on Engineering Design, WDK 13*, Vol. 1, Boston, MA, pp. 94–101.

6 Representation of Conceptual Mechanical Design Knowledge

Albert Esterline, Megan Arnold, Donald R. Riley, and Arthur G. Erdman

Abstract. Conceptual design is typically not well represented by traditional engineering mathematics. This work is concerned with eliciting and representing the knowledge used in the conceptual stage of mechanism design. This is the first stage of design, and, along with formulating the problem, establishes a function structure and selects processes and geometries for components realizing the functions. A formally based representation is developed that reveals conceptual connections and explicates terms and their valid patterns of use. The formalisms are largely adopted from theoretical computer science. Two knowledge components are formulated: one reveals the designer's view of the problem as it evolves, and the other captures aspects of control and strategy. The reliability of these schemes is discussed and characteristics of limited conceptual design are identified. We describe our methods of collecting and encoding protocols and discuss how our formalisms could underlie a software toolkit for acquiring and representing conceptual mechanical design knowledge. Finally, we relate our formalisms to paradigms of conceptual design.

1. Introduction

This work is concerned with one tool used in the application of AI to design: protocol analysis [(see, for example, Ullman and Dietterich (1988) and Waldron and Waldron (1989)]. A protocol is a record of a problem-solving session in which the subject thinks aloud. Analyzing the protocol reveals the concepts brought to bear in the problem area and the inferential relationships among the concepts. Protocol analysis is thus a primary tool for acquiring knowledge used in knowledge-based systems (KBSs). We are concerned with the analysis of mechanical design protocols and especially with the *conceptual* stage of design (Pahl and Beitz, 1984), which establishes a function structure and chooses the physical processes and geometries for components realizing the functions. Conceptual design is followed by *embodiment* design, which determines the layout. The problems we present to our subjects are intermediate in the innovative-routine spectrum of design (Brown, 1985). Initially, the function structure is only roughly known, and there is no hint of the structure of the solution, yet our subjects solve them in 1 or 2 h. While the problem-solving strategies are not known in advance, they are largely determined by the designer's understanding of the problem. We call the sort of design revealed in our protocols *limited conceptual mechanical design.*

This chapter presents a constellation of formally based schemes to represent what is revealed in these protocols and is one step toward integrating "scruffy" and "neat" aspects of AI as they apply to design (Esterline et al., 1989). The fact that knowledge-based design approaches generally ignore problem formulation originally motivated our use of protocol analysis. Our study of problem formulation led to the entire conceptual stage. Indeed, insufficient attention is given to how a problem is initially elaborated even though some sort of structure is assumed by AI approaches to design, such as constraint propagation (Sussman and Steele, 1989), qualitative simulation, and case-based reasoning [see, for example, Riesbeck and Schank (1989) and Sycara and Navinchandra (1989)]. We emphasize the importance of a sound representation, which is reliable in the sense that two people encoding the same protocol into the representation tend to agree, which also allows one to judge the similarity of design approaches and to evaluate prescribed techniques against practice, and which, finally, suggests testable generalizations.

A formally based representation is desirable since we wish to capture conceptual relations that allow the design to carry through and we wish to explicate with computational notions the terms we introduce. In addition to predicate logic, we borrow formalisms from theoretical computer science (denotational semantics, formal language theory, and Petri nets). Note that we use computational formalisms for *representing* a design problem and its evolution. This shows an emphasis different from that traditionally associated with computers in engineering, where typically one attempts to find efficient algorithms for solving problems that can be handled with simple data structures. The formalisms we use come from programming language semantics and the study of concurrent systems. They have been used in software engineering, where representing *what* is required or specified, rather than stating *how* to compute the solution, is emphasized. Knowledge representation in AI has drawn some from formal areas of computer science. Formulating results in computational formalisms not only enhances rigor but also makes these results more accessible to implementation in software, thus promoting the AI goal of modeling intelligent activity computationally. In our case, results become accessible for design automation, the ultimate practical goal of our work.

Our attempt to represent the problem aspects that are significant in a sequence of design steps may be compared with the drill students in philosophy and linguistics go through translating English sentences into logical formulas. Since the goal of this drill is to determine whether an inference is valid, the task is to reveal a sentence's logical form relevant to the inference at hand. The level of detail in this logical form depends on the inferential mechanisms involved. Often detail in the English sentence is suppressed, but often what is only implicit in the English sentence must be explicitly represented in the formalism. Likewise, in encoding design protocols, we capture the conceptual relations that allow the design to carry through; a major goal is to encode at the appropriate level, and this requires encodings of protocol fragments to be both backward- and forward-looking. A related issue is the vocabulary to use in the encoding. On the one hand, we would like to use a uniform vocabulary with a mathematical flavor for all protocols. This would allow conceptual relations to be expressed in a uniform and perspicuous manner. On the other hand, we would like to keep contact with the designer and the design context by using terms from the protocol. We have accepted a compromise.

We first present the *structured instance diagram (SID)*, which represents the designer's view of the problem state evolving through time. Static aspects of this diagram (the "static ontology") are discussed in Section 2, while dynamic aspects (the "dynamic ontology") are discussed in Section 3. Section 4 formulates the static and dynamic ontologies in the domain equations of denotational semantics. These ontologies can be viewed as type systems such that a SID is composed of interrelated instances of items declared in the ontologies and values associated with these instances. Section 5 presents our representation of control knowledge in terms of modified Petri nets. These nets consist of "foci" and transitions among foci, which are controlled by conditions relating to a SID. Each focus has as its name a phrase that describes the focused, coherent aspect of design represented by the focus. Focus names and atomic conditions are described by formal languages (grammars) to ensure that they are formally well-defined.

Note that a SID is specific to a particular protocol. The ontologies, in contrast, are type systems that govern the construction of any SID. The design nets are likewise general. When a protocol is encoded, a document called a *trace* is produced that partitions the protocol into *episodes* identified with net transitions. The conditions and changes in the SID associated with each episode are noted, and the SID is accordingly updated. As a protocol is encoded, new items may need to be added to the ontologies to sanction needed constructs in the SID and new foci or transitions may need to be added to the nets to reflect new kinds of transitions, but these additions are permanent and are available when future protocols are encoded.

We defer to Section 6 a detailed discussion of our encoding process and its reliability so that the formal framework presented in Sections 2–5 may be assumed. Section 7 discusses software support for our encoding schemes. We review several toolkits and indicate how a software toolkit for our schemes could borrow from these. A prior question is whether the general models underlying these toolkits are appropriate for protocol analysis in our domain. We assess in these terms the most influential toolkit, Shelley, and the KADS model of expertise underlying it. Section 8 addresses the difficult notion of strategies, conceived as abstract control patterns with scope greater than but encompassing that of the transitions in the design nets. We review the various computer-science formalisms used to capture the temporal order of events as possible ways to represent strategies in conceptual mechanical design, and again modified Petri nets are found most appropriate. Capturing strategies, however, strains our representational resources and, note, the resources of any formally based schemes. In section 9, we discuss two paradigms of design that together subsume most paradigms that have been advocated recently. We consider how these paradigms relate to and supplement our framework and especially how they interact with our representation schemes. Section 10 is the conclusion.

It is important to realize the limitations of the research reported in this chapter. We do not claim that protocol analysis is the only valid method for investigating conceptual mechanical design. Indeed, in Section 9.1 we describe a method (the "critical instance technique") we have developed to supplement our protocol analysis, and there are many other knowledge elicitation and acquisition techniques that could be applied to our domain. But we do claim that protocol analysis, along with representation of the knowledge thereby revealed, is the principle tool for investigating how conceptual design is performed. The spontaneous flow of a protocol is required to get a handle on the progressive structuring of the problem state that characterizes conceptual design. One devises representation schemes to capture this structure and its evolution and to express control patterns and strategies. In the first instance, knowledge elicitation is not appropriate here since it imposes representation schemes on the problem solving.

Again, problems remain with our schemes. The SID and allied notions are largely worked out, but there remain problems with the design nets, especially as these are intended to support our representation of strategies. There is, however, no need to follow our framework *in toto*. Protocol analysis is a very time consuming endeavor. We tend to select only the more revealing protocols and analyze them from start to finish. One could, however, skip or lightly analyze large parts of a protocol. Again, one could use only part of our representation scheme or use only part in outline. For example, one could construct only a skeletal SID for a protocol and attach comments to the structure it presents.

What is presented in this chapter is not an account of conceptual mechanical design but rather a *framework* of techniques and especially representation formalisms. (Still, this framework sets limits for an acceptable account.) The ultimate practical goal of these formalisms is to facilitate automation of aspects of conceptual mechanical design. Again there is no need for the representations to be automated *in toto*. For automation of conceptual design, the control patterns represented by the design nets, and especially strategies, are particularly important since conceptual design generally offers a multitude of options and picking up a viable line of reasoning is critical. We envisage such automation as human–machine cooperation in part because specifications often need to be added, modified, or refined.

To represent control patterns, we use formalisms intended to capture concurrency. This may seem odd since we usually think of a train of thoughts as serial, although perhaps this is because speech is necessarily serial. The concurrency assumed by the formalisms is the existence of concurrent sequences of events, and this does not imply that the individual events occur at the same time. The usual picture is an interleaving of sequences that is nondeterministic regarding whether certain events in one sequence precede or follow certain events in another sequence. We identify *episodes* in the protocol that correspond to transitions in the design nets. Using formalisms that admit concurrency allows us to look for patterns that abstract from insignificant ordering and fluctuations. Its is also useful to consider episodes that overlap or coincide. This is natural since lines of reasoning often overlap.

In this chapter, we discuss a significant number of formalisms and we describe our rather extensive representation schemes. Space restrictions, however, prohibit detailed discussions and all but a few examples. Generally, we give just enough information on the various formalisms to compare them on critical points and to highlight their strengths and weaknesses as means of representing aspects of conceptual mechanical design protocols. We give a few examples of skeletal SIDs, but we do not show examples of design nets; we give fragments of the semantic grammars for generating focus names and atomic conditions, and it is hoped that the reader will generate several focus names and atomic conditions to get a feel for the nets. We give the key references for each formalism discussed, and the interested reader is encouraged to refer to the appropriate references for a presentation of the formalism; the issues raised here should be kept in mind when the references are consulted. What inevitably is lost in our presentation is a clear picture of how the various formalisms support rigorous reasoning; the reader is asked to accept this on faith until he/she consults the references.

2. The Structured Instance Diagram (SID): A Representation of the Designer's View

The structured instance diagram (SID) is a representational device, constructed as a protocol is analyzed, that represents the designer's view of the problem state evolving through time. It makes explicit the *ontology* inherent in the problem state. Our graphical representation involving entities, relations and properties elaborates the entity-relationship model used in database modeling (Chen, 1976), although our representations denote *instances* of concepts (thus "structured *instance* diagram") and not sets of tuples. The SID is a framework for tabulating information about the things referred to (if only implicitly) in the protocol. It captures this information by relating instances of general concepts of entities, tasks, relations, and properties in a structured way that generalizes graphical representations using nodes and arcs. The SID can give a snapshot of the design at any specific stage. It also includes special relations (which appear as special kinds of arcs) that indicate how the design evolves. Section 3 discusses how the evolution of the design is represented; this section is restricted to the snapshot aspects of the SID. The elements of a SID are, in the first instance and as discussed below, formal elements that feature in domain equations of denotational semantics. They are secondarily elements of a graphical representation that is an intuitive tool allowing one not only to comprehend at a glance the major features of the protocol but also to encode all but the fine points of a protocol with a limited background in computer science formalisms. As a protocol is analyzed, the ontological items are tabulated in a structured way as indicated below. This tabulation translates directly into the graphical representation. Each item tabulated must be declared in the static ontology (discussed in Section 4), and the structure of the SID must be consistent with the type constraints imposed by this ontology. (The evolutionary aspects of the SID—discussed in Section 3—are declared in the dynamic ontology, also discussed in Section 4.) In this section, we first (in 2.1) present and illustrate the basic notions involved in a SID. The representation of time and change (in the task or artifact being designed) is an important special area and is discussed separately (in 2.2).

2.1. Basic Notions

The items represented in the SID include not only entities, properties, and relations, but also tasks.

- *Entities* (represented by rectangular nodes) include not only devices but also operands, obstacles, and more.
- *Tasks* (represented by hexagonal nodes) are functions required by the design.
- *Distinguished relations* (represented by special arcs) are usually among tasks or among entities and tasks.
- A *nondistinguished relation* (represented by a diamond node connected by arcs to the items it relates) typically (but not always) relates entities.
- A *property* (represented by an oval node connected by an arc to what it is a property of) is typically (but not always) a property of an entity.

In general, a relation could be among any kinds of items and a property could be of any kind of item. To relate items in the diagram to tabulated items, either names (preferably taken from the protocol) or reference numbers may be used.

Figure 6.1 presents a problem specification that we have given to designers: to design a device that will tie together bundles of plastic tubes.

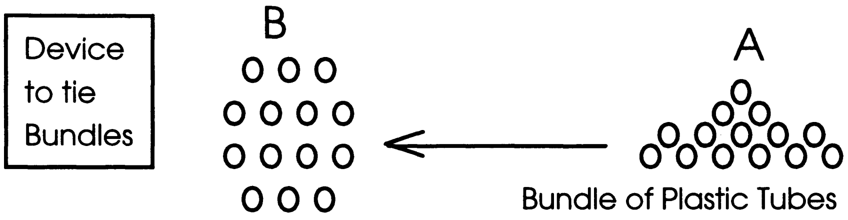


FIGURE 6.1. Problem specification. (1) A device is required to tie mechanically a bundle of flexible plastic tubes. (2) Each tube will be 12″ long and will have an outside diameter of 1/2″. The tubes will be presented to the device already loosely grouped in a bundle, as shown above at A. The bundle will always have 12 tubes. (3) The device should return the tied bundle to approximately the same place where the bundle was presented to the device (that is, A and B are at approximately the same location). (4) The device is to use the relative inexpensive paper enclosed wire "twist-ons" to tie the bundle of tubes.

Figure 6.2 is the tabulation of that part of the SID representing the information in this specification. This was completed after only the specification was read, but it is subject to revision in light of statements later made by the designer that indicate his/her interpretation of the specification. There are three tasks and two entities in Figure 6.2, each labeled with a unique positive integer. Item 1 is the main task of the problem: tying the bundle of tubes. Entity 2 is the bundle of tubes, task 3 returns the bundle, task 4 receives the bundle, and entity 5 is the twist-on used to tie the bundle. Entities and tasks are labeled with consecutive integers, beginning with 1, roughly in the order they appear in the protocol. Relations, properties, and most other information are listed under the appropriate entities and tasks. A relation is listed under its first argument. Additional properties and relations of a given task or entity may appear any time in the protocol, so it is advisable to list each entity or task on a separate sheet of paper. A reference in square brackets to one or more protocol fragments is included as justification at the end of a tabulated entry. The references in Figure 6.2 have the form [0.X] or [0.X, 0.Y]. "0" indicates the problem specification and X and Y indicate the specification (1–5) or the figure (fig) in the specification; the second form indicates that two specifications are relevant. Before a protocol transcription is encoded, it is divided into large sections numbered 0 (for the specifications) on up, and each section is segmented into phrases. (Partition into episodes, reflecting transitions in the design nets, comes later, as part of the encoding process.) Figures (supplied or drawn by the designer) are associated with protocol sections and are enumerated within these sections. A square-bracketed justification in a tabulated entry is a list of one or more references of the form M.X, where M is a section number and X is either the number of a segment or a reference to a figure in that section. (This form of reference to segments is also used to relate episodes to parts of the protocol text.)

1. Compact-typing (task) [0.1]
    - Operand (1, (2, 5)): the operands of 1 are 2 and 5 [0.1, 0.4]
        - $(\mathrm{WR}(1) \cap \mathrm{R}(2))_{\mathrm{some}} \neq \varnothing$ [0.1]
        - $(\mathrm{WR}(1) \cap \mathrm{R}(5))_{\mathrm{some}} \neq \varnothing$ [0.4]
2. Bundle (of flexible plastic tubes) (entity) [0.1]
    - Rel-loc (2, 5): the relative location of 2 to 5 [0.4]
        - initial: 2 distant from 5 [0.4]
        - $\rightarrow 1$ [0.4]
        - final: 2 tied to 5 [0.4]
    - a (2): orientation of 2 (quality) [0.fig]
        - value: The axis of 2 is horizontal and is normal to the direction 2 moves to and from the device realizing 1.
    - b (2): compactness of 2 (accident) [0.2]
        - initial: no [0.2]
        - $\rightarrow 1$ [0.2, 0.3]
        - final: yes [0.2, 0.3]
    - collection (2) [0.2]
        - collection(2) . cardinality (quality); value: 12
        - collection(2) . type (quality); value: tube
            - collection(2) . type . a: flexible (accident)
            - collection(2) . type . b: material (quality); value: plastic
            - collection(2) . type . length (quality); value: 12″
            - collection(2) . type . c: inside diameter (quality); value: 1/2″
    - length (2) [0.2] (quality); value: 12″
3. Returning (task) [0.3]
    - Operand (3, (2)): the operand of 3 is 2 [0.3]
    - end (1) $\leq$ end (3) [0.3]
    - Postrequisite (3, 1): 3 is a postrequisite of 1 [0.3]
4. Receiving (task) [0.3]
    - Operand (4, (2)): the operand of 4 is 2 [0.3]
    - begin (4) $\leq$ begin (1) [0.3]
    - Prerequisite (4, 1): 4 is a prerequisite of 1 [0.3]
5. Twist-on {tie} (entity) [0.4]
    - a (5): composition of 5 (quality) [0.4]; value: paper enclosed wire
    - dimensions (5) [0.4]; value:
        - ($>$ the perimeter of the cross section of $2_{\mathrm{final}(1)}$, $\leq$ 1–2″, negligible)
    - deformed-shape (5) (quality)
        - initial:
        - $\rightarrow 1$ [0.4]
        - final: the largest dimension is deformed to follow the perimeter of a cross section of $2_{\mathrm{final}(1)}$. [0.4]
    - b (5): tied (accident) [0.4]
        - initial: no [0.4]
        - $\rightarrow 1$ [0.4]
        - final: yes [0.4]

FIGURE 6.2. The tabular form of the SID for the specification in Figure 6.1.

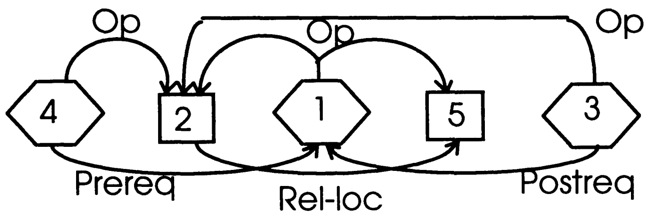


FIGURE 6.3. Graphical representation of major items in the SID tabulated in Figure 6.2. Prereq, Postreq, and Op stand for the relations Prerequisite, Postrequisite, and Operand. Rel-loc is the relative location relation. Tasks are represented by hexagonal nodes and entities by square nodes. The reference numbers refer to the tasks and entities as in Figure 6.2, viz.,1: Compact tying (task); 2: Bundle (entity); 3: Returning (task); 4: Receiving (task); 5: Twist-on (entity).

Figure 6.3 shows the major items in the graphical representation of the SID for the specification given in Figure 6.1. For simplicity, this suppresses much of the information in the tabular form of the SID presented in Figure 6.2.

A task generally has one or more *operands*, the entities on which it operates, and instances of the distinguished relation *Operand* are listed under the appropriate tasks in Figure 6.2. For example, under 1, there is *Operand* (*1*, (*2*, *5*)), indicating that the operands of 1 are 2 and 5. In general, the second argument is an *n*-tuple indicating that all *n* elements are operands of the first argument. The arguments of relations in general are listed as far as possible in the order they appear in their English reading, with the subject first; it is thus natural to tabulate a relation under its first argument.

We frequently use the term "relation" or "property" to mean an *assertion* that a relation holds among certain items or that a property (which mathematically is a unary relation) holds of an item. That is, we often use "relation" or "property" to mean an instance of the relation or property, with all argument positions filled in. In contrast, when we say that a relation is *distinguished*, we mean that the relation itself (with its argument positions unspecified) is accorded special status and its name is part of the established terminology. Names for other, nondistinguished relations must be coined as they are encountered. There are also distinguished properties. For example, *length*(*2*) refers to the length of entity 2 (the bundle). This has an associated value, 12″, and so it is what we term a *quality*. Other properties (such as the compactness of the bundle) are all-or-nothing (at least at this stage of the analysis): if one is asserted of an item, there is no need to specify a value for it. We call such properties (as with "quality," with apologies to Aristotle) *accidents*.

More explicitly, a quality f may be seen as a binary relation $f(\_,\_)$ that is a functional relation: given i (of the appropriate type), there is a unique v such that $f(i, v)$ holds. We could thus represent f explicitly as a function and write

$f(i) = v$. If we allow that i may not be of the appropriate type, then $f(i)$ may or may not be defined [that is, $f(\_)$ would be a partial function not necessarily total on this larger domain]. The values of i for which f is defined (that is, the values constituting the "appropriate type" for i) are the domain of definition of f. When we consider $f(\_)$ to denote a property, we take $f(i)$ to be true or false depending on whether i is in the domain of definition of the function $f(\_)$; if it is, then we may further ask about the value of $f(i)$, where $f(\_)$ is now treated as a function. In database terms, f is an attribute, i is a key (or denotes the individual uniquely determined by a key), v is the value of attribute f for the individual i, i is of the appropriate type if it is in the domain of the database relation in which f appears as an attribute (and possibly must also satisfy certain other integrity constraints), and v is of the appropriate type if it is in the domain of f [which, by an unfortunate linguistic twist, is the case if mathematically it is in the range of the function $f(\_)$]. A parallel analysis of an accident g shows that, in any instance $g(i)$ of g, g is an attribute, i is a key, and the only possible values for g are *true* and *false*. The usual database way to handle this is to have a database relation (that is, a table) in which i occurs as a key iff (if and only if) g holds of i. We thus see $g(\_)$ as a predicate without a corresponding function. Because the emphasis is on functions in denotational semantics, we shall indeed later view an accident as a function with range $\{true, false\}$. For now, however, we use the natural distinction that a quality has an associated (functional) value while an accident has not. The quality-accident distinction also applies to relations. For example, the *Operand* relation is an accident relation: *Operand*(I, L), where I is a task and L is a list of entities, is true or false, and no value is indicated if it is true. In contrast, *Rel-loc*(I, J), where, for example, I and J are entities, is true if there is a relation of relative location between I and J; if there is, then we may ask for an associated value, say, a direction and a distance.

Properties that are not distinguished are given consecutive lower case letters, starting with a, as names [see, for example, $a(2)$ and $b(2)$ under 2 in Figure 6.2] so the SID may be annotated without clutter. Each task or entity has its own sequence of names. Each nondistinguished property is accompanied in the tabulation by an English phrase to explain its meaning. Nondistinguished relations are treated similarly but with upper case letters since our convention is that a relation name begins with an upper case letter.

Properties of properties are represented by recordlike structures. As an example, *collection*(2) in Figure 6.2 asserts that the distinguished property *collection* holds of entity 2 (the bundle). Accident *collection* has properties *cardinality* (a quality whose value is the number of repeated parts) and *type*. The quality *type* has a value but also has its own properties. At this stage, we have three levels of properties, giving expressions such as *a*(*type* (*collection* (2))). Our conventions allow us to write this as *collection*(2).*type*.*a*. Property *collection*(2) also illustrates that, for an item that does not change throughout the protocol, a single reference can apply to that item and all its subordinates.

The values recorded for quality properties and quality relations in Figure 6.2 undoubtedly strike some as strange. The values that pass without question are real numbers, which may be associated with measurements. No measuring procedure, however, is defined early in the protocol and it would be a mistake to distort the protocol by overspecifying the problem state even if the additional precision were accompanied by such qualifications as intervals or probabilities. Sometimes the problem state, by virtue of the specification or a pronouncement by the designer, does afford a real number, as when *length*(*2*) in Figure 6.2 has the value 12″. Sometimes all that is available is something similar to a constraint, like the value for *a*(*2*), the orientation of the bundle. A value might have components, only some of which relate directly to real numbers—see the value of *dimensions*(*5*), the dimensions of the twist-on. In finding an appropriate level of detail at which to encode a protocol fragment, we try to decompose the problem state into ontological items that give the most direct translation consonant with the minimum structure that allows the content of subsequent fragments to be treated as an elaboration of the structure and constraints already present. Values are to be no more precise than the fragment warrants and no more decomposed than references later in the protocol require. One can always revise the encoding of a fragment in light of the detail needed to support later design activity. In fact, we have discovered that usually almost all elaboration can be attributed to new tasks and entities (and their properties and relations) that are introduced as the protocol evolves (see Section 3).

The values of qualities typically are the types of items that further qualify items that have already been identified. For example, an entity must be identified before the question of the numerical value (or any other value) for its length can arise. Our approach emphasizes from the start the ontological items that impart structure on the problem state. Within the structure, it locates restrictions and constraints; these lead to the numerical values exploited by familiar engineering techniques. Not only may a problem state fail to suggest values for familiar qualities, but it may also be mute about instances of certain very general and universally applicable properties and relations. For example, it may be pointless to assert a relation of relative location (*Rel-loc* in the terminology of Figure 6.2) between a certain pair of entities.

2.2. Representing Time and Change

It is essential to represent changes that certain items undergo. For example, the relation *Rel-loc*(*2*, *5*) in Figure 6.2 is a quality relation that changes value when the top-level task is performed. This is indicated by recording, in lieu of a single value, an *initial* value and a *final* value; the $\rightarrow$ 1 between these values indicates that the top-level task (1) intervenes. In general, the changing values of a relation or property are listed under it in chronological order, with tasks that contribute to the change from one value to another listed between them. Since an accident either holds or does not hold, the "values" for a changing accident are *yes* or *no*.

Tasks potentially exhibit the richest temporal nature. When a task is sufficiently specific, a time profile is listed under it; this has the form

initial:	⟨description⟩	[Ref_0]
e_1:	⟨description of e_1⟩	[Ref_1]
...		
e_n:	⟨description of e_n⟩	Ref_n]
final:	⟨description⟩	[Ref_{n+1}]

The e_i, $1 \leq i \leq n$, are events and are listed in the order they occur and [Ref_i] is a reference to the segment(s) in the protocol referring to event e_i. If the task is repetitive, then *final* is replaced by *repeat* and information about such things as period is included in the accompanying description. With constituent events identified, changes in relations and properties can be correlated more finely with events within tasks; we use the notation $e_i(I)$ for the *i*th event of the task whose reference number is *I*.

To state the spatial information available, *WR*(*I*) is used to denote the *working region* of the task with reference number *I*, the region where that task operates. Similarly, if *J* is the reference number of an entity, then *R*(*J*) denotes the region occupied by this entity. [Regions and working regions are histories in the sense of (Hayes, 1985).] To represent a temporal cross section of an item *A*, we use an expression of the form A_T. Here *T* is a temporal reference of the form *initial*(*I*), *final*(*I*), or $e_i(I)$, where *I* is the reference number of a task. Relations among regions and working regions, or temporal cross sections thereof, are expressed in set-theoretical notation. Examples are shown under task 1 in Figure 6.2, where the operator *some* (meaning sometime) is applied as a subscript to entire intersections; the first of the two statements asserts that *WR*(*1*) and *R*(*2*) intersect sometime.

Representations for temporal relations have been extensively studied in the AI literature and elsewhere [see, for example, Allen, (1983), McDermott, (1982), and van Benthem (1983)]. We use the usual relation symbol for total orders ($\leq$) and what can be defined in terms of it and equality to represent temporal relations among events that, at the current level of analysis, are thought of as happening at instants. Events thought of as occurring over time intervals are accommodated by introducing expressions *begin*(*Int*) and *end*(*Int*) to refer to the initial and final instants (or greatest lower bound and least upper bound) of interval *Int*. All these temporal relations are distinguished and are accidents with a mathematical flavor since there is an underlying total order. Indeed, there generally is significant temporal structure explicit in a design problem state. There are also distinguished relations with a temporal aspect that presuppose something more than a mere underlying order. For example, *Prerequisite*(*4*, *1*) in Figure 6.2 indicates that task 1 could not be done unless task 4 is done, and *Postrequisite*(*3*, *1*) asserts that a successful conclusion of task 3 be done. Another distinguished relation is *Causes*(I, X), where I is a task and X can be (the value of) a relation or property or could even be an entity. The last three relations are accidents and have temporal implications (for example, a cause cannot occur after its effect), but they also presuppose some underlying mechanism (usually still to be designed).

Most distinguished relations and properties have a mathematical flavor, yet, unlike temporal relations, are qualities (they have values). Even though the distinguished spatial relations (several of which appear in Figure 6.2) themselves have a mathematical flavor, their values, like the descriptions and values of nondistinguished relations and properties, tend at the beginning of the design process to be stated informally. The distinguished relations and properties, whose names are uniformly imposed independently of the terms used by the designer, frequently are only implicit at the beginning of the design process. The exceptions are temporal relations.

3. Evolution of the SID: Representing the Designer's Changing View

The SID thus far has been considered as it appears at one point in the design process. To capture its evolution as the design progresses, we introduce several distinguished relations. These relations are declared in the dynamic ontology, discussed (along with the static ontology—relating to the SID aspects discussed in Section 2) in Section 4. In this section, we first (in 3.1) present the distinguished relations in question along with some examples and introduce the notion of a "task DAG," which is a skeletal representation of a SID emphasizing evolutionary relations. As the designer's view evolves, alternatives arise and are evaluated. This requires additional representational resources, which are discussed in 3.2. This section concludes with a summary (in 3.3) of how the character of the SID changes as the design activity progresses.

3.1. Basic Notions

One way for a design to progress is for a task to be decomposed into subtasks. If I is a task and $I_1, \ldots, I_n$ are subtasks of I, then *Subtask*(I_i, I) holds for all i, $1 \leq i \leq n$. Another way for a design to progress is by introduction of an entity to realize (partially or wholly) a task. Sometimes a new, more specific entity is introduced to realize a less specific entity; a similar relation can obtain between tasks. Finally, an entity may require one or more tasks to be realized for it to perform as required. The distinguished relations we have just mentioned (after *Subtask*) are encoded as *Realizes*(E_1, T_1),

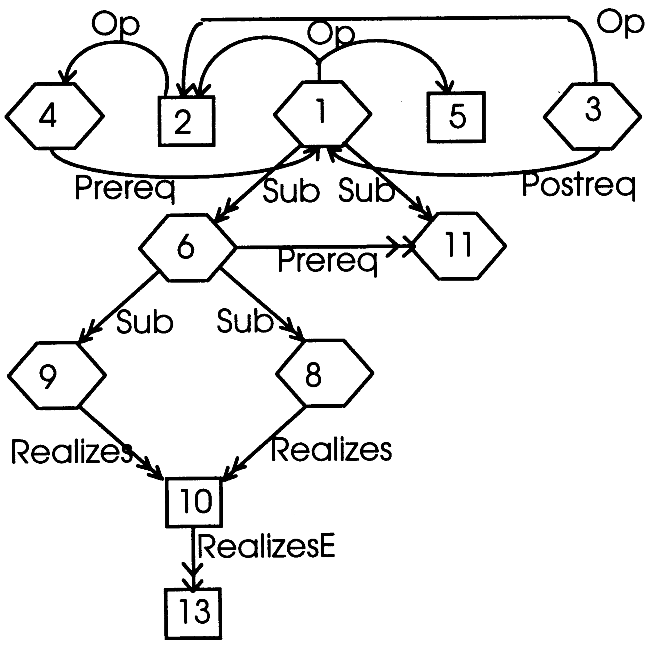


FIGURE 6.4. The major features of the initial expansion of the diagram of Figure 6.3, following A's protocol. Task 7 and entity 12 play a minor role and are ignored here. The labels Prereq, Postreq, Op, and Sub are abbreviations for the relation names Prerequisite, Postrequisite, Operand, and Subtask. The reference numbers refer to the tasks and entities as follows. 1: Compact-tying (task); 2: Bundle (entity); 3: Returning (task); 4: Receiving (task); 5: Twist-on (entity); 6: Compacting (task); 8: Gathering (task); 9: Jiggling (task); 10: Compacting device (entity); 11: Tying (task); 13: Two "C"-clamps (entity).

RealizesE(E_1, E_2), *RealizesT*(T_1, T_2), and *Sustains*(T_1, E_1), where T_1 and T_2 are tasks and E_1 and E_2 are entities. Note that, for fixed T_2, there is at most one T_1 such that *RealizesT*(T_1, T_2); if there were several tasks realizing T_2, these tasks would count as subtasks.

We illustrate these relations from our encodings of two protocols in which the problem shown in Figure 6.1 is solved. The first subject (henceforth called A) is a member of the mechanical engineering faculty at a major university. The second subject (henceforth called B) has more than 20 years industrial experience and is undegreed. At the level of detail shown, Figure 6.3 (the graphical version of the SID for the specification in Figure 6.1) is valid for both subjects. Figure 6.4 shows how this was initially expanded by designer A, and Figure 6.5 shows how it was by designer B. To emphasize the general structure of the diagrams, with two exceptions (the *tied* property and the *detached-part-of* relation in Figure 6.5), only tasks, entities, and destinguished relations are shown, and the *Rel-loc* relation shown in Figure 6.3 is

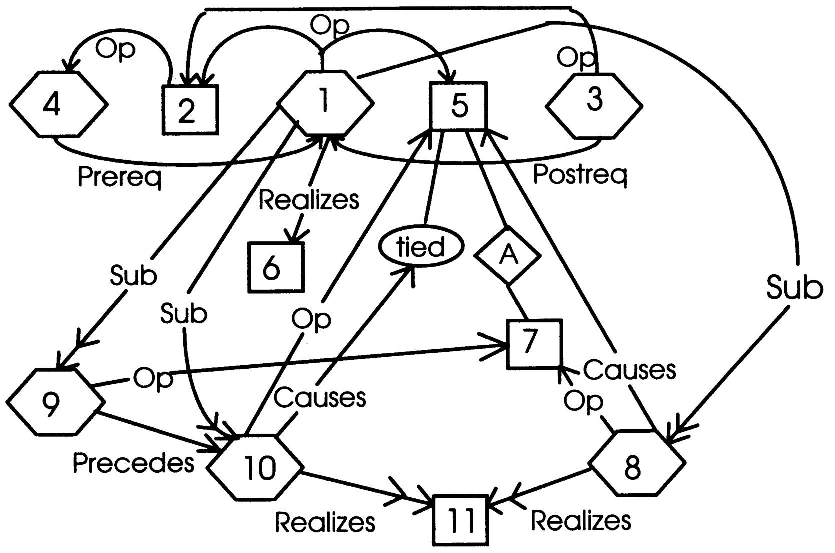


FIGURE 6.5. The major features of the initial expansion of the diagram of Figure 6.3, following B's protocol. The labels Prereq, Postreq, Op, and Sub are abbreviations for the relation names Prerequisite, Postrequisite, Operand, and Subtask. The reference numbers refer to the tasks and entities as follows. 1: Compact-tying (task); 2: Bundle (entity); 3: Returning (task); 4: Receiving (task); 5: Twist-on (entity); 6: Hopper (entity); 7: Wire (entity); 8: Detaching (task); 9: Feeding (task); 10: Winding (task); 11: Spindle (entity). The relational reference A(5, 7) indicates that 5 is a detached part of 7.

ignored. Relations that give the history of the design are shown with double-headed arrows.

Designer A decomposed the top-level task into a compacting task and a tying task, with compacting a prerequisite for tying. B, in contrast, decomposed the top-level task into a detaching (of the twist-on from a uniform length of paper enclosed wire) task, a feeding (of the wire through a groove on the inside of a hopper holding the bundle) task, and a winding (to tie the twist-on) task. The subtasks A chose exhausted the parent task, while those chosen by B did not. A decomposed the compacting subtask into a gathering (the tubes together) task and a jiggling (the bundle so it assumes a stable shape) task but then realized (see the relation *Realizes*) both tasks with a single entity, called simply a compacting devise; it was ultimately realized (*RealizesE*) more concretely as a device consisting of two "C"-shaped effectors. (This remained conceived as a *single* entity throughout the protocol.) B partially realized the top-level task with a hopper then accounted for the remaining aspects of the task with subtasks. He introduced a spindle to do the winding and detaching, but this introduced the need (*Sustains*) for a new task (not shown in Figure 6.5) of rotating the spindle.

If we restrict the SID to entities and tasks and the evolutionary relations among them, the result is nearly a tree or a forest (a set of trees), where the children of a node are those items derived directly from it by one of the evolutionary relations. Because of "function sharing" (where an entity realizes more than one task—see 10 in Figure 6.4 and 11 in Figure 6.5), however, the skeletal diagram may fail to be a tree or even a forest. Still, the set of nodes and the set of arcs in question is formally a graph and it is never possible to find a directed path consisting of evolutionary arcs that cycles back on itself. That is, the skeletal diagram is formally a DAG (directed acyclic graph)—we call it the *task* DAG. This implies that, if two nodes in the task DAG are on a common directed path, then one unambiguously occurs earlier in the path than the other. This imposes a certain (partial) order on the nodes of the DAG, which we envision as ordered top (for items appearing earlier in the protocol) to bottom. The task DAG is similar to what Pahl and Beitz (1984) term a function structure. They see elaboration of a function structure as an initial phase of conceptual design. Our subjects, in contrast, intermingle elaboration of the task DAG with other aspects of design.

Relations and properties are generally inherited along directed paths in the task DAG. The most obvious items not inherited are aggregate properties. This inheritance takes two forms. Relations and properties are inherited in a weak sense when several descendants taken together fulfill the role of an ancestor. For example, if a task is a prerequisite for another, then the subtasks of the first are collectively prerequisite for the subtasks, taken collectively, of the second. A relation or property is inherited in a strong sense when, if it holds of an ancestor, it holds of each descendent on its own. For example, if a task precedes another task, then all subtasks of the first precede the second and all subtasks of the second follow the first. Certain relations and properties (such as temporal relations) that may hold of tasks are not applicable to entities, and some that may hold of entities do not apply to tasks. Thus inheritance may skip items in the task DAG. Still, temporal relations among tasks, for example, impose constraints on the entities realizing those tasks.

The task DAG shows how the problem is decomposed into subproblems. In the simplest case, a subproblem consists of a task, an entity realizing it, the information listed under both in the tabular form of the diagram, and any tasks or entities that evolve from the task and entity in question (that is, any of the subproblem's subproblems). The SID tends to cluster into subproblems; this allows the graphical form of the diagram to be modularized so that the top level of each subproblem may be represented separately. A relation among items both in and outside a cluster represents, from the point of view of that cluster, an externally imposed constraint; such constraints make the diagram an *almost*-hierarchical constraint network in the sense of Sussman and Steele (1980). Inheritance can be exploited to give succinct tabular and graphical representations of subproblems and to avoid redundancy across subproblems.

3.2. Alternatives and Evaluation

Thus far all nodes in the SID have been considered AND nodes. For example, realizing a task requires realizing all its subtasks; the node representing the task is an AND node since it relates to the conjunction of the nodes that are reached from it along single evolutionary arcs. When, however, there are $n > 1$ alternative realizations of an item, we make n copies of the item, each connected to an arc from the original item. Each arc from the original to a copy represents an alternative path in the elaboration of the problem state, and the original node is an OR node since it relates to the disjunction of the nodes reached from it along each such arc. In the tabular form of the SID, each of the n alternatives is assigned a unique number J, $1 \leq J \leq n$, and is listed separately and numbered $I.\ J$, where I is the number assigned to the original; each alternative is annotated with a summary of the alternatives to it.

Each alternative is the starting point for a separate elaboration of the problem state. These elaborations may involve large parts of the structure beyond the alternatives themselves. Any arcs to or from the original node are present for each copy, although inheritance can be invoked to avoid explicit repetition. Even inheritance, however, cannot avoid detail that arises because relations with other parts of the problem are elaborated differently. Furthermore, function sharing in an alternative may cause paths from other parts of the problem to converge on that alternative. The problem state can become complex when there are alternatives for several nodes concurrently, for then in principle each combination of alternatives is a possible starting point for further elaboration. In fact, however, experienced designers are adept at avoiding this combinatorial complexity. There are practical measures to cope with such complexity in the graphical representation of the SID. A picture of the task DAG that suppresses most detail is a useful summary of the overall problem state. Separate sheets of paper can be used for the graphical representation of alternatives and repeated detail can be accommodated by drawing common structure once and photocopying.

Often an alternative is not articulated until another alternative has been worked out in detail. There is apparently a preference for evaluating alternatives against each other rather than on their own merit. The SID thus contains dead parts, representing failed alternatives, dormant parts, corresponding to unexplored alternatives, and growing parts, corresponding to the options currently considered.

Evaluations are encoded as separate items. Such an encoding lists the alternatives being evaluated, their rankings according to any criteria that are mustered, records of which alternatives met whatever conditions are raised, and any facts cited. The encoding concludes with the decision to reject or to accept (possibly tentatively) various alternatives. An evaluation is represented graphically by enclosing its scope within a dashed oval; "accept" and "reject" are written next to the appropriate items.

A typical protocol revealed much less alternative development and explicit evaluation than was anticipated. What generally occurs, however, is a *review* at the end of the design activity to ensure that nothing has been overlooked and that the proposed solution is consistent. Such reviews usually also happen at least once earlier in the protocol; they guard against wasting effort developing results that are already vitiated by a careless mistake. A review may reveal deeper problems with the current (partial) design and occasion further elaboration. For example, the design may violate some condition; if the condition has not already been articulated, noting its violation is an occasion for doing so. The design may also fare poorly according to some criterion, which again may not be articulated until applied. In either case, further facts may be invoked to support the judgment, and, in general, more information is articulated and the diagram is accordingly modified. We do not introduce new representational resources to capture these reviews. We record when a review happens and any features it exhibits in common with evaluations: criteria and how well they are met and any facts raised. Also, if an error is uncovered, the diagram is changed to reflect the corrections the designer makes, but the old version is retained so that the steps taken may be recovered.

The SID brings together all views the designer had during the design, but there is no guarantee that the designer actively retains all the information represented therein or that the designer has at any one time a global comprehension of the current problem state. Our encoding does not represent such limitations on the designer's cognitive facilities. A good designer is aware of his/her limitations and engages in review as frequently as needed to be reasonably assured of consistency and comprehensiveness.

Alternative accounts can be given of the modifications imposed when the current design is found unacceptable. One popular account views design as driven forward by propose–evaluate–accept/reject cycles. Capturing such cycles in our representation, however, would burden it with unjustifiable detail. For one thing, the individual propose–evaluate–accept/reject cycles and the relation of an *accept* or *reject* to a subsequent *propose* would impose many more relations of temporal order than in our scheme. Also, information articulated at any one of the three steps in a cycle could not be assumed to be of the same nature as the information articulated at the other two steps. Finally, since the decomposition of problems into subproblems can carry on arbitrarily far, propose–evaluate–accept/reject cycles can be arbitrarily deeply nested within other such cycles. In fact, given a proposal, the corresponding evaluation or acceptance/rejection could be indefinitely delayed by intervening proposals.

In contrast, our account of the modifications imposed when the current design is found unacceptable is simply as further elaboration of the SID. Alternatives are listed together even when formulated at very different times, so a modification is seen as rejecting one alternative in favor of another. In general, the SID (in contrast to the nets discussed in Section 5) only roughly captures the sequence in which steps are taken. The task DAG captures only a partial order on the tasks and entities articulated in the protocol. If two items are not on a common path, then they are incomparable: we cannot recover from the task DAG which was articulated before the other.

3.3. The Changing Nature of the SID

The SID changes character as it evolves. Entities and tasks at the top are concrete but typically only roughly described and are embedded in the context of the design problem. Early in the design process, entities and tasks are introduced that are more abstract and, in particular, are abstracted from the specific application context. Designers differ on how abstract they make the problem. Identification of the (largely spatio-temporal) properties relevant to mechanical design and especially relations (which are more important than properties in the abstract view) is a major part of abstracting from the context. As the design progresses, the number of tasks and entities increases, increasing the opportunities for relations and the general amount of detail.

Accidents (both properties and relations), which are all-or-nothing, tend to give way to qualities, which may assume various values. Some values are simply *topological*, expressing relations (such as *above* and *below*) among items without specifying distances, time intervals, and so on. Other values are dimensional or *metric*, and express such notions. Apart from quantitative information given in the specification, metric notions initially tend to be expressed qualitatively ("close", "far", "soon after") yet with an intended foundation that supports the comparisons ($\leq$) and operations($+$, $\times$) valid for the real number system. Increasing detail restricts the range of the various dimensions relative to the dimensions in the specifications. Causal relations give way to mechanical relations, and temporal relations become consequences of device structure. Our protocols conclude with all tasks realized by entities, and all mechanically relevant relations identified, although their values often remain fuzzy or even qualitative.

4. Ontological Analysis and Domain Equations

We now show how more formal rigor can be achieved with the representational techniques we have presented. Such rigor is necessary to explicate the meanings of terms and valid patterns of their use. Defining the method in this way allows it to be applied consistently by different researchers. The following owes a great deal to Alexander, Freiling et al. [see Freiling et al. (1985), Alexander et al. (1987), and Freiling (1988)]. Their *ontological analysis*

is a technique for the preliminary analysis of a problem solving domain. An ontology is a collection of abstract and concrete objects, relationships and transformations that represent the physical and cognitive entities necessary for accomplishing a task. (Alexander et al., 1987)

In this section we introduce ontologies, which are type systems available for declaring items that are instantiated in SIDs. While a SID is part of an encoding of a specific protocol, the ontologies are type systems for all SIDs hence for encoding any protocol. When a term is introduced for the first time in a SID (for example, it might be the name of a property taken from a protocol), it must be declared in a *domain function equation* in the static ontology. New domains (types) can be defined by *domain equations*, which impose explicit (and possibly extensive) constraints on how the declared terms can be instantiated together in a SID. The basic notions of ontological analysis are presented in 4.1. We use two of the three ontologies introduced by Alexander et al. The static ontology (presented in 4.2) gives the types of the items discussed in Section 2, that is, items that are instantiated in a SID insofar as part of the SID presents a snapshot of the problem state. The dynamic ontology (presented in 4.3) gives the types of the relations representing in a SID the evolution of the problem state, as discussed in Section 3.

4.1. *Basic Notions*

Ontological analysis is based on the domain equations of denotational semantics [see Gordon (1979) and Stoy (1977)]. For our purposes, domains may be considered sets. There are two basic statement types: *domain equations* (for example, SITE = BUILDING × CAMPUS) define domains or types, and *domain function equations* (for example, csci_building: SITE) declare the domains to which elements belong. The right-hand side of a statement is composed of one or more domains (whose names contain upper case letters) or constant elements (whose names are those used in the SID) with operators relating them. Some domains (such as BOOLEAN and REAL) are given as primitives. There are five kinds of operators:

- the *discriminated union* (generalization) of domains D and E, written D + E, defines the domain composed of each member of D and E, with original domain identity preserved;
- the *cartesian product* (aggregation) of D and E, D × E, defines the domain composed of all ordered pairs whose first element is a member of D and second element is a member of E;
- the *mapping* of D onto E, [D → E], defines the domain of all functions mapping D onto E;
- the *power set* of D, D**2, defines the domain consisting of all subsets of D; and
- the *collection of ordered subsets* of D, D*, defines the domain of all sequences of zero or more elements of D.

Ontological analysis classifies statements into one of three groups (or "ontologies") according to their function. The *static ontology* defines the primitive objects of the problem space. The *dynamic ontology* defines the actions that transform the problem from one state to another. Finally, the *epistemic* ontology defines the control knowledge applied to the static and dynamic ontologies. In our terms, the distinguished relations, such as *Realizes*, that govern the evolution of the ontology network relate to the dynamic ontology. All other items of the SID relate to the static ontology. In section 5, we introduce design nets that do the job of the epistemic ontology but are more concrete. We preempt the epistemic ontology because, we believe, control knowledge is not adequately treated in the manner of a type system. Rather, control knowledge essentially relates to sequences of events and to how one set of events establishes a condition for a subsequent event.

While a SID is specific to a protocol, the static and dynamic ontologies are general and may be thought of as supplying a type system (in the computer science sense) for the items that are recorded in any SID. Distinguished properties and relations are declared (via domain function equations) in advance and may be used in the SID encoding any protocol. Items that are introduced as a protocol is analyzed (and a SID is constructed) must be declared. (One can annotate the tabular form of the SID with type information for each item. Alternatively, one can maintain in parallel a separate dictionary with this information.) Requiring that all items be thus declared is the most fundamental step in promoting the logical coherence of the encoding of a protocol. Experience with protocol encoding indicates that logical coherence, while absolutely vital, is difficult to achieve without explicit formalisms. Also, enforcing a type scheme on the items of an encoding greatly facilitates the move to executable code for a KBS.

4.2. The Static Ontology

Figure 6.6 shows a fragment of the static ontology simplified to avoid the notion of time. The standard primitive domains NAT_NUMBER (natural number) and REAL are used. The domains ENTITY and TASK are defined as domains of atomic objects, and SORTAL and MATERIAL are defined by enumeration. The definitions of INF_DISTANCE, INF_REGION, and TOPOLOGICAL_REL have not been shown because they are complex and involve informal notions. INF_DISTANCE (respectively, INF_REGION) is the domain of informally specified distances (respectively, regions); TOPOLOGICAL_REL can be thought of as the domain of spatial vectors specified only by topological relations (such as *above*). A quality property (for example, *length*) is represented as a mapping from the domain of things of which it holds to the domain of its possible values. A quality relation is similar. Note that [X → [Y → Z]] is the type of a two-argument function whose first argument is in X, second argument is in Y, and value is in Z. An accident property, which either holds or does not hold of an element, may be thought of as a Boolean-valued function, of type [X → BOOLEAN]; a similar comment applies to accident relations (such as *Prerequisite*). Note that the value of *collection* may be false, allowing for entities that are not collections. Also, the type of collection has been defined in terms of COLLEC-

```
ENTITY = ⟨atomic object⟩
TASK = ⟨atomic object⟩
RR = REAL + INF_DISTANCE
length: [ENTITY → RR]
SORTAL = {tube,...}
MATERIAL = {plastic,...}
COLLECTION_RECORD =
        NAT_NUMBER × SORTAL ×
        BOOLEAN × MATERIAL × RR × RR
collection: [ENTITY → (COLLECTION_RECORD +
   {false})]
3D = RR × RR × RR
LL = 3D + TOPOLOGICAL_REL
dimension: [ENTITY → LL]
Operand: [TASK → [ENTITY* → BOOLEAN]]
Prerequisite: [TASK → [TASK → BOOLEAN]]
Prerequisite: [TASK → [TASK → BOOLEAN]]
REGION = LL + INF_REGION
WR: [TASK → REGION]
R: [ENTITY → REGION]
∩: [REGION → [REGION → REGION]]
```

FIGURE 6.6. Fragment of the static ontology when time is omitted.

TION_RECORD, which specifies sextuples that obviously address the current case of a bundle of tubes. In other contexts, we might want a relation type with fewer, more, or even different argument types. Thus, for the general case, we want a *polyadic* relation, and the right-hand side of the domain equation for COLLECTION_RECORD would list all the alternatives, separated by '+'s; such prolixity is the wages of accuracy.

Figure 6.7 shows how time is handled. The domain equations state that a time point is either a primitive time point or a relative time point. A primitive time point is an event in the time profile of a task, and the duration of this event is not considered significant for the problem at hand. A relative time point is a primitive time point with an offset, which is a (temporal) distance. The definition of time interval is similar. A time is either a time point of a time interval. The last line in Figure 6.7 shows the declaration for the property *length* when values are allowed to change. It indicates that an entity at a time (point or interval) has a length. The definition of the entire time profile of a task has been omitted since it requires more sophisticated notions from denotational semantics.

In Figures 6.6 and 6.7, domains that are generally thought of as consisting of real numbers are augmented to allow for values denoted by informal descriptions. This is necessary to allow the values of properties and relations to be refined as the design progresses.

```
POINT_EVENT = ⟨atomic event⟩
INTERVAL_EVENT = ⟨atomic event⟩
PRIMITIVE_TIME_POINT = TASK × POINT_EVENT
TIME_OFFSET = RR
REL_TIME_POINT = PRIMITIVE_TIME_POINT × TIME_OFFSET
TIME_POINT = PRIMITIVE_TIME_POINT × REL_TIME_POINT
PRIMITIVE_TIME_INTERVAL = TASK × INTERVAL_EVENT
DERIVED_TIME_INTERVAL = TIME_POINT × TIME_POINT
REL_TIME_INTERVAL =
    (PRIMITIVE_TIME_INTERVAL + DERIVED_TIME_INTERVAL) ×
    TIME_OFFSET
TIME_INTERVAL =
        PRIMITIVE_TIME_INTERVAL + DERIVED_TIME_INTERVAL +
        REL_TIME_INTERVAL
TIME = TIME_POINT + TIME_INTERVAL

length: [ENTITY → [TIME → RR]]
```

FIGURE 6.7. Fragment for the part of the static ontology defining temporal domains. The declaration of *length* indicates how qualities are declared when their values are allowed to change.

4.3. *The Dynamic Ontology*

The dynamic ontology, as presented in this paper, is quite simple:

Realizes: [ENTITY → [TASK → BOOLEAN]]
RealizesT: [TASK → [TASK → BOOLEAN]]
RealizesE: [ENTITY → [ENTITY → BOOLEAN]]
Subtask: [TASK → [TASK → BOOLEAN]]
Sustains: [TASK → [ENTITY → BOOLEAN]]

5. Design Nets: Models of Control Knowledge

To do the job of the epistemic ontology of Alexander et al., which defines control knowledge, we introduce a set of three *design nets*. In this section, we first (in 5.1) introduce the basic notions and the three design nets we have found necessary for encoding the control patterns in our protocols. In 5.2 we describe the *trace* encoded for a protocol, which (among other things) encodes sequences of protocol episodes as sequences of net transition events. Since a problem decomposes into smaller problems and we maintain that design nets are general structures that govern the evolution of problem states, we must allow that there are different, concurrent *activations* of the nets for different subproblems that are still evolving. In 5.3 we explain the notion of a net activation. Having outlined what we require of design nets, we address in 5.4 how the nets can be formalized. The standard automaton used in computer science for modeling concurrent sequences of events is a Petri net. To capture the salient aspects that we find in control patterns that occur in conceptual mechanical design, we add certain enhancements to Petri nets. Later, in Section 8, after characterizing strategies, we address the advantages for our purposes of Petri nets over other computer-science formalisms that capture the temporal order of events. A Petri net has "places" that are connected by "transitions." We call a place a "focus," a term more appropriate for our application since we relate a focus to a coherent aspect of the problem considered at a particular step in a design process. Each focus has as its name a phrase describing the aspect to which it relates. The major enhancement we make, described in 5.5, is to associate a condition with each transition. A condition states something about the SID and a transition can "fire" only if its associated condition is true. Transition conditions are seen as Boolean combinations of atomic conditions. Thus, certain phrases—focus names and atomic conditions—are integral parts of a design net, and we must ensure that their functions—denoting problem aspects and stating conditions—are formally defined. A phrase must be constructed from a vocabulary with fixed meanings and so that the meaning of the phrase is composed in a well-defined way from the meanings of its constituents. Section 5.6 presents "semantic grammars" that generate these phrases. The control patterns in the traces for conceptual mechanical design protocols are often quite indeterminate, so the design nets have a difficult role. In Section 8, we relate more abstract control patterns—strategies—to design nets.

5.1. Basic Notions

Like the static and dynamic ontologies, the design nets are general structures that govern the construction of any SID. They are an attempt to identify control that is common across diverse designers and mechanical design problems while allowing for differences. Unlike the static and dynamic ontologies (and unlike the epistemic ontology of Alexander et al.), the design nets represent sequences of events and how events establish conditions for subsequent events. An event in this context is a transition from one design step or state to the next, as indicated by an episode in a design protocol. Each net (or, more accurately, net activation—see below) can be in one of several states; the state of a given design activity is captured by the vector of states in the simulation of that activity by the various nets. The simulation of a design activity is a sequence of such vectors of states, and the events of interest are transitions from one state vector to another. Alternatively, we can think of the fundamental kind of event as a state transition in one net; a transition from one state vector to another is determined by one or more simultaneous net-specific events. Furthermore, we can think of the simulation of a design activity as fundamentally a sequence of (possibly simultaneous) net-specific transitions (events).

We recognize three design nets. The *task net* relates to the task DAG and simulates design steps that decompose tasks into subtasks, find dependencies among tasks, and decide which tasks should be realized together. The *behavior/structure net* simulates design steps that articulate items represented in the SID that become manifested in the artifact, such as locations of inputs and outputs. Finally, the *environment net* simulates design steps that flesh out the problem setting by considering such things as the operands of a task and obstacles that exist in the environment. Nets interpret the SID. For example, the behavior/structure net might identify certain changes of location of entities as output motion for the artifact being designed. Again, the environment net might identify a certain entity as an obstacle.

5.2. The Trace Encoded for a Protocol

An encoding of a design protocol using our methodology results in two documents: a SID and a trace of the net transitions. The trace is a collective document, with a separate document for each net. It records each transition, with (among other things) references to the episode(s) in the protocol that it simulates, the (pre)conditions in the SID that (along with the state of the net) enabled it, and the postconditions it established in the SID. The trace is specific to a protocol, but the nets (like the static and dynamic ontologies) are general. The SID is typically more complex than the bare sequence of transitions recorded in the trace since nothing in the structure of the nets represents items (entities, tasks, properties, or relations) in the design state. As a SID is constructed during the analysis of a design protocol, the dynamic and especially the static ontology are updated to declare new items and distinguished properties and relations that are needed to represent the evolving design state. Similarly, as the trace is constructed, the nets are updated to allow for steps, transitions, and conditions that are required by the encoding of the protocol but that have not previously been registered in the nets. The updates to the nets made while encoding a given protocol are typically more involved than the updates to the ontologies. The latter are usually declarations of items introduced specifically for the protocol and are largely independent of each other—the interactions of items are recorded in the SID. Updates to the nets, in contrast, allow steps and transitions that interact with those already allowed, and often a significant proportion of the steps and transitions recorded in the trace require new steps or transitions in the nets.

Recall that the SID has a meager representation of the order in which items are added. The only ordering is supplied by the evolutionary arcs, which impose a partial order on the items in the task DAG. The episodes into which the protocol is partitioned are totally ordered since human speech is necessarily serial. The trace allows for concurrency of events since the nets support such concurrency—so an episode could correspond to several design-step transitions. The nets, furthermore, show how the sequence of events recorded in the trace could be reordered without affecting the design activity recorded in the protocol. The nets thus "make sense" of the trace and allow arbitrary sequencing and fluctuations to be abstracted away. Thus, notions such as strategy and initiating, pursuing, or abandoning a line of reasoning are captured by the design nets and the sequence of design events they simulate.

5.3. Net Activations

To explain how the nets interact, it is helpful first to review the fundamental notions of concurrent processes as supported by parallel or distributed computing systems. A *process* is a running program, or an *activation* of a program, where a program may be considered a block of code. Any number of processes may correspond to a given block of code since the code is unchanged by being executed and each process has a distinct *state*, which changes as the process proceeds. A process state includes the values of program variables, the status of the I/O streams, and the address of the next instruction (in the code block) to execute. Concurrency exists when several processes, which may share code blocks, proceed concurrently. The I/O streams accessed by a process may allow communication with processes concurrent with that process. Concurrent processes may also communicate via shared memory, which allows one process to change the values of some of the variables in certain other processes; thus, one process may change the state of another. Processes can be recursive in the sense that one process, an activation of a given code block, may initiate an activation of the same code block to handle a subproblem of the problem the parent process is handling. A process can recursively spawn any number of new processes, and a recursively spawned process can itself recursively spawn additional processes.

The design nets correspond to code blocks. Corresponding to a process—an activation of a code block—is what we call a *net activation*. The sequence of instructions (with variable values) executed by a process is referred to as an execution trace. This corresponds to the trace that we encode from a protocol except that such a trace generally records transitions from (possibly several) activations of the three nets; more accurately, our trace corresponds to the merged traces of several concurrent processes. We allow a given state of a net activation to include more than one currently active step; in this respect, a state of a net activation is more complex than a state of a process, which has a unique next instruction. Similarly, more than one transition in a given net activation between sets of steps may be recorded in the trace for a single protocol episode. We do not admit message passing (via I/O streams) within or among net activations, but we view the SID as a global data structure accessible to all net activations—as if it were in memory shared by all these activations. The SID, however, is not held to contribute to the state of a net activation, which relates only to the currently active steps. Yet we have (see below) the transitions between sets of steps depend on conditions in the SID and active steps are responsible for changes to the SID.

Conceptual design is typically recursive in the sense that a problem is decomposed into subproblems, which themselves may be further decomposed into smaller subproblems. A major role of the task net is to decompose problems (or tasks to be realized) into subproblems (or subtasks). Each subproblem corresponds to activations of the three design nets. The activation of the environment net for a subproblem handles the interface between that subproblem and the larger problem of which it is a part. Different parts of the SID are elaborated in response to different subproblems, and the recursive decomposition of the entire design problem is given by the structure of the task DAG. The trace produced as part of the encoding of a protocol associates each net transition with a subproblem, in effect identifying the net activation in which the transition takes place. Subproblems generally relate specifically to a topmost task or, occasionally, entity. The identifier for the task or entity is used to identify the corresponding subproblem in the trace. For brevity, we shall often in the sequel refer to nets when more accurately we should refer to net activations. What is intended will be clear from the context.

5.4. Design Nets as Modified Petri Nets

Petri nets (Peterson, 1981) are used to model sequences of events in systems in which events establish conditions for subsequent events and concurrency is allowed. We thus formulate our design nets as modified Petri nets. A Petri net consists of *places* (represented by circles in Figure 6.8), *transitions* (represented by bars), and *arcs*. An arc is either from a place to a transition or from a transition to a place.

A place P is an *input place* of a transition T if there is an arc from P to T; P is an output place of T if there is an arc from T to P. We assume that there is at most one arc from a given place to a given transition and at most one

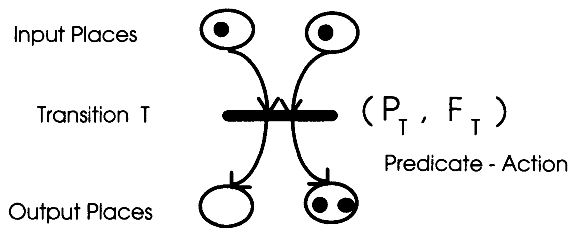


FIGURE 6.8. A Petri net transition along with its input and output places. All places shown except the left output place are marked, that is have at least one token. The right output place has two tokens. This transition, T, is part of a predicate—action system since it is labeled with a predicate—function pair, (P_T, F_T). Ignoring the predicate, the transition is enabled since all inputs are marked.

arc from a given transition to a given place. (We thus restrict ourselves to *ordinary* Petri nets; relaxing these conditions on the arcs allows *general* Petri nets.) A *marking* of a Petri pet is an assignment of zero or more tokens (represented by filled circles in Figure 6.8) to each place. A transition may *fire* when it is *enabled*; it is enabled if each input place has at least one token. A transition fires by removing a token from each input place and depositing a token in each output place. Each place may be an input place for zero or more transitions and an output place for zero or more transitions. A place may even be both an input and an output place for a given transition.

Events are modeled by Petri net transitions. The fact that a condition holds is modeled by the presence of a token in the place corresponding to that condition. The input places of a transition then represent the preconditions of the event in question and its output places represent its postconditions. A generally accepted limitation is that events are modeled as instantaneous and nonsimultaneous; such events are *primitive*. Petri net models are inherently parallel since two enabled transitions that do not share an input place may fire independently. They are also inherently nondeterministic since any one of several concurrently enabled transitions may be the next to fire. Also, the firing of one enabled transition that shares an input place with another enabled transition may disable the latter.

A Petri net place is what we have called a "design step". We now refer to it as a *focus* to emphasize that it identifies features the designer is considering at a certain point. The firing of a transition corresponds to an event recorded in the trace encoded for a protocol and thus relates to an episode in the protocol. In fact, what is encoded for an episode covers a transition *plus* the activity of the set of newly active foci. The state of a net (or, more exactly, of an activation of a net) is simply a marking of the net. The parallelism inherent in Petri nets supports the intra-net concurrency we have noted. Finally, Petri net nondeterminism allows the nets to indicate how the sequence of events recorded in a trace could be reordered without affecting the design activity. Since several transitions could correspond to a single protocol episode, the nonsimultaneity requirement must be dropped; this does not violate the spirit of Petri nets since we view the synchronization of transitions as something added onto the basic net model.

A focus, then, is a coherent aspect of the problem. When a focus contains a token, the corresponding problem aspect is under consideration and the focus is considered to be active in the sense that it can elaborate part of the SID. How a focus elaborates the SID is not encoded from the protocol and need not be addressed, yet the trace records which parts of the SID are elaborated by which foci. We associate each focus with a descriptive phrase, taken to be its name, that identifies the aspect of the problem it handles. These phrases are generated by semantic grammars (Freiling et al., 1985) (as discussed in Section 5.6), one for each net. The phrases that correspond to foci that have been noted in our protocols form a proper subset of the total set of possible phrases; some corresponding foci are unlikely ever to appear.

5.5. Conditions

A *predicate-action* system (Keller, 1976) is a modified type of Petri net in which each transition T has a label of the form (P_T, F_T)—see Figure 6.8—where P_T is a predicate and F_T is a partial function defined only when P_T is true. In such a system, for a transition to be enabled, not only must the input places contain tokens but also P_T must be true. The predicate, P_T, thus adds another condition to the event represented by the transition. When the transition fires, the function (action) F_T is executed. When a predicate-action system is used to model execution of a program, program variables are used as arguments of P_T and F_T.

We label each transition in a design net with a predicate, which we call a *condition*. Instead of program variables, a condition contains referring expressions that denote items in the SID of the appropriate type (as declared in the static or dynamic ontology). The conditions on transitions thus help coordinate transitions in various design net activities and restrict the possible problem states. We associate actions with foci, not with transitions, since an aspect of the problem is associated with a focus. Although transitions are instantaneous, the residence of a token in a focus—the period during which the focus can contribute to the elaboration of the SID—is limited only by the occurrence of a transition that requires that token as input. We exploit this indeterminacy by allowing the various active foci to collaborate in elaborating the SID. This collaboration generally requires several sequentially coordinated actions hence an interval of non-zero duration.

Conditions, like foci, interpret the ontology network. The evidence for the condition on a transition is tied up with the evidence for the input foci for that transition since the marking of the input foci, as well as what we term the condition, make up the condition in the broader sense of what enables the transition. Three kinds of conditions are recognized: *epistemic*, *requirement*, and *factual* conditions. Conditions can also be simply *enabling* or *transformation-requiring*. A condition is enabling insofar as, when it is met, the transition may fire if the input states are marked. A condition is also transformation-requiring if, when it is met and the transition fires, it should be changed by one of the foci activated by the firing. In some sense, transformation-requiring conditions are the reasons (goals) for transitions firing. Epistemic conditions relate to how well formulated the problem is, that is, what is known about the problem state. They may state that certain aspects of the problem are known, are only partially known, or are not known. When the aspect is partially known or not known and the condition is transformation-requiring, the ensuing transition allows a constrained choice that results in the aspect becoming better known. Requirement conditions relate to requirements that remain to be met. They are always transformation-requiring and they do not occur in the environment net and occur sparingly in the task net, where they relate to the operands. Factual conditions are statements of fact; in their strongest form, they state restrictions on the solution domain. Factual conditions are never transformation-requiring.

In the task net, the facts generally relate to the time profiles, operands, and temporal relations of tasks. In the behavior/structure net, they generally relate to the task, device, or operand. Finally, in the environment net, factual conditions state what aspects of the operand or environment affect the task or device or how the task or device requires or affects the outside world. The conditions on a transition as revealed in an episode generally include a transformation-requiring condition and one or more conditions that are simply enabling.

The conditions discussed in the last paragraph are (ignoring for the moment occurrences of the connective "not") logically atomic, that is, they cannot be analyzed into expressions containing logical connectives. As with the names of foci, atomic conditions are generated by semantic grammars, one for each net. The several atomic conditions that are recorded for a transition firing revealed in a protocol episode form a single conjunctive condition: for the transition to fire, all must be true. Certain singular referring expressions, especially definite descriptions (such as "the operand" and "the task"), generally occur in several of the atomic conditions thus conjoined, where they consistently refer to the same items. After firings of the same transition have been identified in several episodes (possibly in different protocols), one generally has a set of such conjunctive conditions, at least one of which holds for each firing. We thus form the comprehensive condition associated with the transition in question by forming the disjunction of the conjunctive conditions in the set. This gives a condition in a much-used normal form, *disjunctive normal form*; that is, it has the form

$$(c_{11} \wedge c_{12} \wedge \cdots \wedge c_{1n_1}) \vee (c_{21} \wedge c_{22} \wedge \cdots \wedge c_{2n_2}) \vee \cdots$$
$$\vee\ (c_{m1} \wedge c_{m2} \wedge \cdots \wedge c_{mn_m})$$

where $\wedge$ is the logical symbol for *and*, $\vee$ is the symbol for *or*, $n_j \geq 1$ for all j, $1 \leq j \leq m$, each c_{ij}, $1 \leq i \leq m$, $1 \leq j \leq n_i$, is an atomic condition, and each $(c_{i1} \wedge c_{i2} \wedge \cdots \wedge c_{in_i})$, $1 \leq i \leq m$, is encoded for a single episode. Note that no *not* connective occurs in this expression. There are several ways to simplify such an expression, most obviously, if one disjunct $(c_{i1} \wedge c_{i2} \wedge \cdots \wedge c_{in_i})$ contains all the conjuncts c_{ij} contained in another disjunct, then the former can be dropped (since, if it is true, so is the latter). Since it is difficult to specify in advance whether a grammatically negative but otherwise atomic condition is tested by searching for the presence or absence of certain features in the SID, we do not explicitly represent *not* as a logical connective. We must therefore guarantee that no conjunctive condition contains an atomic condition and its negation.

5.6. *Semantic Grammars for Focus Names and Atomic Conditions*

To ensure that the phrases constituting focus names and atomic conditions are constructed in well-defined ways from a vocabulary with fixed meanings, we generate these phrases with certain semantic grammars (Freiling et al., 1985). A semantic grammar is a kind of *BNF* (Backus–Naur form) *grammar*, so its rules (or productions) are of the form LHS ::= RHS, where LHS and RHS are (respectively) the left and right hand sides of the rule and ::= is read "is replaced by." (BNF grammars, with minor notational variations, are also known as *context-free grammars*.) Symbols in the grammar are either *nonterminals* (or *variables*), written enclosed in angled brackets, or *terminals*, written without brackets. One nonterminal is designated as the *start symbol*, which is the initial target string. The LHS of a rule is always a single nonterminal. We are interested in grammars in which, for each nonterminal, there is exactly one rule with it as the LHS. The RHS is of the form $S_1 | S_2 | \cdots | S_n$, where $n \geq 1$, each S_i is a string of one or more nonterminals and terminals, and the vertical bar is read "or." (A rule of the form $N ::= S_1 | S_2 | \cdots | S_n$ is equivalent to a set of rules $N ::= S_1$, $N ::= S_2, \ldots, N ::= S_n$, so our requirement that, for each nonterminal, there be *exactly one* rule with it as the LHS is—except for convenience—equivalent to the requirement that, for each non-terminal, there be *at least one* rule with it as the LHS.) A nonterminal N in the target string may be replaced by any S_i in the RHS of the rule with N as the LHS. The target string is rewritten in this way until it contains no nonterminals. Since there are generally choices in how to rewrite a given nonterminal in the target string, there are generally many strings of terminals derivable from the start symbol; the set of all such strings is the language generated by the grammar.

A nonterminal is also called a *category*. Strings of terminals that descend from a category are said to belong to that category. An alternative S_i in a rule may be the special symbol ε, representing the empty string. If ε is chosen when the rule is applied, then nothing remains of the category forming the LHS of the rule in the string of terminals that eventually results.

Space restrictions allow only fragments (in the sense that neither are all rules shown nor are all alternatives within a rule necessarily shown) of the semantic grammars relating to the behavior/structure net to be given. Figure 6.9 gives a fragment of the semantic grammar for the names of foci in the behavior/structure net. Figure 6.10 presents a stylized representation of a device and identifies the conceptual parts that are referred to in the focus names. The term *problem/mechanism* refers to the entire device and its operation. The term *quality* refers to a quality property. *Attribute* also refers to a property, but one that is essential to the conceptual part of which it is a property. (For example, *frequency* in the phrase *frequency transduction* is an attribute; we cannot identify transduction without thereby identifying what is transduced.) Some categories such as ⟨*operand designator*⟩, could be eliminated in that they can be replaced by only one terminal or sequence of terminals. Such categories were introduced both to convey meaning to the terminals and to allow for alternatives that might arise in the future. To illustrate how one possible focus may be a restricted version of another possible focus, note that, for example, *frequency* is only one aspect of *motion* as an instance of the category ⟨*I/O quality*⟩. Thus, a focus with name *input*

```
⟨BS focus⟩ ::= problem/mechanism ⟨p/m quality⟩
               | ⟨I/O⟩ ⟨I/O quality⟩
               | ⟨I/O⟩ ⟨I/O attribute⟩ transduction
               | ⟨relational modifier⟩ relation of ⟨components⟩
               | mechanical connection of ⟨components⟩
               | spatial relation of ⟨components⟩ to ground
               | ⟨BS entity⟩ ⟨BS entity quality⟩
               | ⟨component designator⟩ ⟨components⟩
               | ⟨operand designator⟩ operand
               | ⟨action⟩ operand
⟨p/m quality⟩ ::= degrees_of_freedom | dimensionality
⟨I/O⟩ ::= input | output
⟨I/O quality⟩ ::= dimensionality | forces | frequency | location | motion | orientation
⟨I/O attribute⟩ ::= frequency | orientation | translation
⟨relational modifier⟩ ::= spatial | temporal
⟨BS entity⟩ ::= mechanism | ⟨components⟩
⟨components⟩ ::= ⟨component⟩ and ⟨components⟩ | ⟨component⟩
⟨component⟩ ::= ⟨component attribute⟩ ⟨component type⟩ | ground
⟨component attribute⟩ ::= ε | ⟨I/O⟩
⟨component type⟩ ::= submechanism | element
⟨BS entity quality⟩ ::= forces | geometry | material | joint_type | location | orientation
⟨component designator⟩ ::= number of
⟨operand designator⟩ ::= number of
⟨action⟩ ::= accepting | isolating | securing
```

FIGURE 6.9. Fragment of the sematic grammar for the names of foci in the behavior/structure net.

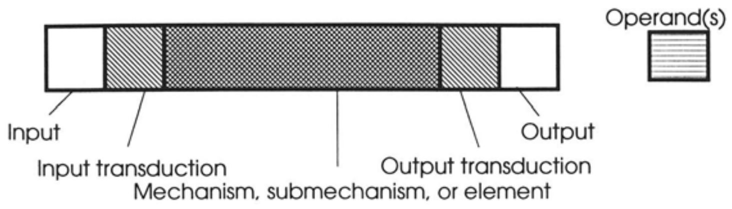


FIGURE 6.10. The conceptual parts of a device as interpreted by the behavior/structure net.

frequency would handle a more restricted aspect of the problem than a focus with name *input motion*.

Figure 6.11 presents a fragment of the semantic grammar for the atomic conditions in the behavior/structure net. To illustrate how one atomic condition can be more restrictive than another, consider two strings of terminals that are instances of the category ⟨*BS epistemic condition*⟩. Suppose that both are produced using the third alternative in the rule with LHS ⟨*epistemic object*⟩. Suppose also that the two strings result from the ε alternative for

```
⟨BS condition⟩ ::= ⟨BS epistemic condition⟩ | ⟨BS requirement condition⟩
                 | ⟨BS factual condition⟩
⟨BS epistemic condition⟩ ::= ⟨epistemic object⟩ is ⟨degree⟩ known
⟨epistemic object⟩ ::= ⟨pseudo-entity⟩ | ⟨property⟩ of ⟨pseudo-entity⟩
                   | ⟨time bounds⟩⟨property⟩ of ⟨quantifier⟩⟨entity⟩⟨state bounds⟩
⟨degree⟩ ::= ε | not | partially | . . .
⟨quantifier⟩ ::= ε | some | most | all | . . .
⟨pseudo-entity⟩ ::= output motion | feature of ⟨entity⟩
⟨time bounds⟩ ::= ε | initial and final
⟨state bounds⟩ ::= ε | from initial ⟨property⟩ to final ⟨property⟩
⟨BS factual condition⟩ ::= output component ⟨property⟩ matches operand ⟨property⟩
                       | ⟨order attribute⟩ tasks have same operand
```

FIGURE 6.11. Fragment of the semantic grammar for the atomic conditions in the behavior/structure net.

⟨*degree*⟩ and that they are identical except that the first results from the ε alternative for ⟨*state bounds*⟩ while the second results from the alternative *from initial* ⟨*property*⟩ *to final* ⟨*property*⟩. The first is more restrictive than the second since it applies to the entire interval in question. Also, note that, concerning instances of the category ⟨*quantifier*⟩ when ⟨*degree*⟩ is ε, *all* leads to more restrictive conditions than *most*, which in turn leads to more restrictive conditions than *some*.

6. The Encoding Process and Its Reliability

Having described the formalisms we use to encode protocols, we can now clearly describe how protocols are collected and encoded and how we check that our representation schemes are followed objectively. In 6.1 we discuss the procedures we use for collecting and encoding protocols and the documents produced by encoding. In 6.2 we discuss the *reliability* of our representations—how well they facilitate agreement between two people independently encoding the same protocol.

6.1. Procedures and Documents

To study limited conceptual mechanical design, we present a subject with a mechanical design problem that only roughly gives the task structure and environment conditions and that says little or nothing about the artifact. The subject completes the design to the point where all significant components and their relations are identified, but most dimensions remain to be determined exactly. This takes one to two hours for the subjects and problems we have chosen.

The designer is asked to think aloud and his words are tape-recorded. It is sometimes necessary to remind him to articulate his thoughts or to stick to the facts; this causes little disruption. The designer numbers all figures he draws and labels significant items. Drawings and the transcribed recording constitute the protocol. The transcription is segmented into sentences or (for long sentences) clauses; segment boundaries are also introduced at pauses. Segments are numbered consecutively for later reference.

We have encoded about ten protocols. A protocol is encoded separately (but concurrently) for each of the three nets, giving collectively a document we call the *trace*, which was mentioned in Section 5. For each net, the protocol is partitioned into episodes intended to reflect transitions in the net. Generally only one transition per net occurs, although occasionally transitions may overlap or even coincide. An episode usually occupies one or two transcription segments, sometimes less than one, and rarely more than five. Episode boundaries not infrequently fall within segments. Perhaps as much as 20% of a protocol is classified as noise—asides in which the designer comments on his design or on general points—and is not encoded. The following topics are recorded for each episode:

- a *reference*, using the segment numbers introduced during transcription;
- the current *subproblem*, identified by its top-level task(s) and entity(s);
- the *output foci* of the transition;
- the *input foci* of the transition;
- the *transformation-requiring* atomic *condition*;
- the *simply enabling* atomic *conditions*; and
- the *modifications* made to the SID.

Modifications are also recorded in the tabular version of the SID. Periodically, the graphical version is updated with the gross changes; this is essential for maintaining a general grasp of the protocol.

We strive for encodings that make engineering sense both in terms of how the current problem state is represented and in terms of what is achieved by each episode. Our protocol analysis teams thus always include at least one mechanical engineer, who explains the designer's activity to other members of the team. These explanations are "internal" in that they appeal only to mechanical principles. Since verbal recording is the only process intervening between the designer's heeding the (largely nonverbal) information and his verbalization of it, our protocols record the sequence of heeded information as determined by the task-directed cognitive processes involved (Ericsson and Simon, 1984). We ensure that such sequences can be recovered from our encodings, so internal explanations form a backdrop for judging the adequacy of the encoding. This backdrop becomes explicit to the extent that constraints among heeded items are essential for internally validating the steps the designer was observed to take. Against this backdrop, we may merge encodings of protocols from several designers, which requires a sizable common vocabulary; part of the internal explanation of a protocol is translating certain lexical items in the protocol into this common vocabulary.

6.2. *Reliability*

A representation is *reliable* if two people independently encoding a protocol into that representation generally agree. Reliability is an issue because there are no operational definitions leading from the words in the protocol to the constructs of our representation. Concern with reliability has had a major impact on the representation schemes presented in the chapter. In testing reliability, we compare encodings of a given protocol by two people. (If either is not a mechanical engineer, he or she is briefed about the engineering content of the protocol by a mechanical engineer on the team.) To avoid severe penalty from cumulative differences, a section of about two pages is encoded and scored, and then the differences are resolved before proceeding to the next section.

Agreement on nets and on the occurrence and extent of episodes was very good—almost 90% of the episodes agreed. There was good agreement on modifications. Without distinguished relations and properties, this level of agreement could not be achieved. Before the semantic grammars were defined, the encoders had difficulty even agreeing on the names or descriptions of foci and atomic conditions; the grammars caused dramatic improvement. Disagreement still arises largely for two reasons: it is possible to produce different combinations of foci or conditions that give much the same effect, and, secondly, it is difficult to determine whether a focus or enabling condition has a minor role or no role. We return to these problems in Section 8.

7. Software to Support Knowledge Acquisition and Representation

Any representation of a nontrivial design protocol, if it is to capture the coherence of the engineering context, must appear complex in some respects. Our formal methods, that constrain and direct protocol encoding, help us accommodate this complexity. These methods themselves, however, can burden the encoders, eventually to the point where consistency is threatened and the time to encode a protocol becomes excessive. The way out of this dilemma is to supply automated assistance. In 9.1 we first consider executable encodings and then review certain software toolkits that support knowledge acquisition. We sketch how a similar toolkit could be used for our encoding schemes. In 9.2 we consider why our scheme is superior to the KADS model, which underlies the most influential toolkit, Shelley. This suggests how a toolkit for our encoding schemes would differ in its underlying assumptions from Shelley.

7.1. *Software*

The most straightforward way to guard against gross inconsistencies is to execute the encodings. This can be viewed as a way to promote the *objectivity* of our encodings. This is promoted, from one side, by using the measures we have obtained in operationalizing our notion of reliability. From another side the objectivity of our encodings is promoted by striving for an ideal of executable (not requiring human interpretation) encodings. To advance this ideal, we have implemented (in the parallel logic programming language Parlog [18]) a prototype interpreter that executes steps in a design protocol recorded as transitions in the design nets. What is executed is actually Parlog code, but there is a rigid and clear correspondence between the Parlog code and the encodings except that actions associated with foci must go beyond what is revealed in the protocols. The conditions and modifications in the SID give input/output specifications for these actions, which are implemented in as abstract a form as possible (see Section 9). Modest forms of the SID, rule interpretation (for modifying the SID—see Section 9), and conditional transition firing are implemented. The user must assist with moves not sufficiently specified in the encoding. We reconcile this with our ideal of executable encodings by introducing (from theoretical computer science) the notion of an *oracle*. A call to an oracle is embedded in an algorithm that asks the oracle a question that is predetermined except for certain parameter values and the oracle supplies outputs, which may be complex structures.

Automated assistance can be directly at the point of encoding. Shelley (Anjewierden et al., 1992) is a software workbench (or toolkit) developed by the KADS project to support knowledge acquisition and engineering. It is analogous to a CASE tool for software engineering. Similarly, KEATS (the "Knowledge Engineering Assistant") (Motta el al., 1989) is a toolkit that provides life-cycle tools for KBSs. Embedded in KEATS-2 is the knowledge acquisition tool Acquist, a hypertext-based facility that allows the knowledge engineer to carry out knowledge acquisition by abstracting and structuring knowledge contained in raw transcript text. Acquist, Shelley, and KRITON (Diederich et al., 1987) are all knowledge acquisition systems that emphasize analysis of transcripts to extract expert knowledge. The transcripts are, in the first instance, transcripts of protocols, but they may also be transcripts of structured interviews and even transcribed portions of textbooks. All three toolkits support defining and structuring the concepts and the relations among concepts that are noted in the transcript and support linking concepts with terms in and fragments of the raw text. Shelley and Acquist support both bottom-up knowledge acquisition, where a model is constructed piecemeal as the text is analyzed, and top-down knowledge acquisition, where a library of "theories" or "interpretation models" is available to assist conceptualization. KRITON supports only a bottom-up style and aims to make knowledge acquisition fully automated by eliciting, analyzing, and representing knowledge. While KADS emphasizes documentation and semi-executable models, KEATS is an environment for building the end product and so also provides facilities at the debugging level.

It is not difficult to imagine the design of a toolkit like Shelley for our encoding schemes. Support for bottom-up knowledge acquisition would help partition the protocol text and update the nets when needed transitions are missing. It would maintain and display the trace as well as both the tabular and graphical versions of the SID, and needed updates to the static and dynamic ontologies would be flagged. Bottom-up support would also maintain links from all encoding documents to terms in and fragments of the protocol text and would maintain the consistency of all documents both internally and with reference to each other. Support for top-down knowledge acquisition would include making available previously recorded net transitions and the semantic grammars when the protocol is partitioned and the trace is produced. It could also make available fragments of previous SIDs (especially fragments of task DAGs) as possible templates for the diagram currently being constructed. The interested reader is encouraged to read the references on KADS and especially Shelley; he/she should be able to expand the sketch given here in light of the earlier sections of this chapter.

7.2. A Comparison with KADS

Any toolkit brings a certain amount of conceptual baggage, and that of the KADS methodology (which some claim is becoming a *de facto* European standard), which lies behind Shelley, is the most fully developed and influential. We briefly consider why our scheme is superior to the KADS model for conceptual mechanical design and probably for conceptual design in general. Part of this superiority is because protocol analysis is particularly important for investigating conceptual design; our scheme was developed for protocol analysis whereas the KADS model is a general model of expertise. Also, our scheme emphasizes what is critical in conceptual mechanical design; it is difficult for a general model to compete in this specialized domain. We allow the possibility that the KADS model and the Shelley toolkit may be adequate for nonprotocol sources and for design stages later than conceptual design. The following comments can be seen as a sketch for specialized additions to the KADS model and initial specifications for a specialized knowledge acquisition toolkit. KADS (Wielinga et al., 1992) imposes a four-layer model of expertise; the layers are: domain knowledge, inference knowledge, task knowledge, and strategic knowledge. $(ML)^2$ (van Harmelen and Baider, 1992) is a formal language for representing KADS models of expertise. The TheME environment supplies automated support for model construction in $(ML)^2$ by blocking modifications that would result in a mathematically ill-formed model or a model violating KADS conventions; it also allows the model to be viewed in various helpful ways. $Si(ML)^2$ is an interpreter for a subset of $(ML)^2$ that allows a model to be executed.

KADS' *domain* layer embodies the conceptualization of a domain in the form of a domain theory whose primitives are based on the primitives of KL-ONE. KL-ONE roughly does the job of our static ontology in that both supply a type system and declare items to be of certain types. As KL-ONE terms can be combined to represent statements (which are either true or false), so the items declared in our static ontology can be instantiated together, in a SID, to represent statements (of what properties an entity has, what task precedes another, and so on). Our scheme, however, explicitly includes items—such as time profiles, regions, and relations between tasks—that relate to mechanical design. Furthermore, the graphical form of a SID, since it emphasizes the relations among entities in a conceptual layout, is conducive to the use of graphical engineering representations, such as bond graphs. On its own, KADS is little use in this domain since the fundamental domain concepts have rather specific application yet form a rich and expressive system. The domain layer is represented in (ML)^2 by order-sorted logic (in which all variables and constants have types, and the types are arranged in a subsort hierarchy); modularity is achieved by partitioning the axiom set into several sub-theories, which can be combined by set-theoretical union. Thus (ML)^2 adds nothing to KADS for our domain and even encourages a level of development that obscures the fundamental concepts and their relationships. A representation of the evolving problem state in mechanical design must focus on key concepts or risk losing sight of design goals. And theories, in the logical sense, can become enormous in engineering design, because of engineering's use of mathematics and physics, with little or no contribution to our understanding of specific cases.

The remaining KADS layers can be characterized as control knowledge. *Inference* knowledge, corresponding to our dynamic ontology, captures inferences, abstracted from the domain theory. An inference is specified as a primitive (a *knowledge source*), fully defined by an input/output specification and a reference to the domain knowledge used. This keeps apart the inference and domain layers. In our scheme, in contrast, the dynamic ontology is presented in the same form as the static ontology and evolutionary relations are represented in the SID along with other relations. In conceptual design, it is natural to integrate the domain and inference layers since the problem state, and the eventual artifact design, is structured by the way the decomposition evolves. We have emphasized this with the notion of a task DAG, a skeleton of a SID showing only the evolutionary arcs connecting tasks and entities. The advantages of our scheme in this respect again relates to the fact that our scheme is specifically for conceptual design. Our scheme allows relations, properties, and values among or of items to be inherited along evolutionary arcs. In contrast, the KADS framework, following KL-ONE, allows inheritance only among items at the domain level, in effect restricting inheritance to design steps that realize an item by specializing it. (ML)^2 represents the inference layer by metalogic, in which terms are names for formulas in the domain layer. Since a KADS metaclass describes a role of domain expressions in the inference process, metaclasses are represented as naming operations, where the names of domain expressions encode the expressions' roles in the inference process. Applying this formalization to conceptual mechanical design would result in something much more complex than the simple dynamic ontology we present. It turns out that, logically, domain equations have a higher-order aspect (terms can denote sets as well as individuals) and meta-logical notions can be translated into a higher-order setting. In our context, the higher-order formulation has an elegance that cannot be approached by a metalogical formulation.

Turning to *task* knowledge, a KADS task (unlike a task in our sense, something in the problem state) is a fixed strategy for achieving a problem-solving goal, where the primitive problem-solving tasks are inferences specified at the inference layer and a vocabulary of control terms (as in a programming language) is used for composing larger tasks. The $(ML)^2$ formulation does not in this case impose further detail since this layer is represented by quantified dynamic logic (QDL), which augments predicate logic with atomic programs and syntactic constructs expressing sequence, condition, iteration, and so on for composing programs. The final, underdeveloped layer, the *strategic* layer, dynamically plans task execution and handles failure of a partial solution by suggesting new lines of reasoning or introducing new information. In $(ML)^2$ this layer is a metalayer for the task layer and reasons about programs expressed in QDL.

Our scheme makes a sharp distinction between what corresponds to the first two KADS layers—items instantiated in a SID and declared in the static or dynamic ontology—and what (roughly) corresponds to the last two KADS layers—the design nets. Note, in particular, that items declared in the dynamic ontology do not occur in the nets. The atomic conditions and focus names interpret the current SID. These phrases are generated by the semantic grammars, which impose a certain structure on them that somehow relates to the types in the static and dynamic ontologies so that this interpretation can be carried out.

Exactly how the phrase structure and the types relate is not formulated. Both the grammars and the ontologies are sufficiently explicit that encoders generally agree on how the phrases interpret items in the SID, but a correspondence between the two as required by an executable specification is left to be worked out in detail as the need arises. Formalizing and operationalizing this correspondence, we believe, is not part of the framework for encoding protocols. Likewise, although the foci are supposed to account for changes in the SID, we do not consider *how* they do so as within the realm of protocol encoding. Indeed, working out how the foci may modify the SID to allow for even a modest executable specification requires design decisions, fine distinctions between various conditions, and allowance for sequencing among code associated with different foci since several foci may be concurrently active. This underdetermined nature of how changes in the problem state are carried out is to be expected with conceptual design since there is no particular representation associated with conceptual design and generally accepted mathematical conventions in mechanical design have been restricted to later, more detailed design stages. Consequently, how one describes changes in the problem state depends on the representation one uses and how the representation is interpreted.

Introducing a control vocabulary to represent the sequence of design steps, as required by the KADS model, would impose something foreign on a conceptual mechanical design protocol. (Such sources as structured interviews and texts relating to later stages of design, however, are likely to require a QDL-like control vocabulary.) It is much more natural to use the concepts that relate to our design nets: pre- and postconditions of steps, concurrent steps, nondeterminism in the order in which steps are taken, and so on. For one thing, the control vocabulary of a programming language, or of QDL, would require steps to be structured in a much too inflexible way. Conceptual design is opportunistic: among the many possible sequences in which decisions may be taken, a designer typically focuses on some salient part of the problem, works out a partial solution for that part, and uses the constraints and global decisions imposed with that partial solution as a foothold for approaching other parts of the problem. Yet designers follow strategies and, indeed, at the conceptual stage, good design depends heavily on strategy. There is no clash between the strategic and the opportunistic natures of design: the scope covered by a strategy is generally much greater than the scope of an opportunistic choice and the nondeterminism that allows opportunism can be constrained by strategies without being eliminated.

8. Strategy

Research has revealed the importance of strategy to knowledge-based systems. We view strategies as control patterns related to high-level design net descriptions. In 8.1 we briefly review the notions of strategy used in KBS research and introduce our basic notions. In 8.2 we consider the various computer-science formalisms used to capture the temporal order of events as possible ways to represent strategies in conceptual mechanical design. Petri nets are seen to be the most appropriate; this further justifies use of modified Petri nets—design nets—to represent control patterns in this domain since strategies are central to control and shade into more concrete control patterns, which show up in the trace of a protocol. Flexible control patterns are difficult to represent, and abstract flexible patterns—strategies—must be identified in protocols, represented, and related to net traces. We address some of these conceptual difficulties in 8.3.

8.1. Basic Notions

Gruber (1989) relates strategy or *strategic knowledge* to the order and selection of actions and characterizes it as knowledge used to decide what action with real-world consequences to perform in a given situation. Strategic knowledge, then, is manifest in activities from asking questions in an intelligent order to recommending a sequence of corrective actions. Gruber contrasts strategic knowledge, used to evaluate possible actions given a state, with *substantive knowledge*, used to identify states in the world. If we ignore the model (*cf.* "possible") aspects of Gruber's strategic knowledge, it essentially imposes temporal relations on what is covered by substantive knowledge. When a planner has a complete model of the world, the goals, and the effects of actions, then planning can be treated as search. Otherwise, Gruber claims, planning must use strategic knowledge in one of two ways. Strategic knowledge could be implicit in skeletal plans that are specified in advance and refined as the need arises, or it could be exercised in "reactive planning", making dynamic decisions by relating a current situation to a goal. KADS' task knowledge (the third of the four layers) is similar to Gruber's strategic knowledge but does not require that the actions have real-world effects. KADS' strategic knowledge (the fourth layer), among other things, determines what goals are relevant, dynamically plans task execution, and recovers from impasses. We talk about strategies not strategic knowledge. In our case, a strategy is like a skeletal plan but admits steps where, for example, goals are established and plans are filled out up to a certain horizon. [A somewhat similar notion of a "plan" in engineering design is given in Brown and Chandrasekaran, (1989).] Again, since conceptual design is opportunistic, we need strategies that may be refined so frequently that we could call the refinement reactive. Finally, we would include as a strategy an efficient heuristic (or policy) used with algorithms. In our account, more than one strategy may be active.

We conceptualize a strategy as a high-level description of the essential foci and their conditions, essentially a summary of a family of design instances. To achieve a summary, some foci can be designated as intermediate goals; carrying out the strategy then involves finding an achievable sequence of transitions to reach these foci. In Petri nets in general, there are ways to collapse a set of adjacent places (foci) and transitions into a single non-primitive transition. This would allow a hierarchy of strategies, where some foci in a strategy higher in the hierarchy correspond to sets of foci and transitions in a strategy lower in the hierarchy. We have only sketched out the effort required to resolve the technical issues alluded to in this brief paragraph. We believe that a major contribution of our research is to have isolated these issues since the notion of a strategy is perhaps the key notion in formalizing conceptual design and establishing specifications for software support for conceptual design.

8.2. Temporal Formalisms for Representing Strategies

Several formalisms have been used in computer science to model temporal aspects of systems, particularly concurrency. In addition to Petri nets and QDL, there are *temporal logics* and *process algebras*. A temporal logic (Manna and Pnueli, 1992) includes temporal operators (such as those corresponding to the English terms "always" and "eventually") that are applied to propositions to indicate the extent in time these propositions hold. Temporal logics extend predicate logic, which generally suffices to describe the effect of any *transformational* program. Such a program is a usual, sequential program, whose role is to produce a final result at the end of a terminating computation. The power of a temporal logic, however, is needed to describe the effects of a *reactive* program (such as an operating system or a program controlling a real-world process), whose role is to maintain some ongoing interaction with its environment. The notions of reactivity and concurrency are closely related: in any program containing parallel processes (processes running concurrently), from the view point of each process the rest of the program is the environment. A process algebra [see Hennessey (1988) and Baeten and Weijland (1990)] is a mathematical language with basic constants, operators to construct larger processes, and equations as axioms to describe the nature of processes. Thus, we speak of *process terms*, which denote processes, and process algebras are subsumed under that part of modern algebra called term algebras. The interaction of a process with its users (which may be other processes) is thought of abstractly as communication. Concurrency arises because there can be more than one user and, inside the process, more than one active subprocess.

Olderog (1991) considers different levels of abstraction at which concurrent processes may be described and specified. At the most abstract level are logic formulas, which, it is held, *specify* the communication behavior required. At the next level are process terms, which constitute an abstract concurrent language stressing compositionality. At the least abstract level are Petri nets, which describe processes as interacting "machines" (in the automata-theoretic sense). As models of computation, Petri nets are more powerful than finite automata. Statecharts (Harel, 1987) are another formalism introduced to specify the behavior of complex reactive systems. They extend finite automata with features that allow for hierarchical descriptions, interlevel transitions, and multilevel concurrency.

Petri nets, the least abstract of the three formalisms considered by Olderog, are, we maintain, the most appropriate formalism to represent strategy since Petri net transitions correspond directly to protocol episodes. The formal insight they provide is to view conceptual mechanical design as a sequence of interdependent events. As specifications for design software, design nets identify specific steps. Temporal logics allow us to specify desirable properties holding of program variables. Such properties include, for example, safety properties, which state that certain relations among certain variables always hold or always do not hold (that is, never hold). They also include liveness properties, which state that a certain relation eventually holds among certain program variables. Thus, temporal logics are appropriate for expressing certain global invariants (safety properties) and goals (liveness properties), but these properties are much more abstract than the sequence of episodes in a protocol. The communication and composition expressed by process algebras could be useful in our scheme in describing the coordination among different activations of the various design nets since these activations are essentially processes. On their own, process algebras do not support a notion of strategy since the communication and composition they describe assumes there is already something of which there are activations and which directly relates to the sequence of episodes; process algebras are too abstract (although not as abstract as temporal logics). QDL, on the other hand, is less abstract than Petri nets even though it is similar to temporal logics but with modalities constructed using operators reminiscent of those of process algebras. QDL is less abstract because its operators in fact are control primitives for building *programs* (not processes) from atomic programs. We previously stated that this control vocabulary is foreign to conceptual mechanical design protocols.

We could draw on the temporal formalisms other than Petri nets to help define strategies and to improve encoding reliability. This need not add more steps to an already extensive encoding procedure, for other formalisms might only add high-level guidelines, theoretical support, or specifications to be met by the encoding procedure itself. We have already mentioned that eventuality properties expressed in temporal logics give a notion of a goal and that process algebras give an explicit and sound way to relate net activations. Temporal logics could also relate to the atomic conditions and focus names generated by the semantic grammars. It might be useful to express that some condition should eventually, always, or never hold. Or, concerning foci, we might specify that, if we have considered certain problem aspects, then eventually we should consider certain other aspects. Handling within a logic the phrases generated by the grammars might give a welcome logical link between the grammars and the ontologies, which declare the items in the SID to which the phrases refer. It might even be useful to specify in a temporal logic properties that must hold of any conceptual mechanical design activity, such as the following (where we give the English phrase that translates the formula):

If a task T is decomposed into two subtasks T_1 and T_2, then eventually either T_1 and T_2 are realized by the same entity or T_1 (respectively, T_2) is realized by an entity E_1 (respectively, E_2) and the behaviors of E_1 and E_2 are coordinated in such a way that the behavior required for T is achieved.

Again, the hierarchical nature of statecharts could relate to the hierarchies we have noted among strategies. Finally, going beyond protocol encoding and aiming for our ideal of executable encodings, the various temporal formalisms could be very useful in specifying how the actions of various concurrently active foci should coordinate to effect an update to the SID.

8.3. *Additional Considerations*

The conceptual difficulties that arise when we attempt to represent strategies are, we suspect, related to the reasons mentioned in 6.2 that cause unacceptable reliability scores for our encodings. Fundamentally, two encoders may encode a part of a protocol in what are intuitively similar ways, but the reliability score for that part of the protocol may be poor since we lack a way to indicate when differences are small and that the differences may tend to cancel. In the end, we probably must accept that there is not a unique mapping of a sequence of protocol episodes into a design net trace. Any number of traces may be acceptable as long as there are well defined ways to transform one acceptable trace into another. One could try to define operations that add or remove detail or transform one subnet into another. Such changes will show up in the encoded trace, which is formally a string of symbols, and there are definitions of *edit distance* between strings and of *time warping* to compress or expand one string to match another (Sankoff and Kruskal, 1983). Also, there is a metric function called *synchronic distance* (Reisig, 1985) that indicates the coupling between sets of transitions. The distinction among the three design nets rests on the distinction among the three pairs of semantic grammars for the nets. We need a more principled way to distinguish the vocabularies of these grammars so they may shade into one another and capture conceptual relations among design nets.

The indeterminacy accepted in the last paragraph is to be expected. We claim not that the nets are part of a designer's mental apparatus but only that they capture the temporal dependencies among protocol episodes and how these episodes address and depend on the evolving problem state. Translation from the protocol text into our formalisms must be somewhat indeterminate since we go from a largely informal language to a formal language.

A generally recognized control aspect of design that presents difficulties for our nets is *backtracking*. Backtracking is natural in an area, such as design, whose solutions involve combining parts. If an attempted combination must be rejected, we backtrack to the point where that combination was selected and select another combination. Very little backtracking of this simple, sequential nature was found in our protocols, although (as discussed in Section 3.2) often two or more alternatives are concurrently available and are evaluated against each other and unpromising alternatives are rejected. Also, designers review their designs, which may lead to certain parts being rejected. Even the rejection, however, was seen as coupled with elaboration (or re-elaboration). Perhaps, then, the appropriate notion is that of *intelligent backtracking*, where, rather than selecting the next combination in some arbitrary enumeration, one repairs the previous combination only where it needs repairing.

To allow backtracking in the design nets, we would have to assume that a sequence of net states (markings) is remembered and that, when a line of reasoning is abandoned, there is some way to determine where the "wrong turn" was taken, some way to restore the states of the nets just before the wrong turn was taken, and some way to erase the elaboration done to the SID since the wrong turn was taken. A certain amount of intelligence could be imparted to this scheme by being specific about the wrong turn (for example, a certain transition in a certain net), backing up the net states only as these depend on that transition, and undoing only the changes to the SID that are responsible for the problems. In fact, introducing backtracking into the nets would probably be an unnecessary complication. The appropriate notion here is again strategy since a useful strategy references control patterns for handling rejection of partial solutions and control patterns for reviews. These additional control patterns would not necessarily introduce many new foci since it is plausible that foci not only can elaborate the problem state but also can retract previous elaborations and even replace (retract and re-elaborate) items in light of new considerations.

9. Design Paradigms

A design paradigm is a research community's unifying vision of the nature of design that directs performance, investigation, and automation of design. A design paradigm should help one make sense of design activity and should suggest useful ways to solve design problems. In this chapter, we have not yet addressed the question of a paradigm appropriate for conceptual mechanical design, yet viewing design from the perspective of a given paradigm profoundly impacts how protocols are encoded. In this section, we concentrate on two design paradigms, which were investigated in projects at the University of Minnesota that carried on from our protocol analysis project (Esterline et al., 1992). The paradigms are *case-based design* (in 9.1) and *generative constraint based designed* (in 9.2). In 9.3 we consider the extent to which these two paradigms account for all aspects of design; we also consider the implications for protocol analysis of viewing design from the perspective of these and similar paradigms.

9.1. Case-Based Design (CBD)

Case-based reasoning (CBR) is reasoning from precedents, adapting old solutions to solve new problems, or retrieving old cases to illustrate aspects of the current situation. Case-based reasoning improves problem-solving behavior by, for example, using shortcuts, anticipating and avoiding errors, and appropriately focusing its reasoning (Kolodner and Simpson 1989). We refer to precedent-directed design as case-based design (CBD). [For some early work in CBD, see Goel and Chandrasekaran (1989) and Sycara and Navinchandra (1989)]. In principle, any aspect of design can exploit previous cases, and, indeed, our protocols themselves are records of extended cases of design. Also, storing a case can be viewed as knowledge acquisition. Traditionally, CBD systems fully automate this acquisition. The CBD system implemented in the project at Minnesota (Bose et al., 1992a; 1992b) even automated construction of the initial cases, but projected enhancements extend to realms with more flexible and extended conceptual design stages, realms in which our representations are applied.

The current CBD system performs preliminary four-bar linkage synthesis and contains four modules. The Retriever uses similarity metrics to retrieve cases with functional properties similar to the problem specification. The Potential Evaluator studies the functional differences between each retrieved case and the problem and passes the case with the most potential, along with an abstraction of its violations of specifications, to the Adapter. If success is not forthcoming with this case, successively less promising cases are passed until success ensues or no promising cases remain. The Adapter uses the violation abstractions to alter the case structurally to reduce the violations. The Simulator simulates the adapted case. If this indicates compliance with the specifications, we have success and a candidate for inclusion in case memory. Otherwise, the adapted case is sent back to the Potential Evaluator, and the evaluate–adapt–simulate loop is repeated unless the number of iterations exceeds a certain threshold.

Slightly more involved mechanism design problems introduced a more global design perspective, involving modest strategies for problems not yet clearly formulated. This level was attacked by adapting Flanagan's *critical incident technique* (Flanagan, 1962), giving what was called the *critical instance technique.* [Compare with Waldron and Waldron (1987).] One application involves presenting task-mechanism pairs (where the mechanism is intended to realize the task) and asking the observer (an experienced designer) to pick where the mechanism is particularly effective or ineffective in realizing the task; in so doing, the observer identifies the key features of both that are significant for the evaluation. The result of the study is a mapping from mechanism features to task features. Given this mapping, one can present task–mechanism–design strategy triples and ask the observer to pick where the strategy is particularly effective or ineffective in designing an instance of the mechanism for realizing the task; one can also ask for the most effective strategy for a given task–mechanism pair and thus increase the stock of strategies. When the mechanisms are linkages that are relatively simple (say, six-bars) but not too simple (four-bars), this method gives an extensive and controlled set of design cases with guidelines for adaptation and limits on the strategies. It also gives failure cases, which allow failure to be anticipated hence avoided. For a six-bar that is designed as a four-bar with dyad added, the strategies are nearly algorithms. In less clear cases, the strategies are flexible and more like skeletal plans. If a CBD system based on the mappings discussed here were implemented for six-bars, it would apparently include as a subsystem the CBD system for four-bars since one way to construct a six-bar is from a base four-bar. We would then have an evaluate–adapt–simulate loop nested within another. When the artifacts designed are reasonably complex, a more abstract and global view of the problem, involving more extensive strategies that start earlier in the design process, is needed for the problem solver to avoid being overwhelmed by evaluate–adapt–simulate loops.

When the mechanisms of interest are, say, six-bars, although some design net structure is evident and general notions (such as that of a task or of an entity) from the ontologies are applied, there is little need to rely on our representation schemes. In more general and flexible design domains, more advanced representations are needed and cases require correspondingly more analysis. If we think of one of our protocols as a case, then a case corresponds to a specific SID and a trace of a set of net activations. The ontologies and design nets, in this context, can be thought of as integrating any number of cases since the schemes are updated whenever they are unable to represent some aspect of a protocol.

Note that the knowledge acquisition effort required for a particular case increases as the domain becomes more flexible or general. In the four-bar case, there is no knowledge acquisition except automatically storing cases. For slightly more involved mechanisms, the critical instance technique is used to perform *knowledge elicitation.* A domain model is already at hand and the expert designer only fills in details within this framework. For general mechanical design, the structure must be extracted from the protocol. There is automation appropriate for any point in this spectrum. In the simple case, all is automated, and we discussed software for protocol analysis in Section 7. Knowledge editors (Musen, 1989), where the user enters and refines the contents of the knowledge base, have become popular for knowledge elicitation.

9.2. Generative Constraint Based Design (GCBD)

The other project that carried on from our protocol analysis project addressed constraints and grammars for mechanical design. Constraint networks have been used for some time to represent and to solve design problems. Initially, local constraint propagation techniques dominated, but more global techniques [see, e.g., Finger (1987)] are now popular. The formal grammars most widely used for engineering design are graph grammars (Ehrig, 1979). Graph grammars are similar to our semantic grammars except that the non-terminals (variables) and terminals are now vertex labels and the target that is rewritten is a graph not a string of symbols. In engineering design, Finger and Rinderle (1990) and Rinderle and Balasubramanian (1990), for example, represent a behavior specification by a bond graph and, to realize the specification with only available components, define a bond graph grammar and associate a bond graph with each available component.

The project at Minnesota (Shanmugavelu et al., 1991; 1992) represented constraint networks with *hypegraphs.* A graph generalizes to a hypergraph, whose *hyperedges* may connect zero or more vertices. Hypergraphs are used because graphs can represent only binary relations whereas hypergraphs can represent properties and relations of any arity. To modify or otherwise generate hypergraphs, hyperedge replacement (HR) grammars (Courcelle, 1990) are used. In an HR grammar, the nonterminals and terminals are hyperedge labels, the LHS of a production is a nonterminal, and the RHS is a hypergraph. A production is applied to the target hypergraph by replacing a hyperedge labeled by the LHS with its RHS. Implementation is in the constraint logic programming language CLP(R), which solves constraints over the reals but delays solving a given constraint until values have been supplied for all variables with nonlinear occurrences in it. [For CLP(R), see Cohen (1990); for advantages of CLP(R) in structural design, see Lakmazaheri and Rasdorf, (1989).] CLP(R) allows as subgoals not only Prolog terms but also equality and inequality constraints between arithmetic expressions containing real numbers and variables.

In this project, hypergraph vertices are represented with CLP(R) (real) variables. A (sub)system is considered a hyperedge connecting all the vertices representing the parameters that together define a specific instance of the subsystem. A hyperedge label that is not a relational or arithmetic operator is treated as a CLP(R) functor. A term with a hyperedge label as its principal functor contains variables representing the vertices connected by a hyperedge with that label. Finally, the relational operators in CLP(R) constraints, supplemented with arithmetic operators, are (possibly composite) labels of hyperedges connecting the variables appearing in those constraints.

Three systems were implemented within this framework. The first two address rotary power transmission systems [the taxonomy in Kannapan et al., (1989) was followed] where all rotations are about axes parallel to the three coordinate axes. Trains of spur gears or shafts coupled by V-belts are examples. Such systems are easily decomposed and the equalities and inequalities used in describing them are generally linear in the system parameters. There are usually several kinds of solutions with very different structures for a given problem, but the best kind of solution can usually be found by testing the values of a small number of parameters. The first system is really a CLP(R) programming convention and some reusable code. HR grammar rules are clauses of the form

System :- *Constraints*, *Subsystems*.

The constraints can cause choices among the structurally different kinds of systems. The second system is an HR grammar interpreter written in CLP(R) and thus offers more flexible control since control is not bound to CLP(R)'s implementation of resolution. Given a desired behavior, it applies the rules to produce a hypergraph of a device producing that behavior. The hypergraph is defined by the constraints imposed and the terms maintained in a list. Here hyperedges are discarded when they are replaced so the list of terms for the final hypergraph all represent primitives. Rules are stated as CLP(R) clauses of the form

hg_rule(⟨*Old term*⟩, ⟨*New term list*⟩) :- ⟨*Constraints*⟩.

If no constraint violation arises, the generator can remove the old term from the list of terms for the hypergraph (i.e., from the target expression) and include the new terms in the list. The third system was developed by modifying the form of the clauses implementing HR rules in the first system so that constraints are partitioned into required and preferred constraints. This system was tested by using it for initial selection of dwell linkage models.

Abstracted from the particular implementations, this approach to design is called *generative constraint-based design* (GCBD). The representation of the problem state as a hypergraph was inspired by the SID used in our protocol encodings, which is a constraint network that allows relations of arity greater than two. To relate a SID to the hypergraph maintained by the second system, which maintains only the current form of the problem state, we define the *frontier* of a SID as follows. Given a SID, erase any task or entity that is at the tail of an evolutionary arc and push down to the remaining tasks and entities any properties or relations that are inherited from the items erased. The result is the frontier of the SID. The structure of a frontier of a SID is somewhat different from a hypergraph as used to represent the current problem state, where vertices are real variables representing metric properties, hyperedges are relations among such properties, and tasks and entities are represented by subhypergraphs. Having vertices represent metric properties is something of an artifact of the level of analysis. If an HR grammar is used, elaboration occurs on the hyperedges, but a hyperedge may connect only one vertex so, in effect, correspond to that vertex. With a logic programming language, there is no need to solve numerical constraints and no need for variables to assume only real numbers as values. In general, logic programming languages are good for representing structure and for implementing grammars, so the hypergraph grammar systems could be generalized to prenumerical representations for early design stages.

The simulation of the evolution of a SID by a graph or HR grammar is about as abstract as we can get. An abstract simulation is desirable if we wish to emphasize the pure concepts unobscured by implementation details. Thus, in the prototype encoding interpreter implemented in Parlog, the design net foci were fleshed out with HR grammar rules so that a minimum of implementation detail would be added to what is revealed in the protocol.

Concerning control as expressed by the design nets, the implementation language, CLP(R), already has a sort of strategy since it delays considering a constraint until the only variables in it that remain uninstantiated are linear. The interpreter was developed to free control from CLP(R)'s implementation of resolution. CLP(R), like most logic programming languages, backtracks on failure. An interpreter could catch failures and initiate a more appropriate control pattern. In fact, the interpreter used in the second system simply relies on CLP(R)'s backtracking when failure is encountered, and a major effort would be required to specify control patterns for all possible failures. More generally, the design nets can be used as control specifications for the interpreter by including the names of foci among the terms representing the hypergraph and including the constraints on transitions in rules. There are several ways, using similar implementation techniques, that strategies, as summaries of paths through the design nets, could be implemented.

9.3. Design Paradigms and Representation Schemes

Research in design theory has been converging on the paradigms of CBD and (under various names) GCBD [see Brown and Chandrasekaran (1990) and especially Maher (1990)]. Often the single paradigm of GCBD is treated as two paradigms: transformation and decomposition (Kott and May, 1989). Transformation is usually explained in terms of grammars or something similar but informal. We see grammar rules at work in problem decomposition; also, decomposition imposes constraints on the subproblem that must be satisfied when the subsolutions are combined. We do not, however, see these two paradigms as the whole story, for we view them against a background of strategy and in a larger, reactive context. This is significant because how one characterizes conceptual design has a profound influence on how one analyzes a design protocol.

Viewing design from the perspective of CBD does not impose representation schemes on one's characterization of conceptual design since CBD as a general paradigm is neutral regarding representation. Rather, one must devise representation schemes to sanction retrieval and adaptation procedures used in particular applications of the paradigm. For interpreting conceptual design protocols, the perspective afforded by CBD is useful for at least two reasons First of all, it suggests that we must accept without involved explanation certain leaps in a designer's line of reasoning. For here the designer might be following a precedent, so, apart from similarity between the current case and the precedent, there can be no deeper explanation than that the designer happened to address a similar problem previously. Secondly, protocols themselves are cases. Attempting to relate the encodings of several protocols (especially when collected from different designers) encounters many of the same problems that arise when CBD attempts to characterize different cases as similar or to derive one case from another by certain adaptations.

Viewing design from the perspective of GCBD, on the other hand, does have representational implications: the problem state is conceived as a constraint network and its evolution is viewed in terms of grammar rule applications. The general paradigm, however, is mute about what domain the constraints are over. The project described above considered constraints over the real numbers, where CLP(R) is appropriate, but conceptual design constraints may be over various domains. (Indeed, CLP(R) is just one possible instance of CLP(D), where D is the constraint domain.) Assuming one is interested in a representation that is allied to computation, there are two points in particular that make GCBD attractive. Firstly, most interesting models of computation are equivalent to some family of grammars. Secondly, any set of first-order predicate logic formulas can be viewed as a system of constraints, and extensions to first-order logic can be seen as allowing more expressive constraints; so anything mathematically expressible can be expressed as a system of constraints. Whether it is *natural* or *useful* to view the problem state and its evolution in conceptual mechanical design in these terms is another question, one to which this chapter has given a positive answer supplemented with a framework for control patterns.

A paradigm of general problem solving that is considerably less expressive and flexible than GCBD yet shares its transformational nature is given in Newell and Simon's (1972) *Human Problem Solving* and has influenced most protocol analysis projects. It is assumed that the problem specifies an initial state, that there is a fixed set of operators that can transform the current state into the next state, and that a new state is evaluated against the goal state to determine whether progress has been made. Although this paradigm is reasonable for embodiment design, where there is a fixed set of design parameters, in conceptual design the operators, and so the search space, are not even defined until the problem is sufficiently formulated. And, once the problem is formulated, unless care is taken to maintain constraints in a usable form, the number of ways to transform the problem state that must be considered can become overwhelming. Our protocols revealed little overt evaluation and our representations of evaluations are meager. It appears that experienced designers are adept at elevating constraints from their role as "filters," in which they are used to accept or reject alternatives, to the role of "generators," in which they are used to create a tightly bounded number of alternatives. This says more than that constraints become "compiled" hence covertly applied, for it says something about a change in their abstract role. Finally, our protocols show little sense of progress *to* a goal but do show designers striving to keep on track so as to progress *along* a strategy.

10. Conclusion

We have presented formally-based schemes to represent the knowledge revealed in limited conceptual mechanical design protocols. We have also sketched our method of collecting and encoding such protocols. A protocol is directly encoded into a *trace*, consisting of three documents, one for each net, encoded concurrently. The trace partitions the protocol into (possibly overlapping or coinciding) *episodes*, which correspond to net transitions. For each episode, we record the subproblem in which it occurs, the input and output foci of the net transition, the SID conditions that enabled the transition, and the changes the newly active foci make to the SID. The SID (*structured instance diagram*) for a protocol is constructed as the trace is encoded. It represents the designer's view of the evolving problem state. Items recorded in the SID are tasks, entities, properties, and relations; certain properties and relations are *distinguished*, forming a predefined special part of the scheme. Both a graphical and a tabular form of the SID are maintained. The static aspects of the SID represent a snapshot of the problem state and its dynamic aspects indicate how the problem state evolves. Dynamic aspects are represented by certain distinguished relations, which graphically are represented by arcs connecting tasks and entities. The outline of the evolution of the problem is represented by the *task DAG*, which consists of the tasks and entities in the SID connected by the evolutionary arcs.

For each protocol, there is a trace and a SID, but these instantiate or reflect general structures. These structures are updated as needed when a protocol is encoded, so they contain at least the resources to represent the content of all protocols so far encoded. The *ontologies*, formalized in the domain equations and domain function equations of denotational semantics, form a type system for the items instantiated in a SID; the static ontology relates to static aspects and the dynamic ontology relates to the dynamic aspects of SIDs. The *design nets* are modified Petri nets. Each *focus* ("place") has as its name a phrase describing the design aspect it represents and each *transition* is associated with a condition referring to the SID. An episode in a trace is interpreted as one or more concurrent net transitions. When a focus is first active (receives a token), it accounts for changes in the SID. For a transition to be enabled, not only must all its input foci be marked but also its condition must be true. A transition condition is a Boolean combination of atomic conditions in disjunctive normal form; atomic conditions refer to features in the SID. Focus names and atomic conditions are generated by certain *semantic grammars* to ensure that they are well-defined and meaningful. The grammars divide quite neatly into three groups, inducing three separate nets. There are different activations of all three nets for different subproblems. Essentially, the nets integrate interlocking control patterns that are instantiated in protocols. To make sense out of the flow of a protocol, it is necessary to identify higher level control patterns that, however, can be related to the detail of the nets. These patterns are *strategies* and introduce problems we are still resolving.

Two steps in particular were taken to foster objective and accurate encodings. First of all, certain measures of *reliability* were defined to indicate how well two people agree when encoding a protocol into our representation schemes. Low reliability on some feature of the schemes is reason to reformulate or supplement that feature. Secondly, an encoding team always includes a mechanical engineer, who can explain design steps in terms of mechanical principles. Such steps are necessary because our representations are not operationally defined in terms of the words in a protocol. No matter what steps are taken, however, a certain indeterminacy must remain in our encodings since we translate largely informal language into formal representations.

Conceptual problems, including often unacceptable indeterminacy, mostly relate to the design nets. There is a threat that the nets might become too extensive and fail to capture significant generality, although the semantic grammars help by constraining the number and kinds of foci and atomic conditions allowed. Strategies are a more abstract way to capture generality, but technical problems remain in relating strategies to the detailed nets. We need well-defined ways to determine that two net fragments are similar (or more similar than two other fragments) and to determine that one net fragment is a summary or abstraction of another fragment. Again, the partition into three nets is founded on the differences among the semantic grammars. These grammars might sanction useful finer partitions or partial mergings of the nets.

Eventually, the work described here should be implemented for design automation. We would expect an implementation to draw on only part of our scheme and to use man-machine cooperation. Implementation could be facilitated by relating our schemes to certain design paradigms (as discussed in Section 9). *Case-based design* (CBD) retrieves and adapts previous cases to help solve new problems. It depends heavily on the representations available for retrieving cares similar to the problem at hand and for adapting retrieved cases; our work affords representation schemes that should be exploitable by CBD. CBD has generally not been used to govern control patterns or to suggest strategies. Given an adequate representation of control patterns, however, CBD could (depending on the previous cases recorded) afford automated guidance through much of conceptual design.

Another design paradigm can be called *generative constraint-based design* (GCBD). It views design as elaboration of a constraint network by applying formal grammar productions (graph grammar or hyperedge replacement rules, say) to the problem state and solving the constraint network, most usefully in an opportunistic fashion that allows partial solutions to guide the elaboration. We see GCBD as subsuming the more common paradigms of transformation and decomposition. The SID produced in encoding a protocol can be interpreted as a constraint network. The information in a protocol shows a SID being elaborated but it does not suggest *how* it is elaborated or *how* values meeting the constraints are selected. GCBD supplies answers to these "how" questions and, thus, suggests implementation methods. Furthermore, GCBD's answers to the "how" questions are about as abstract as possible, which is desirable at the level of specifications. GCBD, however, has little to say about control patterns other than emphasizing the advantages of an opportunistic policy.

It is important to consider our work in a broader context. Our protocols address relatively small problems. Larger mechanical design problems are often solved by teams. In teams, coordination is a problem since activities proceed concurrently. But we already handle concurrency since we have concentrated on the conceptual relations that allow a design to proceed. Once we move to a broader context, conceptual design is linked not only with later stages of design but also with possibly complex physical, economic, and social systems that transcend design proper. [This broader context is addressed by concurrent engineering—see Sprague et al. (1991), Rosenblatt et al., (1991), and Nevins and Whitney (1989).] It is interesting to consider how our schemes could be enlarged to accommodate these designtranscending aspects. Obviously, the ontologies would become much larger and a typical SID would be much more complex. Evolutionary relations would have to capture notions other than those that relate to decomposition and realization, but the fundamental categories of tasks, entities, properties, and relations—and the general framework built around them—would still apply. Three design nets would no longer suffice when new realms are taken into account. The indeterminacy we noted in the control patterns for conceptual design might be less severe in these new realms, which may be governed by policies or conventions that have evolved or been imposed to coordinate the activities of cooperating agents.

References

Alexander, J. H., Freiling, M. J., Shulman, S. J., Rehfuss, S., and Messick, S. L. (1987). Ontological Analysis: An Ongoing Experiment, *Int. J. of Man-Machine Studies 26*, 473–485.

Allen, J. (1983). Maintaining knowledge about temporal intervals. *Communications of the ACM*, *26*(11), 832–843.

Anjewierden, A., Wielemaker, J., and Toussaint, C. (1992). Shelley—computer aided knowledge engineering. *Knowledge Acquisition 4*(1).

Baeten, J. C. M., and Weijland, W. P. (1990). *Process Algebra.* Cambridge: Cambridge University Press.

Bose, A., Gini, M., Riley, D. R., and Esterline, A. (1992a). On reusing linkage designs. *Proc. 1992 International Conference on Tools for AI.*

Bose, A. (1992b). *Case Based Design of Planar Linkages.* Ph.D. thesis, Dept. of Computer Science, University of Minnesota.

Brown, D. C. (1985). Capturing mechanical design knowledge. *ASME CIE*, Boston, MA, pp. 121–129.

Brown, D. C., and Chandrasekaran, B. (1989). *Design Problem Solving: Knowledge Structures and Control Strategies*,London: Pittman.

Chen, P. P. (1976). The entity-relationship model—Toward a unified view of data. *ACM Trans. on Database Systems, 1*(1).

Cohen, J. (1990). Constraint logic programming languages. *Communications of the ACM, 33*(7), 52–68.

Courcelle, B. (1990). Graph rewriting: An algebraic and logic approach. In J. van Leeuwen (Ed.). *Handbook of Theoretical Computer Science.* Amsterdam: Elsevier Sci. Pub., Vol. B, pp. 193–242.

Diederich, J., Ruhmann, I., and May, M. (1987). KRITON: A knowledge acquisition tool for expert systems. *International Journal of Man Machine Studies 26*, pp. 29–40.

Ehrig, H. (1979). Introduction to the algebraic theory of graph grammars (a survey). In V. Claus, H. Ehrig, and G. Rozenberg (Eds.). *Graph-Grammars and Their Applications to Computer Science and Biology.* Berlin: Springer-Verlag, pp. 1–69.

Ericsson, K. A., and Simon, H. A. (1984), *Procotol Analysis: Verbal Reports as Data.* Cambridge, MA: The MIT Press.

Esterline, A., Riley, D. R., and Erdman, A. G. (1989). Design theory and AI Implementations of design methodology. *NSF Engineering Design Research Conf.*, Univ. of Massachusetts, Amherst, MA, pp. 205–220.

Esterline, A., Bose, A., Shanmugavelu, I., Riley, D. R., and Erdman, A. G. (1992).

Concepts and paradigms for the early stages of mechanical design. *Proc. 1992 NSF Design and Manuf. Systems Conference.*

Finger, J. J. (1987) *Exploiting Constraints in Design Synthesis.* Ph.D. thesis, Computer Science Dept., Stanford University.

Finger, S., and Rinderle, J. R. (1990). Transforming Behavioral and Physical Representations of Mechanical Designs. In P. A. Fitzhorn (Ed.). *Proc. First Int. Workshop on Formal Methods in Engineering Design, Manufacturing, and Assembly.* Fort Collins, CO: Dept. of Mechanical Eng., Colorado State Univ.

Flanagan, J. C. (1962). *Measuring Human Performance.* Pittsburgh: The American Institute of Research.

Freiling, M., Alexander, J., Messick, S., Rehfuss, S., and Shulman, S. (1985). Starting a knowledge engineering project: A step-by-step approach. *The AI Magazine*, Fall, 150–164.

Freiling, M. J. (1988). Designing an inference engine: from ontology to control. *1988 Int. Workshop on Artificial Intelligence for Industrial Applications*, Hitachi City, Japan, 1988.

Goel, A., and Chandrasekaran, B. (1989). Functional representation of designs and redesign problem solving, *IJCAI-89*, Palo Alto, CA: Morgan Kaufmann, pp. 1388–1394.

Gordon, M. J. C. (1979). *The Denotational Description of Programming Languages.* New York: Springer-Verlag.

Gregory, S. (1987). *Parallel Logic Programming in PARLOG.* Reading, MA: Addison-Wesley.

Gruber, T. R. (1989). *The Acquisition of Strategic Knowledge.* Boston: Academic Press.

Harel, D. (1987). Statecharts: A visual formalism for complex systems. *Science of Computer Programming 8*, 231–274.

Hayes, P. (1985). The second naive physics manifesto. In Brachman, R. and Levesque, H. (Eds.) *Readings in Knowledge Representation*, Palo Alto, CA: Morgan Kaufmann.

Hennessy, M. (1988). *Algebraic Theory of Processes.* Cambridge, MA: The MIT Press.

Kannapan, S. M., Marshek, M. K., and Gerbert, G. (1989). An approach to nomenclature and classification for mechanical transmissions. Tech. Rep. 214, Mechanical Systems and Design, The Univ. of Texas at Austin.

Keller, R. M. (1976). Formal verification of parallel programs. *Communications of the ACM*, *19*(7).

Kolodner, J. L., and Simpson, R. L. (1989). The mediator: analysis of an early case-based problem solver." *Cognitive Science 13* (4), 507–549.

Kott, A. S., and May, J. H. (1989). Decomposition vs. transformation: Case studies of two models of the design process. In *Proc. ASME Computers in Engineering Conf.* New York: ASME, pp. 1–8.

Lakmazaheri, S., and Rasdorf, W. J. (1989). Constraint logic programming for the analysis and partial synthesis of truss structures. *AI EDAM*, *3*(3), 157–173.

Maher, M. L. (1990). Process models of design thesis. *The AI Magazine*, Winter, pp. 49–58.

Manna, Z., and Pnueli, A. (1992). *The Temporal Logic of Reactive and Concurrent Systems.* New York: Springer-Verlag.

McDermott, D. V. (1982). A temporal logic for reasoning about processes and plans. *Cognitive Science 6*, 101–155.

Motta, E., Rajan, T., and Eisenstadt, M. (1989). A methodology and tool for knowledge acquisition in Keats-2. In Guida, P. and Tasso, G. (Eds.). *Topics in the Design of Expert Systems.* Amsterdam: North-Holland, pp. 265–296.

Musen, M. A. (1989). *Automated Generation of Model-Based Knowledge-Acquisition Tools*. San Mateo, CA: Morgan Kaufmann.

Nevins, J. L., and Whitney, D. E. (Eds.) (1989). *Concurrent Design of Products and Processes*, New York: McGraw-Hill.

Newell, A., and Simon, H. A. (1972). *Human Problem Solving*. Englewood Cliffs, NJ: Prentice-Hall.

Olderog, E.-R. (1991). *Nets, Terms and Formulas*. Cambridge: Cambridge University Press.

Pahl, G., and Beitz, W. (1984). K. Wallace (Ed.). *Engineering Design*, Bath, UK: The Pitman Press.

Peterson, J. L. (1981). *Petri Net Theory and the Modeling of Systems*. Englewood Cliffs, NJ: Prentice-Hall.

Reisig, W. (1985). *Petri Nets: An Introduction*. Berlin: Springer-Verlag.

Riesbeck, C. K., and Schank, R. C. (1989). *Inside Case-Based Reasoning*, Hillsdale, NJ: Erlbaum.

Rinderle, J. R., and Balasubramanian (1990). Automated modeling to support design. *Design Theory and Methodology—DTM '90* (2nd Int. Conf. on Design Theory and Methodology, Chicago, Sept. 1990), New York: ASME, pp. 281–290.

Rosenblatt, A. et al. (1991). Concurrent engineering (Special Report). *IEEE Spectrum*, July, pp. 22–37.

Sankoff, D., and Kruskal, J. B. (Eds.) (1983). *Time Warps, String Edits, and Macromolecules: The Theory and Practice of Sequence Comparison*. Reading, MA: Addison-Wesley.

Shanmugavelu, I. (1992). *A hypergraph grammar with opportunistic control scheme for conceptual design automation with application to mechanism synthesis*. Ph.D. thesis, Dept. of Mechanical Engineering, University of Minnesota, 1992.

Shanmugavelu, I., Esterline, A. Riley, D. R., and Erdman, A. G. (1991). An opportunistic Approach to conceptual design. *1991 Computers in Engineering*. New York: ASME.

Sprague, R. A., Singh, K. J., and Wood, R. J. (1991). Concurrent engineering in product development. *IEEE Design & Tests of Computers*, March 1991, pp. 6–13.

Stoy, J. E. (1977). *Denotational Semantics*, Cambridge, MA: The MIT Press.

Sussman, G. J., and Steele, G. L. (1980). CONSTRAINTS—A Language for expressing almost-hierarchical descriptions. *Artificial Intelligence 14*, 1–39.

Sycara, K. P., and Navinchandra, D. (1989). Integrating case-based reasoning and qualitative reasoning in engineering design. In J. Gero (Ed.), *AI in Engineering Design*. Southampton, UK: Computational Mechanics Publications.

Ullman, D. G., and Dietterich, T. G. (1988). Progress in Understanding the Process of Mechanical Design, *Design Theory '88*, The 1988 NSF Grantee Workshop on Design Theory and Methodology. RPI, Troy, NY: pp. 1–11.

van Benthem, J. (1983). *The Logic of Time*. Dordrecht: Reidel, 1983.

Waldron, K. J., and Waldron, M. B. (1987). A retrospective study of a complex mechanical system design. In M. B. Waldron (Ed.), *The Results from the NSF Workshop on the Design Process*. The Ohio State Univ., pp. 109–141.

Waldron, M. B., and Waldron, K. J. (1989). Empirical study on generation of constraints which direct design decision in conceptual mechanical design. *NSF Engineering Design Research Conference*, Univ. of Mass., Amherst, MA: pp. 15–30.

Wielinga, B. J., Schreiber, A. T., and Breuker, J. A. (1992). KADS: A modelling approach to knowledge engineering. *Knowledge Acquisition 4*(1).

7
Configuring Systems Using Available Assets: A Conceptual, Decision-Based Perspective

P. N. Koch, J. D. Peplinski, F. Mistree, and J. K. Allen

Design using available assets, in the context of theory and methodology, is more a state of research than a state of practice. At a low level of abstraction, design using available assets, or catalog design, is a procedure in which a system design is realized by assembling standard components selected from catalogs. A nearly endless supply of available components and component assemblies, defined in terms of key features, can be stored in catalogs or computer databases as available assets to realize new designs. If this notion of catalog design, or design using available assets, is abstracted to higher levels and implemented in the earliest stages of the design of a product, a consistent method for quickly exploring new designs based on that which already exists can be developed.

In this chapter we present a conceptual framework for designing when using available assets at different levels of abstraction. The foundation for our approach is rooted in the twin paradigms of Decision-Based Design and a Living Systems Analogy (Koch et al., 1994; Mistree et al., 1990). Systems are modeled at a high level of abstraction in terms of design requirements using this analogy of living systems adapted from Living Systems Theory (Miller, 1978). Function level system models are used to identify feasible existing concepts and components to realize a system. The use of Decision Support Problems (Mistree et al., 1993b) to evaluate and select among feasible concepts and to identify feasible configurations of available components is discussed.

1. Frame of Reference

The notion of design using available assets has its roots in the familiar topic of catalog design, a procedure in which a system is assembled by selecting standard components from catalogs (Vadde et al., 1992a). In this definition "components" include various sizes of gears, bearings, shafts, motors, tubing, etc., that are known to exist in catalogs and databases and are accessible to designers. If we begin with this definition of catalog design and expand it to include existing human and physical resources, processes, solution principles, abstract and concrete subsystems, etc., as well as components, we arrive at design using available assets. The end result, however, is more than just a procedure; we feel design using available assets is a *philosophy* for approaching design as much as it is a method for design.

Our starting paradigms for this conceptual perspective of design using available assets are rooted in Decision-Based Design and an analogy drawn from Living Systems Theory. We present a brief overview of these paradigms in the remainder of this section.

Decision-Based Design

At the Second National Symposium on Concurrent Engineering we presented a conceptual model for decision-based concurrent engineering design for the life cycle (Mistree and Muster, 1990). We offer Decision-Based Design (DBD) as a starting point for the creation of design methods that are based on the notion that the principal (not only)[1] role of an engineer in the design of an artifact is to make decisions. We recognize that the implementation of DBD can take many forms, our implementation being the Decision Support Problem (DSP) Technique (Bras and Mistree, 1991; Mistree et al., 1990; Mistree et al., 1993c; Muster and Mistree, 1988). The DSP Technique is being developed and implemented to provide support for human judgment in designing systems that can be manufactured and maintained. The DSP Technique is used in this chapter as a framework within which guidelines for a method of design using available assets are implemented. An extensive review of the relevant literature and a description of the DSP Technique is given in the references cited earlier and is not be repeated here.

Living Systems Analogy

Living Systems Theory (LST) (Miller, 1978; 1990) is a conceptual framework that has been created to develop a unified theory dealing with the hierarchical structure of both living and nonliving systems. Any biological system can be characterized by its roles and functions within the system using the essential characteristics of 20 subsystems. Although LST has been developed to model biological systems, by analogy subsets of the twenty subsystems have been used to model nonliving systems, (Koch, et al., 1994; Miller, 1978, 1990; Miller and Miller, 1992; Swanson and Miller, 1989; Walker and Thiemann, 1990). The Living Systems Analogy (LSA) is postulated for use in dealing with non-living systems, which in effect are viewed as a subset of living systems. LSA is based on the use of the LST icons and provides

[1] This does not exclude other activities that are performed by a designer in the process of design. One classification of these subordinate activities is described in Bras and Mistree (1991).

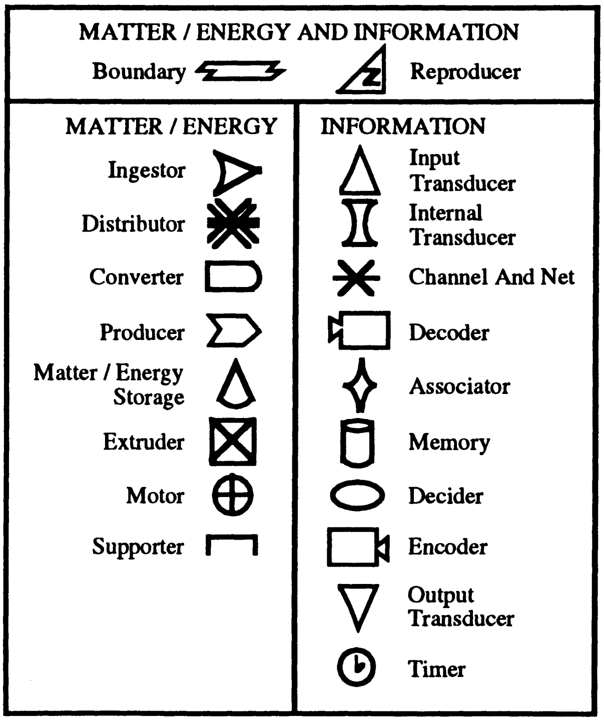


FIGURE 7.1. Symbolic representation of the subsystems of Living Systems Theory (Mistree et al., 1993a).

a useful paradigm, or language, for engineers interested in modeling the design of open engineering systems in general and engineering systems, in particular.

The symbolic representation scheme for the twenty subsystems that compose LST is presented in Figure 7.1. These twenty subsystems are classified into three categories:

- Subsystems (two) that process both **matter/energy** and **information**
- Subsystems (eight) that process **matter/energy**, and
- Subsystems (ten) that process **information**

Systems to be designed can be modeled by assembling the LST subsystems to meet the design requirements given in a problem definition (Koch et al., 1994; Mistree et al., 1993a). These subsystems are "assembled" through the flow of matter/energy and information and by recognizing the system/environment boundary.

Matter/Energy and Information Processing Subsystems

In the first group are the *reproducer* and *boundary*. The *reproducer* is capable of giving rise to other systems similar to itself. The *boundary* is the subsystem at the perimeter of a system that holds together the components that comprise the system, protects them from its environment and excludes or permits entry.

Matter/Energy Processing Subsystems

In the second group (arranged in the order of subsystems that deal mainly with inputs to those that deal mainly with outputs) are the *ingestor*, *distributor*, *converter*, *producer*, *matter/energy storage*, *extruder*, *motor* and *supporter*. The *ingestor* transports matter/energy across the *boundary* from the environment. The *distributor* moves matter/energy around the system; this matter/energy may be external inputs to the system or outputs from other subsystems. The *converter* subsystem transforms some matter/energy inputs to the system into usable forms. The *producer* is the subsystem that forms stable associations among matter/energy inputs to the system or outputs from the *converter*. Materials synthesized by the *producer* are used for the benefit of the system, for growth, for repair, for component replacement and for providing energy for the system's outputs of products or information markers.[2] *Matter/energy storage* is the subsystem that retains packets of matter/energy in the system. The *extruder* transmits matter/energy out of the system in the form of products or wastes. The *motor* moves the system or parts of it in relation to part or all of its environment or moves components of its environment in relation to each other. The *supporter* maintains the system components in a predetermined spatial relationship.

Information Processing Subsystems

The subsystems that process information are in the third group. These include the *input transducer*, *internal transducer*, *channel and net*, *decoder*, *associator*, *memory*, *decider*, *timer*, *encoder*, and *output transducer*. Again, the subsystems are arranged in the order of those that deal mainly with inputs to the system to those that deal mainly with outputs. The *input transducer* is the sensory subsystem that transports markers bearing information into the system. The *internal transducer* receives markers bearing information and transforms them to forms suitable for transmission within the system. The *channel and net* is the subsystem of direct channels by means of which markers bearing information are moved to all parts of the system. The *decoder* subsystem alters the code of information input to it through the input transducer or internal transducer into a private code used internally in the system. The *associator* is the subsystem that deals with learning in its first stages by forming enduring associations among the items of information

[2] Information in living systems is defined as the forms or patterned arrangements of the matter and energy in such a system. The transmission of this information on channels in space or its retention over a period of time requires that the information be embedded in a matter/energy packet which Miller (1978, p. 12) calls a *marker*.

within the system. The *memory* subsystem conducts the second stage of learning, by storing items of information for different periods of time. The *decider* is the executive subsystem that receives information from all other subsystems, processes what it receives, chooses a course of action and then transmits the decision. The *timer* subsystem transmits to the decider information about time-related states of the environment or of components of the system. The *encoder* subsystem alters the code of the information-processing subsystems, from a private code used internally by the system to a public code that can be interpreted and used by other systems in the environment. The *output transducer* transforms information markers used within the system into other matter/energy forms that can be transmitted over channels in the system's environment.

The Living Systems Analogy is particularly useful in the early stages of a product realization process when only the functional requirements of a design problem are known. LSA provides a convenient domain-independent, icon-based language that can be used to represent the means by which these requirements will be satisfied. We contend that the concept of LSA is also particularly useful when designing using available assets. The use of LSA at the function level to represent a system provides a convenient and consistent approach to identifying and exploring feasible existing concepts and components.

In the following section a conceptual framework for designing when using available assets is introduced and defined in terms of the three levels of abstraction mentioned (function, concept, and component level). Each of these levels is elaborated, including example, in the following sections. The use of Decision Support Problems (DSPs) to formulate and solve available assets design problems is also discussed.

2. Design Using Available Assets

In the context of theory and methodology, design using available assets is a current state of research that can be described as a process of configuring systems using that which already exists. At a low level of abstraction, design using available assets becomes the familiar topic of catalog design: a procedure in which a system design is realized by assembling standard components selected from catalogs. Selection in design is a process of making a choice between a number of possibilities while recognizing a number of measures of merit or attributes. Methods including, but not limited to, qualitative optimization, interval approaches, fuzzy set theory, stochastic optimization, subjective design evaluation, and heuristic approaches have been formerly developed to handle various kinds of information in selection (catalog) design problems. Bradley and Agogino have presented a method of qualitative optimization using monotonicity analysis for catalog selection (Bradley and Agogino, 1991). Wilde (1978), Papalambros and Wilde (1988), and Agogino and Almgren (1987) have extensively discussed monotonicity analysis and qualitative optimization. An interval arithmetic method for catalog selection has been proposed by Bradley and Agogino (1991). Habib and Ward have used labeled interval calculus for inferences on catalogs (Habib and Ward, 1990). Moore (1979) has dealt with methods and applications of interval analysis. Baas and Kwakernaak (1977), Allen et al. (1989), and Vadde et al. (1992b) have used fuzzy set theory to solve multiple-attribute selection problems under uncertainty. Stochastic optimization in selection has been considered by Kahne (1975). Subjective design evaluation with multiple attributes was treated by Thurston (1990). Wood (1990) provide a good discussion of the use of fuzzy and probability calculus in engineering design. Waldron et al. (1986) and Mittal and Arya (1986) have explored the application of expert systems to component selection. DeBoer (1989) presents a review of available selection methods.

An overview of these methods as applicable to specific selection problems and useful in specific situations is presented in Vadde et al. (1992a). Vadde and coworkers then suggest that a more general approach is the treatment of coupled selection design problems in the framework of the compromise DSP (Mistree et al., 1993b). This approach is demonstrated in this chapter. For simplicity, the selection of existing components to realize a design is restricted to the crisp form (uncertainty is not included) following that of Bascaran et al. (1989).

Design using available assets, then, can be defined at the component or detailed level as the familiar topic of catalog design, a procedure in which a system is assembled by selecting standard components from catalogs (Vadde et al., 1992a). This definition, however, limits the scope of "design using available assets." A conceptual framework for designing when using available assets is presented in Figure 7.2. Within this framework the definition of available assets is expanded to include assets other than components in catalogs:

Design using Available Assets: *The use of existing human and physical resources, processes, solution principles, abstract and concrete subsystems, living and nonliving subsystems, to realize a system configuration.*

Given this definition, three different levels of abstraction, namely, function, concept and component, for modeling and processing different levels of information detail define the conceptual framework presented in Figure 7.2. Each of these levels, as well as their significance for designing when using available assets, are defined and discussed in this section.

2.1. *Levels of Abstraction*

The initial information available for a product to be designed is captured in the recognition of need and statement of the problem, as noted in Figure 7.2. The representation of a product to be designed begins as a set of functional

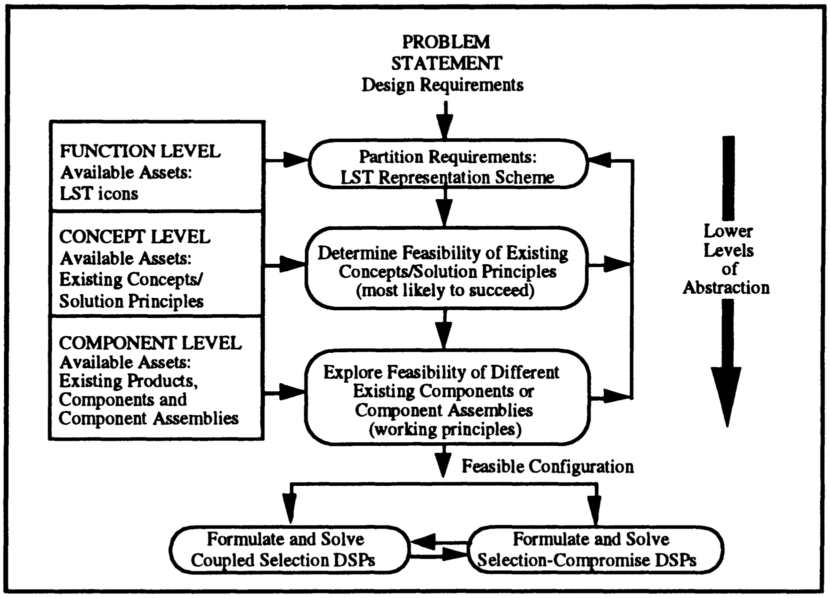

PROBLEM STATEMENT
Design Requirements
FUNCTION LEVEL
Available Assets:
LST icons
CONCEPT LEVEL
Available Assets:
Existing Concepts/ Solution Principles
COMPONENT LEVEL
Available Assets:
Existing Products, Components and Component Assemblies
Partition Requirements: LST Representation Scheme
Determine Feasibility of Existing Concepts/Solution Principles (most likely to succeed)
Explore Feasibility of Different Existing Components or Component Assemblies (working principles)
Lower Levels of Abstraction
Feasible Configuration
Formulate and Solve Coupled Selection DSPs
Formulate and Solve Selection-Compromise DSPs

FIGURE 7.2. A conceptual framework for designing when using available assets incorporating levels of abstraction (Koch et al., 1994).

requirements and specifications, partitioned from a problem statement of requirements for the design, and is transformed into a detailed representation of a physical entity or group of entities. The type of information and knowledge generated about a product changes as its design process progresses. As the conceptual design stage is entered and progresses, a system modeled as requirements must be represented in terms of the concepts explored to meet the requirements. As the selected concepts are defined in the detailed stages, a detailed modeling of the components for realizing the concepts becomes necessary. Regardless of the current stage during a design process, however, the available information about a product being designed can be represented as a system model at different, definitive levels of abstraction. Three such distinct levels of abstraction for modeling systems, defined in terms of the class of information represented and processed at each, are identified (see Figures 7.2, 7.3): the *function* level, the *concept* level, and the *component* level.

These levels of abstraction are defined in terms of the information managed, processed, and represented at each:

- **The function level of abstraction**—the level at which design requirements and systems are represented generically with no mention of conceptual or physical realization.

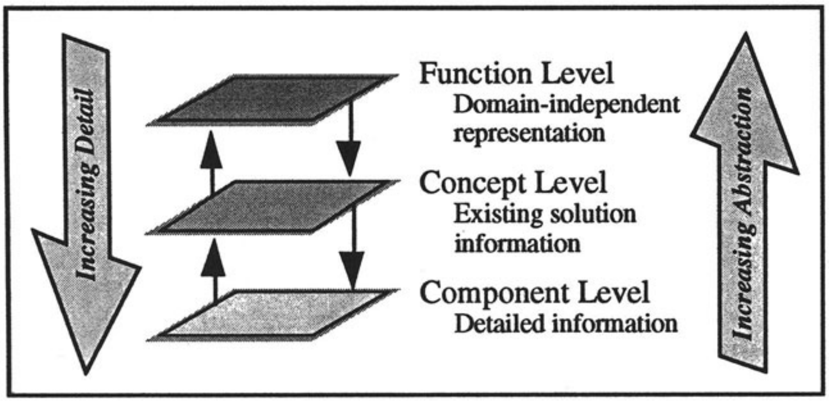


FIGURE 7.3. Layers of information representation for design using available assets.

- **The concept level of abstraction**—the level at which systems are modeled in terms of possible solution principles for fulfillment of the design requirements.
- **The component level of abstraction**—the level at which systems are modeled in terms of specific, physical entities that may realize the chosen concepts and/or fulfill the design requirements.

Defined as such these levels can be viewed as layers of information representation as shown in Figure 7.3. As the model of a product being designed moves from the function to the concept to the component level of abstraction, the detail of information represented by the model steadily increases while the amount of generality and design freedom decreases. When abstracting from the component to the concept to the function level in Figure 7.3, the level of abstraction or generality increases.

The representation and processing of information when designing at each of these levels is elaborated and discussed through example following sections. Before presenting this discussion the conceptual framework for designing when using available assets presented in Figure 7.2, as well as the transformation of information that occurs at each layer in Figure 7.3, is elaborated.

2.2. A Conceptual Framework for Designing When Using Available Assets

The conceptual framework for design using available assets is defined at each of these levels of abstraction in terms of the assets available at each level, and the necessary transformation of information at each layer of Figure 7.3 that allows the design to be represented using these assets. The assets identified as available for designing at each level are presented in Table 7.1. The LST

TABLE 7.1. Assets available for designing at each defined level of abstraction.

Abstraction level	Available assets
Function level	LST icons and representation scheme
Concept level	Existing, defined concepts and solution principles
Component level	Existing, defined products, components, and component assemblies

icons and representation scheme are used to partition and represent a system at the function level of abstraction. At the concept level, existing concepts or solution principles (Pahl and Beitz, 1988) are sought to realize the product specific functions represented at the function level (for example, a shaft and coupling for transferring torque). Existing and readily available physical components found in vendor catalogs are defined as available assets for designing at the component level of abstraction (existing, standard shafts and couplings).

Using these available assets the layers of information representation shown in Figure 7.3 are defined in Figure 7.4 as a transformation of input information into the necessary outputs. The foundation of each transformation is the use of the available assets defined at each level of abstraction to realize the representation of each layer. Between each layer in Figure 7.4, a test of the validity of the information represented is shown that restricts the move between layers.

Given a problem statement containing requirements for the design of an engineering system a model of the system is create using the LST icons and representation scheme at the *function level of abstraction*. Thus the inputs at this level in Figure 7.4 are the available design requirements; the LST icons and scheme are used to transform these requirements into an LSA system model (for example, a *motor* LST subsystem for representing a system motion requirement). After a function level system model has been created, the problem statement and available design requirements are reviewed to ensure that the system model has captured all the available information at this level (completeness and correctness in Figure 7.4) before moving to lower levels.

When confident that the LST system model is complete and correct, a designer can move to the next level of abstraction or representation layer. At the *concept level of abstraction*, then, it is possible to define available assets as existing groups or categories of solution and working principles (Pahl and Beitz, 1988) that have previously been developed and produced and for which developed products are known to exist. The system model of function and specifications is transformed (Figure 7.4) into a conceptual configuration of solution principles. For example, in meeting the system motion

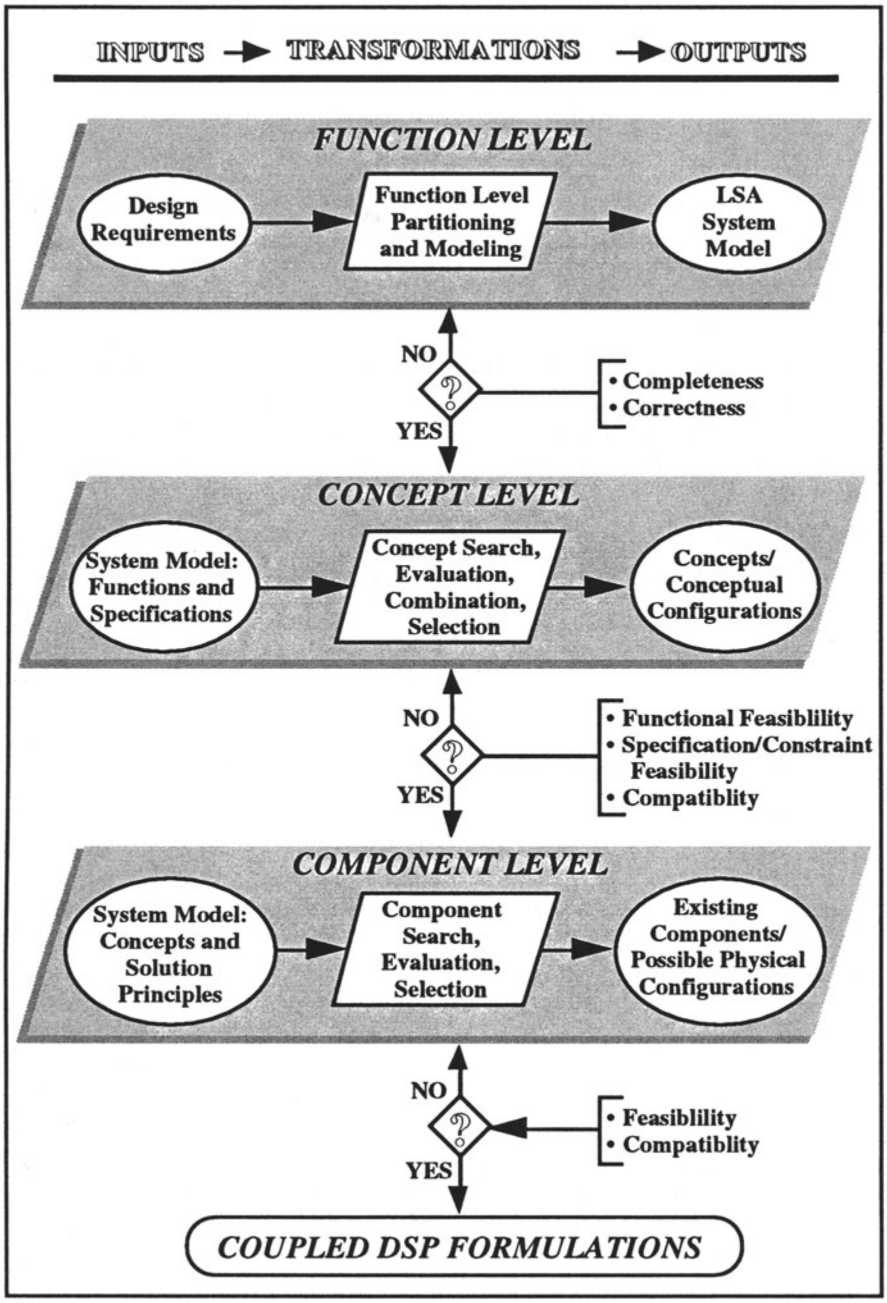

INPUTS → TRANSFORMATIONS → OUTPUTS
FUNCTION LEVEL
Design Requirements
Function Level Partitioning and Modeling
LSA System Model
NO
YES
• Completeness
• Correctness
CONCEPT LEVEL
System Model: Functions and Specifications
Concept Search, Evaluation, Combination, Selection
Concepts/ Conceptual Configurations
NO
YES
• Functional Feasiblility
• Specification/Constraint Feasibility
• Compatiblity
COMPONENT LEVEL
System Model: Concepts and Solution Principles
Component Search, Evaluation, Selection
Existing Components/ Possible Physical Configurations
NO
YES
• Feasiblility
• Compatiblity
COUPLED DSP FORMULATIONS

FIGURE 7.4. Information transformation at each level of abstraction when design using available assets.

requirement modeled as a LST motor icon at the function level, existing concepts include levers and linkages, engines and motors, springs and actuators, thrust producing combustion concepts (rockets), etc. Working products are known to exist for these conceptual components. Before exploring the different types of each at a lower level of abstraction, however, a designer is able to investigate and evaluate these concepts using the design specifications given with the problem statement to reduce the domain of feasible or likely-to-succeed working principles. At this level a test for feasibility allows the appropriate proceeding steps to be determined: move to component level, seek additional concepts, or return to the function level for more detailed partitioning. Three determinants of feasibility are identified: the concepts must meet the functional requirements for the design, they must not violate any of the constraints or specifications, and they must be compatible with one another. If no feasible concepts are known to exist, a designer can further partition the system representation at the function level to provide more detail.

Finally, at the *component level of abstraction*, a designer is able to explore the available assets that exist within the groups or categories of working principles: products, components, and component assemblies that have been previously developed, tested, and marketed, that may meet the functional requirements, the design specifications, and the chosen concepts, for the design problem at hand. At this level of Figure 7.4, the feasible solution principles discovered at the concept level are transformed into physical realizations using existing components. For example, for the motion requirement, if the motor concept is chosen for further exploration, many existing electric motors can be found in catalogs defined by size (in terms of input and output variables). At the component level, then, a designer explores the possibility of using available components for which the product specifications and system performance information is available to realize a concept. The output information represented at this layer is one or more possible physical configurations. Again a test for feasibility and compatibility of each alternative component is developed before selecting among the alternatives. If no feasible products are found to exist, the designer can iterate, going back to the function level and partitioning further, in more detail, or explore other system configurations.

The foundation for this conceptual framework for design using available assets is the transformation (partitioning and modeling) of information at the function level of abstraction using the LSA. Through the consistent and domain-independent representation of function level information that is achieved when using the LST icons, the evaluation and determination of feasibility of existing concepts and components is possible. The strength of the LSA in providing this foundation includes the capability to model existing concepts and components using the same modeling scheme, and thus abstract this detailed information to the function level for evaluation and comparison.

Within this conceptual framework, once types or families of existing components for physically realizing the feasible concepts are identified, a selection of compatible components or component assemblies must be made. The solution method for this selection of existing components is through the formulation and solution of the appropriate DSPs. Decision support will be discussed in Section 5. In the next section designing at the function level is elaborated.

3. Partitioning at the Function Level of Abstraction

Designing at the *function* level of abstraction is defined as a process of partitioning and representing. This necessary partitioning is divided into two principal partitioning activities, as illustrated in Figure 7.5: (1) the breakdown and allocation of requirement information and (2) the decomposition of systems. Support for representing partitioned information is provided through the use of the LST icons and scheme. In this section, these two principal partitioning tasks are elaborated.

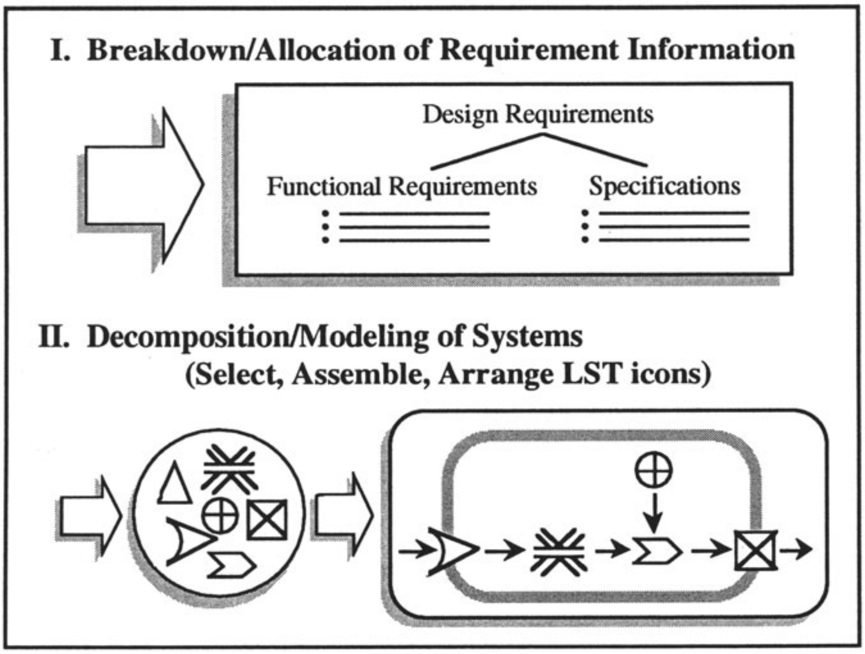

I. Breakdown/Allocation of Requirement Information
Design Requirements
Functional Requirements
Specifications
II. Decomposition/Modeling of Systems
(Select, Assemble, Arrange LST icons)

FIGURE 7.5. Two principal partitioning tasks for designing at the function Level of abstraction.

3.1. Partitioning Design Requirements

The term *design requirements* is employed to encompass the complete body of information that governs the realization of a system to be designed. Information extracted from a design problem statement for the design of a new system can then be classified as either *design requirements* or other information (evaluation of need for a new product, market studies, history, etc.). Only the requirements for a product are addressed here.

Design requirement information can then be classified further as fitting into two categories, as illustrated in Figure 7.6, based on *what* a system must do (functions), and *how* the functions of a system are limited (properties and preferences that a system must meet or should possess). The *what* requirements have already been defined as the functions a system must perform, or *functional requirements*. Functional requirement information includes not only process information (functions) but also information about that which is to be processed (flow information). The necessary system properties and preferences govern how a function is limited, and provide additional flow information. These requirements allow the feasibility of a solution principle to be determined, and/or provide the means for comparison of different solution principles for a particular function or subsystem. We employ the term *specifications* to represent these governing requirements.

Design requirements included in a problem statement provide the necessary information for the initial development of a system model: the functional requirements provide the function and related flow information (pro-

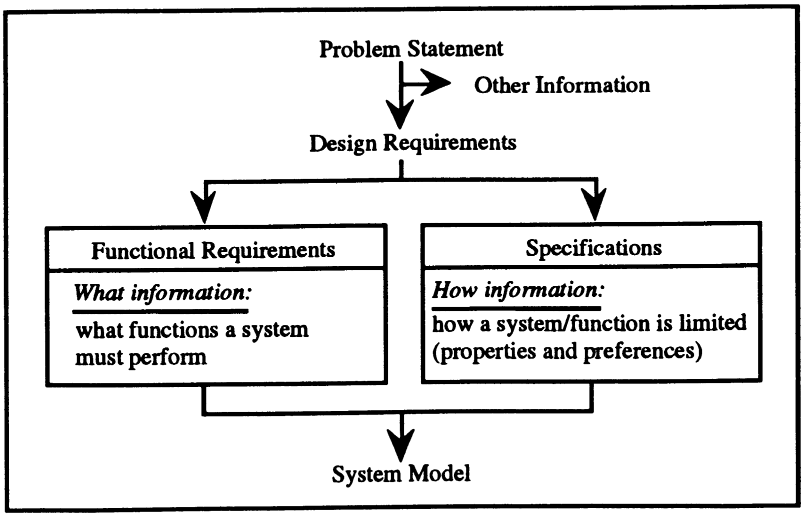


FIGURE 7.6. Classification of information for the breakdown of design requirements (Partitioning task I).

cess and structure) and the specifications provide characteristic information linked to a system model that governs the realization of the system. A problem statement is first partitioned into a set of functions (functional requirements) that the system must meet. The verb/noun function representation employed in much of the literature (O'Shaughnessy and Sturges, 1992) may be used as an aid to extract this information, as the necessary functions may be given in this format (cut grass, store grass, for example). The initial flows (matter, energy, or information) must then be determined. Once the functions and flows have been identified, the specifications are then partitioned, allocating each to the function(s) or flow(s) that they limit. This format of partitioned information will foster the creation of a graphical system model using the LST icons.

3.2. Modeling Systems Using the Living Systems Analogy

Given the partitioned requirements for a design problem (functions, flows and specifications), an initial system model can be created. In defining systems and creating a system model, the capability to establish and consistently represent hierarchy is essential. Agreement is apparent in the literature, that a system is (1) composed of more than one part and (2) is itself one part of larger systems. Without both (1) and (2) a system cannot be defined completely. For engineering applications, the breakdown of a system is defined in terms of subsystems and/or components (at a lower level of detail). For complex systems, this breakdown will likely be in multiple hierarchical levels of subsystems.

In partitioning design requirements the hierarchical structure of the functional requirements will, to a certain extent, develop naturally. A list of functional requirements will necessarily include an overall function and the subfunctions needed to meet the overall function (for example, *move system* and *store grass* sub-functions of the overall function *cut grass* for a lawnmower system). The identification of lower level functions defining ambiguous or complex sub-functions will guide the establishment of a function hierarchy for a system. In creating the function hierarchy, a system (model) hierarchy will be initiated. The system model hierarchy is defined in terms of increasing levels of detail (decreasing abstraction) that includes a system level and one or more subsystem levels. The position of a particular subsystem in the system model hierarchy will be denoted using a superscript as shown in Figure 7.7 (subsystem^2 for second level in system hierarchy, for example).

In establishing a hierarchy of the partitioned functional requirements, it is essential to maintain the assigned specifications. Given the breakdown of design requirements and establishment of hierarchy, the Living Systems Analogy can be implemented. The LST icons are employed to create the initial system model hierarchy.

To select the appropriate LST icon, the flow to be processed must be

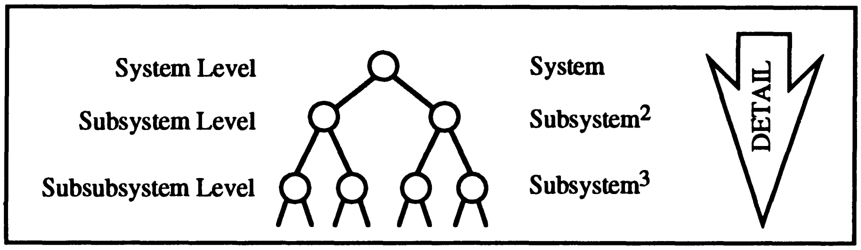


FIGURE 7.7. Levels of detail in a system model hierarchy.

defined as matter, energy, or information for a particular functional requirement. The set of possible subsystems then becomes a subset of the twenty LST subsystems (eight for matter/energy processing, etc.). The necessary process (function) and flow must then be defined clearly. Again, the verb/noun descriptions prevalent in the literature (O'Shaughnessy and Sturges, 1992) are useful, as both the function and the flow are described. The definitions of the LST subsystems within the reduced set are then reviewed, noting that subsystems are organized in order from those that process mainly inputs to those that process outputs. Given the concise description of a requirement and the definitions of the possible subsystems, a match is made, and the appropriate subsystem selected. The specifications listed for the functional requirement must then be assigned to the selected LST subsystem.

Once the necessary LST subsystems have been selected, an initial system model is created by assembling the subsystem icons. The LST icons are assembled through representation of the necessary flows of matter, energy, and information. In addition, the established requirement hierarchy fosters the creation of the system model, and must be observed. With experience and a solid understanding of the LST subsystems and representation scheme, the assembling of icons to create the graphical model becomes a natural mapping of the partitioned requirements, the established hierarchy, and the flow information.

To demonstrate the use of this Living Systems Analogy and partitioning support for designing at the function level of abstraction, the example of the design of an aircraft evacuation system is introduced.

3.3. An Example: Function Level Design of an Aircraft Evacuation System

For the design of an aircraft evacuation system, the following problem statement is given:

An emergency evacuation system for a wide body, passenger carrying aircraft must be designed. There are typically two types of aircraft emergency incidents: on land and on sea. During an emergency, the evacuation system must be capable of safely

TABLE 7.2. Aircraft specifications.

Door dimensions	42″ wide by 74″ high	
Aircraft sill heights	Maximum	= 280″
	Normal	= 200″
	Minimum	= 80″

TABLE 7.3. Evacuation system specifications.

1. The total system weight must be less than 170 lb.
2. When stored, the entire system must fit in a volume no greater than 8.5 ft^3.
3. The system shall be automatically actuated, completely self contained, and with no form of remote control.
4. The system must be ready for use in no more than 6 s from actuation.
5. The cost of the system should be kept to a minimum without sacrificing system performance.
6. The system must be capable of supporting passengers evacuating the aircraft at a rate of 60 passengers per lane per minute without failure at normal sill height.
7. The system must be capable of properly functioning after being exposed to any temperature in the range −40°F to 160°F for a 24 h period.
8. The system must be capable of operating in 25 knot winds from any direction.

conveying passengers to the ground or water and providing flotation, if necessary. The ultimate goal is to evacuate the aircraft while minimizing the probability of passenger injury.

When not in use the evacuation system must not interfere with the normal use of the aircraft. It must not extend past the aircraft fuselage or wing exterior geometry, for aerodynamic reasons, or far into the interior of the aircraft for reasons of operation and passenger comfort. Given these basic requirements, the passenger transport and/or flotation system must be moved from a stored location to a usable position. The system must be completely self contained, and ready for use without remote control in no more than 6 seconds.

The primary design specifications available for the design of the aircraft evacuation system are given in Tables 7.2 and 7.3. These specifications include aircraft dimensional information for compatibility (Table 7.2), and the specifications limiting the design and operation of the evacuation system itself (Table 7.3).

Given the information included in the problem statement and the design specifications for the evacuation system, the requirements for the system can be partitioned. The primary functional requirement for this system is to evacuate passengers. Two evacuation scenarios are defined for which the system must meet the necessary requirements: land evacuation and sea evacuation. As suggested in the specifications in Table 7.3, this problem requires that the system be stored when not in use, and deployed when needed. Thus the complete system design includes not only the system for conveying

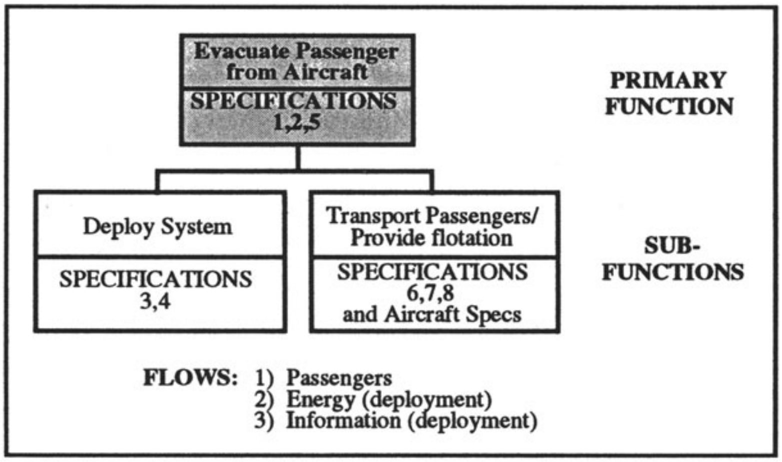

Evacuate Passenger from Aircraft
SPECIFICATIONS 1,2,5
PRIMARY FUNCTION
Deploy System
SPECIFICATIONS 3,4
Transport Passengers/ Provide flotation
SPECIFICATIONS 6,7,8 and Aircraft Specs
SUB-FUNCTIONS
FLOWS: 1) Passengers
2) Energy (deployment)
3) Information (deployment)

FIGURE 7.8. Partitioning of initial requirements for the design of an aircraft passenger evacuation system.

passengers to the ground or providing flotation, but also a system for deploying the transport/flotation system(s). These two systems thus become the primary subsystems of the overall evacuation system.

For the evacuation of passengers and the two identified sub-functions, three primary flows are identified: the flow of passengers when the system is in use, the flow of energy when the system is being deployed, and the flow of information regarding the necessary use (triggering) of the system. The breakdown of available requirements for the initial creation of a system model at the function level is shown in Figure 7.8. The specification numbers assigned or allocated to the primary and sub-functions refer to the specifications listed in Table 7.3. The specifications for the primary function apply to the overall function and thus the entire system.

It is conceivable that the *transport passengers* and *provide flotation* requirements be considered as separate functions for which individual subsystems are designed. Given the space (overall volume) and weight requirements, however, it is desirable that a single subsystem meeting both of these requirements be realized. Thus these functions are grouped in Figure 7.8, providing a starting point for modeling the system at the function level.

Using the structure of the partitioned requirements, a hierarchical representation of the evacuation system is established with two discrete levels: a system level and a subsystem level. The first pass system level model of the evacuation system is represented in Figure 7.9. The borders around the *motor* and *storage* subsystems signify that these are the two main subsystems (as described above) of the overall evacuation system. These subsystems must be defined in more detail at a lower level of the system hierarchy.

At the system level (Figure 7.9), information regarding the state of the

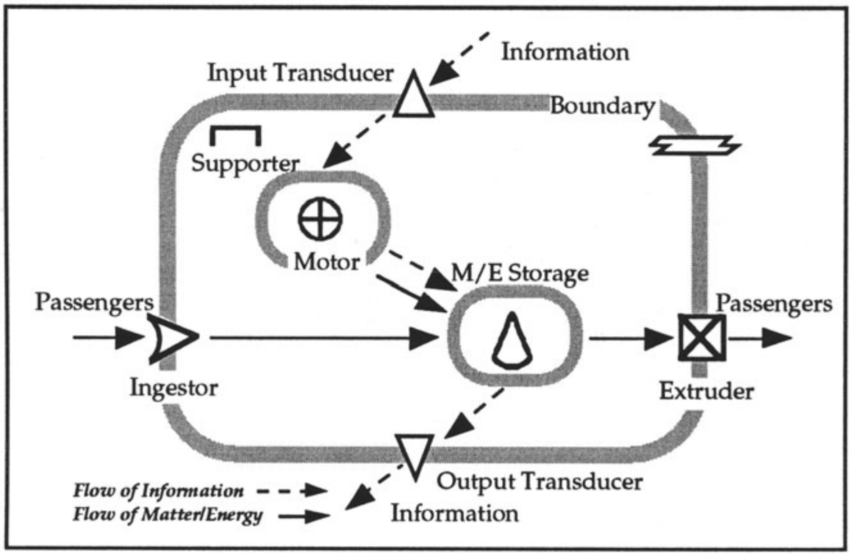


FIGURE 7.9. System level representation of the aircraft evacuation system.

system (storage, activation, deployment) enters the system through an *input transducer*, and is passed to the motor subsystem. This information is processed to determine the current state of the system. During activation, this information signals the motor subsystem to release the necessary energy to move the storage subsystem from its stored state within the aircraft. This information passes from the system through an *output transducer* when activation is not necessary.

The storage subsystem in this model is loosely defined, and thus requires further explanation. According to the problem statement the evacuation system must provide means of safely conveying passengers to the ground and provide flotation in the event of a ditching scenario. As mentioned, these requirements have been grouped. This grouping is accomplished through specifying that the storage time depends on the specific emergency scenario. On land, the time the passengers remain in the system is slight, so the storage system virtually becomes a *distributor*. At sea, the passengers remain in the system for an extended period of time and are thus stored within the system. Information is passed to the storage subsystem regarding the emergency scenario, allowing the storage time required to be determined.

The remaining icons of the system level representation include an *ingestor*, an *extruder*, a *supporter*, and the *boundary*. The supporter and boundary are always present to maintain spatial relations and separate the system and environment, respectively. The ingestor and extruder allow passengers to enter and exit the system (cross the system boundary). These may actually be

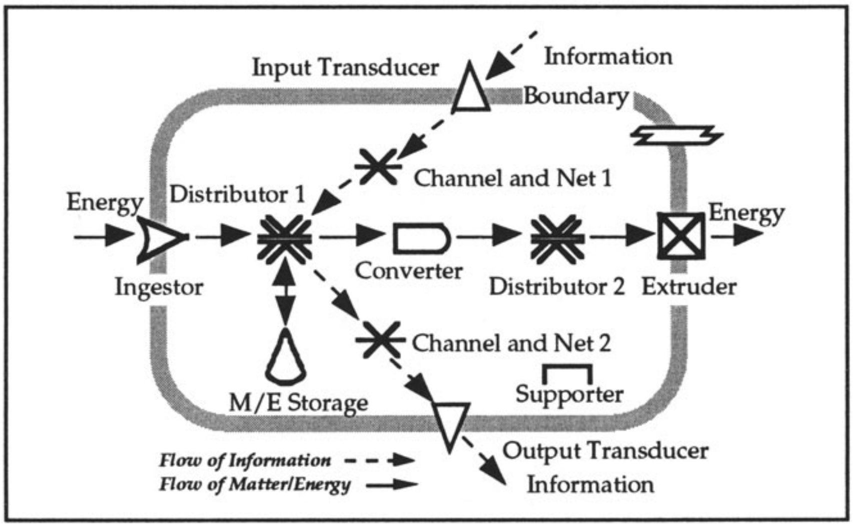


FIGURE 7.10. Motor subsystem^2 representation for the design of an evacuation system.

a part of the storage subsystem but are shown for clarity in the system level representation.

The lower level representation of the motor subsystem^2 (recall that the superscript represents the level in the system hierarchy; second level in this case) is presented in Figure 7.10. At this level the functional components necessary for moving the storage system out of its stored location within the aircraft are modeled. Energy is needed for this deployment. Thus energy enters the system through an *ingestor* and is stored within a *storage* subsystem until it is needed. This ingesting and storing of energy to be used by the system would be done during system maintenance or installation, prior to operation of the aircraft.

Information flows from the environment, through an *input transducer* to the distributor regarding this necessity of deployment. When deployment is necessary the energy stored within the motor subsystem^2 is released by the storage device and flows by way of a *distributor* to the *converter*. The converter changes the energy released from the storage device to the form necessary for use in deployment, as in the case of a chemical reaction. This energy is transported by way of a second *distributor* and exists the motor subsystem^2 through an *extruder*.

The energy exiting the motor subsystem^2 serves the purpose of moving the storage (system level) subsystem from the aircraft. Thus this flow of energy ties the two primary subsystems of this hierarchical representation together. The subsystem^2 level representation of the system level storage system is

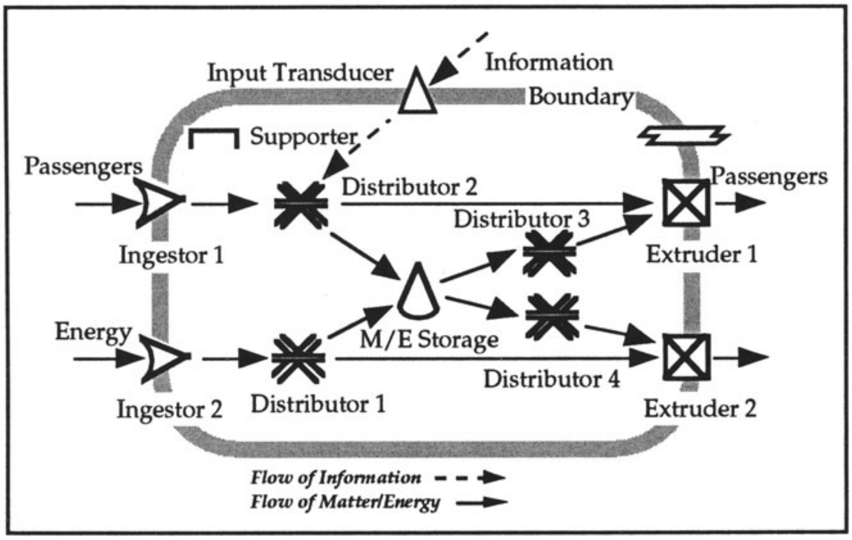


FIGURE 7.11. Passenger transport/storage subsystem^2 representation for the design of an evacuation system.

presented in Figure 7.11. This representation includes two matter/energy flows: The flow of energy from the motor subsystem, and the flow of passengers through the subsystem^2. The energy from the motor subsystem^2 enters the storage subsystem^2 through an *ingestor*, is distributed or stored (*distributor* or *storage*), and leaves the system through an *extruder* when it is no longer needed. At this point it is unknown whether the energy will be stored (inflatable device, for example) or will be used and dissipated (unfolding stairs), and both options are represented. If this energy does not need to be stored (simply used for deployment), it passes directly through this subsystem^2.

In Figure 7.11, the passengers enter the storage subsystem^2 through an *ingestor* and are transported by a *distributor*. Information enters the subsystem^2 through an *input transducer* regarding the evacuation scenario that exists. During land evacuation, the passengers are transported by the distributor to the *extruder* where they leave the system safely. In the event of a ditching scenario, the passengers are transported by the distributor to the *storage* device where they remain until they are transported by a second *distributor* and leave the system through an *extruder*.

This initial representation for the aircraft evacuation system, presented in Figures 7.7, 7.8, and 7.9, captures the information contained in the partitioned requirements, and the additional representation information needed to partition the subsystems. The possible realization of this system and its subsystems using existing concepts and components can now be investigated.

In the next section, feasible concepts and components are identified for the subsystems of the evacuation system.

4. Physical System Realization: Increasing Detail/Decreasing Abstraction

We now return to the design using available assets framework presented in Figure 7.2. The capability to configure systems at the function level of abstraction using the LST icons as available assets has be discussed and illustrated. In this section we progress through the lower levels of abstraction within this framework (concept and component level). In moving to lower levels of abstraction, the level of information detail of a design increases and this method of design using available assets approaches that of catalog design. We elaborate the evacuation system problem for example.

4.1. Identifying Feasible Existing Concepts

Once a feasible function level representation has been created for the product to be designed, the next step is to explore the different concepts that are available to the designers to meet the functional requirements. In taking this step, we progress the design from the function level to the concept level of abstraction. There are many different ways to perform this exploration; in Pahl and Beitz (1988) the following alternatives are offered: literature searches, analysis of existing technical systems, brainstorming, and discursive methods where checklists (that classify the types of concepts available) are used. In a computer environment, a database of existing concepts could be developed and classified in a manner consistent with the representation scheme of LST, and the search for feasible concept alternatives enhanced.

To evacuate passengers from an aircraft, various concepts may include ladders for transport to the ground, air bags to cushion the passengers' jump to the ground, inflatable slide/rafts for the passengers to slide to the ground, or a telescoping segmented slide at each exit door. Assume the inflatable slide/raft concept (in current use) is chosen for further investigation. Given this concept selection, the extruder of the motor subsystem^2 must supply a flow of air to inflate the slide/raft. There are many ways that this pressurized air could be provided; some examples include using compressed air cylinders, inducing a chemical reaction, or continuously pumping in external air. If the components for more than one of these methods are available, then each method could be examined to explore separate feasible alternatives. For the purpose of illustration, just one of the alternatives is pursued: supplying air in compressed air cylinders.

If the compressed air is stored in cylinders, then a valve will be needed to release this air, along with tubing or pipe to transfer the flow of air to the inflatable slide/raft. Reviewing the motor subsystem^2 model in Figure 7.10,

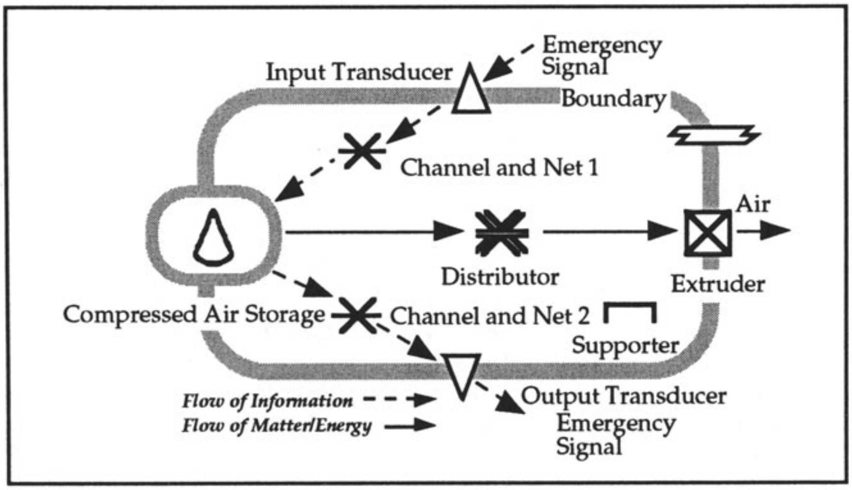


FIGURE 7.12. Motor subsystem^2 using cylinders of compressed air.

we see that the representation no longer describes the current alternative. Modeling a system at the function level using the Living Systems Analogy allows the representation to be changed easily to explore different system configurations. The new representation, specific to using cylinders of compressed air, is shown in Figure 7.12.

This new representation is realized by creating a compressed air storage subsystem that is placed on the motor subsystem^2 boundary. This new subsystem^3 (modeled at the third level), would include the necessary *ingestor* and *extruder* for adding and removing the air (this simple representation is not shown). The valve then becomes the *extruder* for this subsystem^3. To complete the flow of air from its stored state to the slide/raft, a form of tubing or pipe (*distributor*) is needed to transfer the flow of air from the valve to the inflatable slide/raft. The compressed air released from the cylinder(s) serves to simultaneously deploy and inflate the slide/raft.

The motor subsystem^2 representation shown in Figure 7.12 represents one concept level configuration (abstracted to the function level). *Is this configuration feasible*? At the concept level of abstraction, three determinants of feasibility can be identified: the concepts must meet the functional requirements for the design, they must not violate any of the constraints or specifications, and they must be compatible with one another. Recall that the storage subsystem^2 must provide passengers an exit to the ground in an emergency situation, and also must provide flotation if the aircraft lands on water. Reviewing the system model in Figure 7.9, the passenger storage subsystem must transport a flow of passengers, and receive a flow of matter/energy for deployment. At this level of abstraction, an inflatable slide/raft is feasible. The motor subsystem must receive an emergency signal and send a flow of matter/energy to the storage subsystem. The collection of cylinders, tubing, and solenoid valve allows these requirements to be met. Thus the concepts meet the functional requirements for the design.

To the extent that they have been defined, these concepts do not violate constraints at this level. Constraints must be reviewed in more detail upon moving to the component level, since some of the constraints, such as material type, system volume and system weight, air volume and pressure, depend on the specific components chosen. Since the hierarchical selection of concepts was constrained so that each new concept was chosen based on compatibility with the previous choices, the necessary compatibility is ensured. Ideally, the lists of concepts for each subsystem, or group of subsystems as the case may be, would be narrowed simultaneously, taking compatibility into account, so that the design is not restricted by each concept selection. At the concept level of abstraction, the evacuation system design is feasible, and the move to the component level is possible.

4.2. Identifying Feasible Existing Components

At the component level the available assets become specific components or component assemblies that can perform the required subfunctions and for which existing physical realizations are readily available. Defining the design at the concept level of abstraction focuses and simplifies the search for components to those represented by the feasible concepts. For the aircraft evacuation system, catalogs or databases of existing components could be employed to locate available slide/rafts, pressure tanks, valves, and fittings.

At the component level, all the information needed to determine feasibility is available, and the physical relationships between the components are well-defined. For example, if the evacuation system must be less than a specific weight or volume, the total system weight and volume can be computed. Also, physical constraints, such as the capacity of the air cylinders and the air necessary to inflate the slide to provide sufficient sliding support and flotation, can be computed.

The constraints for the design of a system using available assets, and the appropriate design goals, can be formulated mathematically and represented by a coupled Decision Support Problem (Mistree and Muster, 1990). The solutions of the resulting DSP, if they exist, provide feasible system configurations. This formulation and solution of available assets problems as coupled DSPs is the focus of the next section.

5. Decision Support for Formulating and Solving Available Assets Problems

5.1. Coupled Selection of Compatible Concepts/Components

If a system is realized entirely through the use of existing components and/or component assemblies, compatible components for dependent subsystems

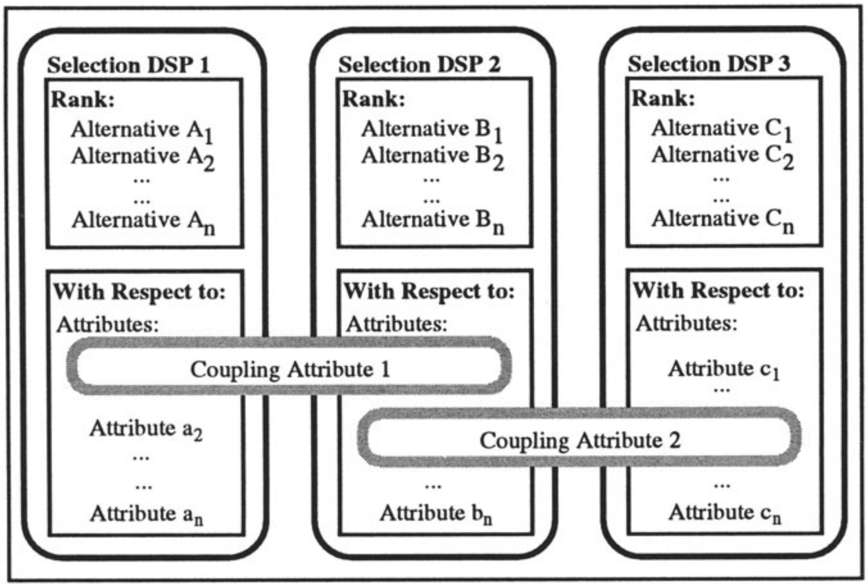


FIGURE 7.13. Illustration of coupled selection DSPs (Bascaran, et al., 1989).

must be selected. The selection of components for individual subsystems will be coupled through compatibility, flows of matter, energy, and information, assembly requirements, and other constraints specific to the problem at hand. The coupled selection DSP formulation is an effective method to realize a physical system configuration through the entire use of available assets.

In Bascaran et al. (1989) a method of formulating and solving coupled selection DSPs in which multiple selection problems are coupled through their dependent attributes (characteristics or criteria that apply to more than one selection problem) and solved simultaneously, is presented. The coupled selection DSP is illustrated in Figure 7.13. The mathematical formulation of the coupled selection DSP, as well as detailed steps for formulating and solving this type of DSP, are include in Bascaran et al. (1989).

5.2. Coupled Selection–Compromise for Detailed Configuration Design

If a system is to be realized through partial use of existing components while the remaining subsystems are designed for manufacture, dependence or coupling between the individual subsystems, both those selected and those designed, will also exist. Again, the selection of components will be coupled through compatibility (assembly constraints, for example) and other con-

Selection DSP 1 (Air Cylinder Type)	**Selection DSP 2** (Slide/Raft Fabric Type)	**Compromise DSP** (Slide/Raft Design)
Find X, e^-, e^+ ***Satisfy*** $\Sigma X = 1$ $MF_1(Y,W) \cdot X + e^- \cdot e^+ = 1$ $0 \leq X \leq 1$ $e^- \cdot e^+ = 0$	***Find*** Y, d^-, d^+ ***Satisfy*** $\Sigma Y = 1$ $MF_2(X,W) \cdot Y + e^- \cdot e = 1$ $0 \leq Y \leq 1$ $d^- \cdot d^+ = 0$	***Find*** W, h^-, h^+ ***Satisfy*** $g(X,Y,W) \geq 0$ $A(X,Y,W) + h^- - h^+ = G$ $W^{min} \leq W \leq W^{max}$ $h^- \cdot h^+ = 0$
Minimize (Lexicographically) $Z = \{f_1(e^-, e^+, d^-, d^+, h^-, h^+), \ldots, f_k(e^-, e^+, d^-, d^+, h^-, h^+)\}$		

X,Y,W - system variables $e^-, e^+, d^-, d^+, h^-, h^+$ - deviation variables (≥0)
g - constraint functions A - goal functions G - goal target values
MF_i - merit function of alternative i
Z - deviation function (Preemptive form, K priority levels)

FIGURE 7.14. Illustration of coupled selection–compromise DSP.

straints, and the original design of the remaining components or subsystems will be dependent on these selections. The selection of existing components may also be dependent on the final design parameters, dimensions for example, of the designed components. For this case the coupled selection–compromise DSP, illustrated in Figure 7.14 for the evacuation system problem, is introduced.

The coupled selection–compromise DSP can support any number of selection problems, coupled through coupling attributes, and components whose parameters are to be determined through compromise. The selection and compromise problems are coupled through the system variables. The parameters for a component to be designed influence the selection for an adjoining component. Likewise, the selection of an existing component influences the outcome of the parameters of a component to be designed.

The aircraft evacuation example has been formulated and solve as a coupled selection–compromise DSP in Koch (1994). Based on the selection of the inflatable slide/raft concept, and pressurized air for inflation, component alternatives were identified for each concept. For this case, then, the necessary selections include the air cylinders for compressed air storage (selection among identified existing types, based on size and pressure characteristics) and the type of fabric for the slide construction (available fabric types

TABLE 7.4. Selection and compromise problems for evacuation system formulation.

Selection	Compromise
Air cylinder type	Number of air cylinders
Fabric type	Dimensions of slide/raft and Inflated pressure of slide/raft

identified; strength, weight, cost influencing selection). The compromise decisions include the number of air cylinders necessary, the dimensions of the side/raft, and the inflated pressure of the slide/raft. These decisions, listed in Table 7.4, are all interdependent. The type of compressed air cylinder selected depends on the inflated pressure of the slide/raft, which is related to the slide/raft dimensions, which is affected by the type of fabric chosen. Therefore, the aircraft evacuation slide problem becomes a coupled selection-compromise DSP.

The framework of the coupled selection–compromise DSP (the word formulation) is formulated as a single compromise DSP, whose framework is as follows (Mistree et al., 1993b):

Given:	*Assumptions of the model*
	Independent system variables
Find:	*The value of the system variables and deviation from goals*
Satisfy:	*System Constraints*
	System goals
	Bounds on the system variables
Minimize:	*A deviation function, expressed as the difference between the target values for the goals and the actual achievement of the goals.*

What are the assumptions inherent in this model? The inflatable slide/raft is modeled as a single rectangular slab, six feet wide and 39.4 feet long (size necessary to meet capacity requirements for sea ditching scenario). Currently evacuation slides are constructed from inflated tubes in a complex configuration (Fisher, 1984); the rectangular slab is used in this model for simplicity. It is also assumed that the stresses in the slide will not stretch the fabric significantly; the affects of fabric elongation are not addressed in slide deflection computations. It is assumed (also for simplicity for this case study) that minimizing the sum of individual component volumes will accomplish the goal of minimizing the system volume. (This will not always be true, depending on the geometry of the components.)

What are the system variables in this DSP? Four alternatives were identified in catalogs for the compressed air cylinder type, and four alternatives were chosen for the fabric type. The alternatives and their attributes are listed in Tables 7.5 and 7.6. Thus eight system variables for the selections exist. The inflated pressure of the slide is also a system variable, as well as the

TABLE 7.5. Cylinder alternatives and attributes.

Cylinder	Max. pressure (psi)	Internal volume (in^3)	External volume (in^3)	Weight (lb)
CYL1	2200	225	290.9	12
CYL2	2015	943	1194.6	40
CYL3	1850	1800	1978.3	18.6
CYL4	3000	40.8	42.9	1.43

TABLE 7.6. Fabric type alternatives and attributes for slide/raft.

Fabric	Variable name	Tensile strength (psi)	Elongation at break (%)	Density (lb/in.^{3})
Kevlar 49	KVLR49	400,000	2.5	0.05202
Kevlar 29	KVLR29	400,000	3.6	0.05202
Nylon	NYLON	143,000	18.3	0.04119
Polyester	POLYST	162,000	14.5	0.04986

TABLE 7.7. Twelve system variables for coupled selection-compromise DSP.

System variable	Description
CYL1	Set equal to 1 if Cylinder 1 selected; 0 otherwise.
CYL2	Set equal to 1 if Cylinder 2 selected; 0 otherwise.
CYL3	Set equal to 1 if Cylinder 3 selected; 0 otherwise.
CYL4	Set equal to 1 if Cylinder 4 selected; 0 otherwise.
KVLR49	Set equal to 1 if Kevlar 49 selected; 0 otherwise.
KVLR29	Set equal to 1 if Kevlar 29 selected; 0 otherwise.
NYLON	Set equal to 1 if nylon selected; 0 otherwise.
POLYST	Set equal to 1 if polyester selected; 0 otherwise.
NUMCYL	The number of air cylinders required (integer).
PRESSR	The inflated pressure of the slide/raft (lb/in.^{2}).
AIRTHK	The thickness of the inflated slide/raft (in.).
FABTHK	The fabric thickness (in.).

number of compressed air cylinders. Since the slide/raft was modeled as a single inflated slab with a fixed length and a fixed width, the remaining dimensions to be varied become the thickness of the air between the fabric layers, and the thickness of the fabric. The list of 12 system variables is given in Table 7.7.

Given these system variables, *what are the constraints and goals of this coupled DSP*? Constraints are used to represent what the design *must* accomplish, and goals are used to represent what a designer would *like* the design to accomplish. Passenger safety is the primary concern. Three constraints were identified as necessary for the system to function properly and safely evacuate passengers: the fabric must withstand the inflated pressure of the slide without bursting, there must be enough compressed air in the cylinders to inflate the slide to its required pressure, and the slide must not deflect (bend under passenger loading) more than a specified value (various limits were explored). Four goals were also identified:

- minimize the overall system weight
- minimize the overall system volume
- keep the slide deflection to a very low value
- and select the fabric with the least amount of elongation at break

As shown in Figure 7.14, the deviation function to be minimized for this coupled selection-compromise DSP is formulated using the preemptive form with K priority levels (five levels for the evacuation system formulation). For the preemptive form, each goal is given a level of priority or importance (level 1 being most important). The normalized deviation from the goal target value for the first priority level is minimized before moving to the second level. The second priority level deviation can then only be reduced without increasing the deviation values for the first level, and so on.

For the evacuation system formulation, the system weight goal is placed in the first priority level, followed by system volume at priority level 2, slide deflection at priority level 3, fabric elongation at priority level 4, and the goal for the value of NUMCYL (the number of cylinders) to be an integer at priority level 5. The constraints and goals for this formulation are identified to represent the requirements of the evacuation system problem statement. The details of this formulation (equations, derivations, and complete model)

TABLE 7.8. Initial Starting Points Run on DSIDES for evacuation system case study.

Variable	Start Pt.1	Start Pt.2	Start Pt.3	Start Pt.4
KVLR29	0	0	1	0
KVLR49	0	0	1	0
NYLON	0	0	1	0
POLYST	0	0	1	1
CYL1	0	0	1	0
CYL2	0	0	1	0
CYL3	0	0	1	1
CYL4	0	0	1	0
NUMCYL	1	1	1	2
PRESSR	20	20	20	20
AIRTHK	4	4	4	4
FABTHK	0.005	0.005	0.005	0.005
Notes:	NUMCYL: integer goal	NUMCYL: integer constraint		Feasible Starting Point

TABLE 7.9. Solutions for different starting points run on DSIDES for evacuation system case study.

Variable	SOLUTION Start Pt.1	SOLUTION Start Pt.2	SOLUTION Start Pt.3	SOLUTION Start Pt.4
Converged?	No	No	Yes	Yes
Feasibility?	No	No	Yes	Yes
Fabric	Nylon	Nylon	Nylon	Polyester
Cylinder	Cyl 4	.2Cyl 3, .8Cyl 4	Cyl 1	Cyl 3
NUMCYL	5	1	1.55	1.06
PRESSR	10.39	10.11	10.02	80.0
AIRTHK	3.129	3.27	3.32	1.63
FABTHK	0.001	0.1455	0.001	0.00125
Deviation Function Values: (1: weight, 2: volume, 3: deflection, 4: elongation, 5: NUMCYL integer)				
Priority Level 1	1.18	0.808	3.30	3.80
Priority Level 2	0	0	0	0.265
Priority Level 3	51.78	47.37	47.04	11.23
Priority Level 4	18.22	18.3	18.3	14.5
Priority Level 5	0.861	—	3.01	1.31

are included in Koch (1994). The results of this example, summarized in the next section, include a feasible component level system configuration that employs available (off-the-shelf) assets to realize the design.

5.3. Representative Results: Feasible Evacuation System Configuration

DSIDES has been used to solve this coupled selection–compromise DSP (Mistree et al., 1993c). Since real variables are employed, the selection variables are constrained to take on values of zero or one. The number of cylinders was also represented with a real variable, and constrained to take on integer values from one to five.

This model was initially run using four different starting points. These starting points are summarized in Table 7.8 and the results from these four runs are shown in Table 7.9 (including the deviation function value at each priority level). The deviation funtion and priority levels are very important concepts. They are explained in detail in Mistree, et al. (1993b). A brief explanation is included to facilitate understanding. A value of zero for the deviation function is indicative that in addition to the system constraints being satisfied all of the system goals have been achieved. A nonzero number represnts the degree by which the goal (or a number of goals) at a particular priority level are achieved. In this case, there is only one goal at each priority level, for example, weight is at priority level 1, volume is at priority level 2 and so forth. For solution Start Pt.1, the target value for weight is not achieved by 1.18 units, the volume is below the target value, etc. Starting points 1 and 2 are identical except the number of cylinders (NUMCYL) is forced to an integer value in the formulation using a goal for starting point 1 and using a constraint for starting point 2 (see Notes, Table 7.8). The integer goal, which was more successful, is used for the remaining points. With all the selection variables initially set to zero (Start Pt. 1), cylinder 4 was selected and NUMCYL (the number of cylinders) was increased to the maximum of five. Additional cylinders would be required to reach feasibility. With all the selection variables set to one (Start Pt. 3, an obviously infeasible starting point since only one cylinder and one fabric can have a value of one), cylinder 3 was selected and feasibility was attained. When starting from a feasible point, (Start Pt. 4, see Notes Table 7.8) the solution improved slightly.

In observing the deviation function values for the first priority level (system weight, see Table 7.9), the best solution results from starting point 3, but this solution has 1.55 cylinders, which is not realizable. If the number of cylinders is rounded to 2, the value for the first priority level changes and the best solution then results from starting point 4. *How good is this solution*? Rounding the number of cylinders to 1.0 maintains feasibility, and the system weighs 22.83 pounds. The system volume is 1.17 cubic feet, and the slide deflection is 12.23 inches, all well within bounds. To verify these results for this simple case, the sixteen possible combinations of the two selections were run separately.

For these additional 16 runs, the selection variables are removed, reducing the number of system variables from 12 to 4. The necessary cylinder and fabric information is fixed in the model for each case. The variable values for the best solution obtained, yielding the lowest value for the first priority level, are listed in Table 7.10. For this run, nylon and cylinder 3 were fixed as the fabric and cylinder choice. *How good is this solution*? The system weighs 21.37 pounds, has a volume of 1.186 cubic feet, and deflects 16.02

TABLE 7.10. Best solution for sixteen combination runs.

Variable	Value
Fabric type	Nylon
Cylinder	Cylinder 3
Number of cylinders (NUMCYL)	1
Air pressure (PRESSR)	99.25
Air thickness (AIRTHK)	1.006
Fabric thickness (FABTHK)	.001

inches. For practical purposes, this solution and the previous solution (starting point 4) are equivalent solutions; both are feasible, both achieved convergence, and the goals values for each are nearly identical. The solution obtained by running all the sixteen combinations of selections separately (listed in Table 7.10) appears slightly better than the solution obtained when a fabric and cylinder type was also to be chosen (including the two selection DSPs).

Why does this solution appear slightly better than the initial solution obtained when including the selection DSPs? Inherent in this model, multiple local optima exist. After selecting an initial fabric and cylinder, feasibility and convergence are achieved; a *satisficing* (Simon, 1982) solution is obtained. By exploring all selection combinations, a slightly better solution is discovered (a global optimum for this formulation).

6. Closing Remarks

Our approach to design as described in this chapter can be characterized by the phrase "more with less"; that is, we are seeking a systematic design methodology that allows new systems to be configured by using that which already exists, available assets. The basis for the conceptual framework for design using available assets presented in Section 2 is the capability to partition the requirements for a design and represent the system in terms of these requirements. The Living Systems Analogy introduced in Section 1 and discussed and illustrated in Section 3 provides the foundation for this decomposition and modeling of information that is necessary to support the efficient realization of system configurations using existing solution principles and existing components.

The capability to identify and evaluate the feasibility and compatibility of existing solution principles and components, however, does not depend only on the consistency and effectiveness of function level modeling. Support for both concept and component level configuration exploration, and the capability to consistently map between function, concept, and component levels is necessary. Some open research issues regarding this conceptual framework for design using available assets include:

- What methods for identifying and exploring existing concepts and components, and assessing feasibility, can be incorporated?
- Is it possible to classify and store existing concepts and components in a manner consistent with the LST representation scheme?
- If so, how could stored information about existing concepts and components be accessed and evaluated against a model of system requirements?

Detailed methods and support for identifying existing concepts and components is essential to the usefulness of this conceptual framework. Catalogs and databases are available to designers to store and retrieve existing components. It may also be possible to store concepts or particular solution principles. The primary open issue then becomes the classification of existing concepts and components that will allow access and evaluation based on the requirements, and will be consistent with the LST scheme. This classification and storing of function, conceptual, and physical component information would allow system configurations to be quickly arranged and explored.

To remain competitive in a rapidly changing global marketplace it is necessary to concurrently reduce time to market, to reduce cost, and to increase quality. The capability to explore system configurations quickly supports this demand. Using available assets to configure systems further supports this demand, as well as supporting the growing concerns regarding recycling and re-manufacture.

Acknowledgments

We gratefully acknowledge support from NSF Grant DDM 93-96052. The information and support provided by Dr. J. A. Shupe and Dr. Sudhir Hublikar of the BF Goodrich Company to develop the evacuation system example is duly recognized and appreciated. P. N. Koch is supported as a Gwaltney Manufacturing Graduate Research Fellow. J. D. Peplinski is an NSF Graduate Research Fellow.

References

Agogino, A. M., and Almgren, A. S. (1987). Techniques for integrating qualitative reasoning and symbolic computation in engineering optimization. *Engineering Optimization*, *12*, 117–135.

Allen, J. K., Simovich, G., and Mistree, F. (1989). Selection under uncertain conditions: A marine application. *Fourth International Symposium on Practical Design of Ships and Mobile Units*, Varna, Bulgaria, Bulgarian Ship Hydrodynamics Centre, pp. 80.1–80.8.

Baas, S. M., and Kwakernaak, H. (1977). Rating and ranking of multiple-aspect alternatives using fuzzy sets. *Automatica*, *13*, 47–58.

Bascaran, E., Bannerot, R. B., and Mistree, F. (1989). Hierarchical selection decision support problems in conceptual design. *Engineering Optimization*, *14*, 207–238.

Bradley, S. R., and Agogino, A. M. (1991). An intelligent real time design methodology for catalog selection. In Stauffer, L. A. (Ed.). *Design Theory and Methodology*, New York: ASME, pp. 201–208.

Bras, B. A., and Mistree, F. (1991). Designing design processes in decision-based concurrent engineering. *SAE Transactions*, *Journal of Materials & Manufacturing*, Warrendale, PA: SAE International, pp. 451–458.

De Boer, S. J. (1989). *Decision Methods and Techniques in Methodical Engineering Design*, De Lier, The Netherlands: Academisch Boeken Centrum.

Fisher, J. M. (1984). Evacuation slide design. U.S. Patent #4,434,870.

Habib, W., and Ward, A. C. (1990). Proving the labeled interval calculus for inferences on catalogs. *Design Theory and Methodology*, New York: ASME, pp. 63–68.

Kahne, S. (1975). A procedure for optimizing development decisions. *Automatica, 11*, 261–269.

Koch, P. N. (1994). Design using available assets: A living systems approach. M.S. Thesis, School of Mechanical Engineeing, Georgia Institute of Technology, Alanta, Georgia.

Koch, P. N., Peplinski, J. D., Allen, J. K., and Mistree, F. (1994). A method of design using available assets: Identifying a feasible system configuration. *Behavioral Science*, *39*(3), 229–250.

Miller, J. G. (1978). *Living Systems*, New York: McGraw-Hill.

Miller, J. G., and Miller, J. L. (1992). *Applications of Living Systems Theory*, New York: Plenum Press.

Miller, J. L. (1990). The timer. *Behavioral Science*, *35*, 164–196.

Mistree, F., Allen, J. K., and Attia, F. (1993a). Designing at a high level of abstraction. *Behavioral Science*, *38*, 124–137.

Mistree, F., Hughes, O. F., and Bras, B. A. (1993b). The compromise decision support problem and the adaptive linear programming algorithm. In Kamat, M. P. (Ed.). *Structural Optimization*: *Status and Promise*, Washington, DC: AIAA, pp. 247–289.

Mistree, F., Smith, W. F., and Bras, B. A. (1993c). A decision-based approach to concurrent engineering. In Paresai, H. R. and Sullivan, W. (Eds.). *Handbook of Concurrent Engineering*, New York: Chapman & Hall, pp. 127–158.

Mistree, F., Kamal, S. Z., and Bras, B. A. (1989). DSIDES: Decision support in the design of engineering systems. Systems Design Laboratory Report, University of Houston, Houston, Texas.

Mistree, F., and Muster, D. (1990). Conceptual models for decision-based concurrent engineering design for the life cycle. In Woods, R. T. (Ed.). *Proceedings of the Second National Symposium on Concurrent Engineering*, Morgantown, West Virginia, pp. 443–467.

Mistree, F., Smith, W. F., Bras, B., Allen, J. K., and Muster, D. (1990). Decision-based design: A contemporary paradigm for ship design. *Transactions*, *Society of Naval Architects and Marine Engineers*, Jersey City, New Jersey, pp. 565–597.

Mittal, S., and Arya, A. (1986). A knowledge-based framework for design. *AAAI*, Los Altos, CA: Morgan Kaufmann, pp. 856–865.

Moore, R. E. (1979). *Methods and Application of Interval Analysis*, Philadelphia, PA: SIAM.

Muster, D., and Mistree, F. (1988). The decision support problem technique in engineering design. *International Journal of Applied Engineering Education*, *4*(1), 23–33.

O'Shaughnessy, K., and Sturges, R. H. (1992). A systematic approach to conceptual engineering design. In Stauffer, L. A. and Taylor, D. L. (Eds.). *Design Theory and Methodology—DTM '92*, New York, ASME, pp. 283–291.

Pahl, G., and Beitz, W. (1988). *Engineering Design*, London/Berlin: The Design Council/Springer-Verlag.

Papalambros, P. Y., and Wilde, D. J. (1988). *Principles of Optimal Design: Modelling and Computation*, New York: Cambridge University Press.

Simon, H. A. (1982). *The Sciences of the Artificial*, Cambridge, MA: The MIT Press.

Swanson, G. A., and Miller, J. G. (1989). *Measurement and Interpretation in Accounting*, New York: Quorum Books.

Thurston, D. L. (1990). Subjective evaluation with multiple attributes. In Rinderle, J. R. (Ed.). *Design Theory and Methodology*, New York: ASME, pp. 355–361.

Vadde, S., Allen, J. K., and Mistree, F. (1992a). Catalog design: Design using available assets. In Hoeltzel, D. A. (Ed.). *Advances in Design Automation*, New York: ASME, pp. 345–354.

Vadde, S., Swadi, S., Allen, J. K., and Mistree, F. (1992b). Design of an aircraft tire: A study in modeling uncertainty. In Hoeltzel, D. A. (Ed.). *ASME Design Automation Conference*, Scottsdale, Arizona, pp. 315–325.

Waldron, K., Waldron, M., and Wang, M. (1986). An expert system for initial bearing selection. *ASME Design Engineering Technical Conference*, 86-DET-125, Columbus, Ohio.

Walker, J. F., and Thiemann, F. C. (1990). The relationship of the internal security system to group level organization in Miller's living systems theory. *Behavioral Science*, *35*, 147–153.

Wilde, D. J. (1978). *Globally Optimal Design*, New York: Wiley.

Wood, K. L. (1990). A Method for the representation and manipulation of uncertainties in preliminary engineering design. Ph.D Dissertation, California Institute of Technology, 1990.

8
Group Decision Making in Design

Deborah L. Thurston

Abstract. Part 1 deals with the problem of balancing conflicting objectives in group decision making. Several methods and their limitations are described, including matrix, voting, ranking, and rating schemes. Two analytic decision tools are presented; multiattribute utility analysis and the analytic hierarchy process. An example illustrates the problem of eliciting and aggregating individual preferences for managing a long range, multiple product design plan and schedule. The group includes engineering, manufacturing, marketing and environmental personnel. Part 2 deals with communication. Based on cognitive models of communication processes and failures, a method is presented for designing not the artifact, but the interdisciplinary design team itself. The objective is to minimize the expected effect of communication failures through failure modes and effects analysis.

1.1. Introduction

A group effort during the design process is both necessary and desirable. First, even the simplest design tasks often require input from more than one technical specialist. Second, recent efforts towards improving the engineering design process stress the importance of considering multiple perspectives, including impact of design decisions on the customer, the manufacturing process, and the environment.

The recent emphasis on "concurrent engineering" illustrates the need to integrate multiple considerations into engineering decision making. In the past, these considerations were ignored until after the design process was completed, at which point the design was thrown "over the wall" to be evaluated. The manufacturing engineer would report that the design was too difficult to form within desired tolerances, the customer would report that the product was too expensive, and environmental regulatory personnel would report an unacceptable waste disposal problem. The engineers were then sent "back to the drawing board" to deal with "the manufacturing problem," "the cost problem," or "the environmental problem." This was obviously a very inefficient and ineffective process.

Design teams now include individuals who bring a diverse range of perspectives to the drawing board. This chapter does not address how to resolve differences of technical opinion such as determining the correct mathematical model of physical phenomenon. Rather, we address group decision making in the context of interdisciplinary design teams and their conflicting preferences. The general group decision-making problem is not a trivial one. The problem consists of two parts: (1) aggregation of individual, conflicting preferences to determine the best decision for the group and (2) communication between group members during the design process.

Part 1. Balancing Conflicting Objectives

1.2. Eliciting Individual Preferences

Concurrent engineering brings together individuals who might not traditionally interact with each other, and who might not view the decision problem in the same way or "speak the same language." These individuals might have a different set of interests in the outcome of the decision. For example, the engineering design group is traditionally most concerned with the quality of the finished product, while the marketing group is more interested in maintaining or increasing market share, while the accounting group is most interested in quarterly profits. These parties have focussed on their own interests and previous experience, and might be unaware of the implications of their decisions on other parties involved in or affected by the design process. Hogarth labels these and other decision making biases "selective perception" (Hogarth, 1980).

Popular tools for managing interdisciplinary design include Quality Function Deployment (QFD) (Sullivan, 1986; Hauser and Clausing, 1988) and Pugh's method (1981, 1990). Both methods employ a matrix to relate design criteria rows to decision element columns. QFD originated in Japan and is used to construct a "House of Quality" matrix. The matrix deploys the "voice of the customer" throughout the design process by providing a structure within which the relationships between engineering design decisions and resulting design performance including customer attributes can be recorded. For example, imagine a "House of Quality" for the problem of designing an automotive bumper beam. The row categories or customer attributes might be (1) reasonable vehicle purchase price, (2) good gas mileage, (3) no repair required after minor impacts, and (4) protection of the car body from minor impacts. Their relative importance is indicated on a scale such as 1–10. The column headings refer to the engineering characteristics such as (1) manufacturing cost, (2) weight, and (3) deflection. Symbols within the matrix and the "roof" indicate the existence of positive or negative relationships (conflicting or non-conflicting) between customer attributes and engineering characteristics, and also between engineering characteristics themselves. For example, an "X" symbol might indicate a "strong negative relationship" (i.e., improvement in one worsens the other) between weight and deflection, and a "/" symbol a "strong positive relationship" between purchase price and manufacturing cost. Design alternatives, including the competitor's, can be compared on the basis of a weighted average of their performance with respect to the target set for each engineering characteristic. Pugh's method uses "+" and "−" signs to indicate performance relative to an alternative selected as the standard.

At this point, the critical decision as to whether a particular tradeoff is beneficial is left to the decision making group. For example, the decision that "benefits outweigh costs" for a particular design change is based on discussion and debate. As Hauser and Clausing (1988) themselves state, "The house relieves no one of the responsibility of making tough decisions. It does provide the means for all participants to debate priorities."

Within the business and management science communities, research on Group Decision Support Systems (GDSS) has been carried out Nunamaker et al. (1988) describe principles for group planning and policy making. Three levels are described by DeSanctis and Gallupe (1987): (1) technology-based systems which remove communication barriers, such as large screens for displaying information, (2) problem structuring techniques, such as project scheduling and multi-criteria decision models, and (3) machine-induced group communication, whereby information exchange is actively controlled and structure.

Several other researchers have addressed group design. Gebala and Eppinger (1991) describe methods for analyzing and managing the design process through a matrix which represents information flow. McMahon (1991) describes the results of a computer based group design system (GDS) used to record the transactions which occur during the group design process. Krishnan et al. (1991) present a method for cooperative group design through the use of a quality loss matrix. While these methods facilitate communication flow, they provide no formal, explicit procedure for using this information to balance individual preferences that are in direct competition.

1.3. Aggregation of Individual Preferences

In engineering design, attributes are often conflicting; an improvement in one leads to a worsening of another. For example, increasing the thickness of an automotive body panel improves stiffness but worsens weight. Not only are the attribute themselves in conflict, but so are the preferences of individuals. Even when two individuals agree that an attribute is important, they often disagree as to its relative importance in relation to other attributes, and differ in their willingness to trade one attribute off against another. These conflicts make it difficult to reach a consensus on the best course of action.

Thus, a group of individuals, each with his or her own set of values and preferences, must identify the alternative that is "best" for the entire group. The problem of combining conflicting individual preferences into a measure of overall group preference has proven to be extremely difficult. The central issue is the difficulty of defining the *criteria* for combining the expressed preferences of individuals to determine the optimal group choice. Several popular and easy to use methods are described in this section, including majority rule voting schemes, individual priority rankings of alternatives, and rating schemes. Their limitations for engineering design are described.

1.3.1. Majority Voting

The simplest method, majority voting, identifies the alternative which receives the most votes as the best alternative for the group as a whole. While this might appear to be a logical and fair approach, its limitation for design is that it identifies only the most preferred alternative for each individual, and provides no information on their preferences among the other alternatives. If the decision criterion is to identify the course of action which results in the greatest level of satisfaction for the group as a whole, one can easily imagine a scenario whereby a simple voting scheme leads to a suboptimal group choice. For example, imagine a group of 10 individuals who must collectively choose between design alternatives A, B, C and D. Their preferences as expressed in a rank ordering are shown in Table 8.1.

The winner in a simple voting scheme is Design A, with the greatest number of votes, four. However, this is the least desirable option for six of the other group members. One could reasonably argue that Design B is a better choice, as it is preferred to A by a majority of six individuals, and is the second choice (out of four design alternatives) of the remaining four individuals. Simple voting schemes do not consider the effect on group members whose first choice is not that of the majority.

TABLE 8.1. Individual preferences of a group of ten individuals.

Number of individuals	Preferences in rank order
four individuals	$A > B > C > D$
three individuals	$B > C > D > A$
three individuals	$C > B > D > A$

TABLE 8.2. Rank order preferences of design alternatives A, B and C by three individuals.

Individual	Preference in rank order
Individual #1	A > B > C
Individual #2	B > C > A
Individual #3	C > A > B

1.3.2. Ranking Schemes

In design, group members often wish to consider the effect of the group decision on members who are not in the majority. One reason is the desire to maintain good-will and a willingness to compromise between group members who will (most likely) continue to interact in the future. After decisions are made for one product, the same individuals, or at least representatives of the same division within the organization, collaborate again on other projects. Therefore, the group has a vested interest in ensuring that all individuals are satisfied with the outcome of the decision making process, even though their first choice is not selected.

Suppose then that we attempt to resolve this difficulty through rank ordering and pairwise voting. It requires each member to indicate their preferences by rank-ordering each alternative. The following example derived from Condorcet (1785) and Arrow (1951) illustrates that this approach can result in a group rank ordering of alternatives that violates the desired property of transitivity. For example, say three individuals have expressed preferences among design alternatives A, B, and C as listed in Table 8.2.

Starting with a pairwise comparison between A and B, A is preferred by a majority of two individuals. Comparing the remaining alternatives A and C, we find that C is preferred by a majority of two individuals. However, comparing B and C reveals that B is preferred to C by a majority of two, yielding an intransitive group rank ordering of C > A > B > C. Of course, this result is highly sensitive to how the voting procedure is carried out. The identification of the best choice depends entirely on the order in which the pairwise comparisons are performed.

1.3.3. Rating Schemes

One attempt to resolve this difficulty is by allowing individuals to express degrees of preference. For the example in Table 8.2, say individual #1 had a very strong preference for design alternative A over alternative C, and individuals #2 and #3 each have only a very weak preference for alternative C over A. To maximize total group satisfaction, the group might select A as the best alternative for the group as a whole.

TABLE 8.3. Individual and total scores for alternatives A, B and C.

	Design A	Design B	Design C
Individual 1	9	5	1
Individual 2	1	5	2
Individual 3	6	5	7
Sum of Scores	16	15	10

Strength of preference can be quantified by rating schemes. For the same group, each individual assigns a score to each alternative on a scale of 1–10 which reflects its worth to that individual, as shown in Table 8.3. One can then define the group decision criterion as the alternative with the greatest total score, summed over all group members, shown in the last row. Design alternative A has the highest total score of 16, followed by alternative B with a score of 15. Alternative C is least desirable with a score of 10. Thus Alternative A might be chosen on the basis that its cumulative score is highest.

However, while this criterion reflects strength of preference for all alternatives, it ignores the issue of fairness or equity. Note that individual #1 very strongly prefers A to C, and less strongly prefers B to C. Thus, the difference between individual #1's scores for the three alternatives is great, ranging from a high of 9 to a low of 1. In contrast, it appears that individual #3 has expressed relatively weak preference for Alternative C over A, and A over B.

Also note that the difference between the three individuals' scores for Alternative A is great, ranging from a high of 9 to a low of 1. If the individual scores are taken to mean the degree of satisfaction of each group member, the selection of Alternative A is not equitable in that Individual #2 is significantly worse-off than Individual #1. In contrast, there is no difference between the three individual scores of "5" for Alternative B. Design alternative B might reasonably be deemed more desirable than A because the *distribution* of scores between individuals is more equitable, and each individual is equally well-off.

The method of summing individual rating scores thus makes no provision for consideration of equitable distribution between individuals, although it implicitly assigns equal importance to each group member's preferences.

1.3.4. Efficiency vs. Equity

This brings us to a critical problem in group decision making; interpersonal comparison of preferences or degrees of satisfaction. Extensive research has been performed in economic analysis of group (or social) welfare. Researchers have shown that it is not possible to develop a mathematical formulation for a group welfare function which maximizes both the sum of individual welfare and equity. A general interpretation of Arrow's Impossibility Theorem (1951) is that given a set of reasonable conditions, there is no procedure for combining the rankings of alternatives by several members of a group into an overall group ranking that does not directly or indirectly include comparison of preferences between individual members. Kirkwood (1979) showed that strictly "efficient" methods, ones which have Pareto optimality or maximization of total welfare as the sole objective, are incompatible with methods which include consideration of equity or "fairness." In group decision making, pareto optimality is achieved when it is not possible to increase the degree of satisfaction of any individual without decreasing the satisfaction of another individual at the same time.

So, the "bad news" is that there is no mathematically sound method for aggregating the preferences of individual group members into a group preference function to determine the optimal or "best" decision for the entire group. The good news is that several well-established methods do exist that are a significant improvement over simple voting, ranking and rating schemes. These methods, described in the next section, help groups attack decision problems in a structured, analytic manner. The problem of efficiency vs. equity is dealt with indirectly by requiring that the group reach consensus on disaggregated components of the decision problem.

1.3.5. Group Decision-Making in the Iterative Design Process

Another weakness of voting, ranking or rating methods is their limited usefulness in the iterative design–evaluate–redesign process. None of the methods provides any information that the designer can use during the redesign stage after the "best" design alternative is identified. The reason is that the individuals are not required to describe their reasons for their voting, ranking, or ratings. The designer does not know what features of a particular alternative make it more desirable than another, and so cannot predict whether or not design modifications would be desirable.

1.4. Analytic Methods for Evaluation of Conflicting Preferences

In spite of the impossibility of formulating a normative group welfare function, two well-established methods do exist that are a significant improvement over the methods described above. These two methods help groups attack decision problems in a rigorous manner; multiattribute utility analysis (MAUA) developed by von Neumann and Morgenstern (1947), Savage (1954), Luce and Raiffa (1957), Keeney and Raiffa (1976), and others, and the analytic hierarchy process (AHP) developed by Saaty (1980).

It is important to note that neither MAUA nor AHP was specifically developed for group decision making. They do not resolve the impossibility of interpersonal comparison of utility, nor do they solve the problem of efficiency vs. equity discussed earlier. For example, using MAUA or AHP to determine individual scores such as those shown in in Table 8.3 would not resolve the difficulty of selecting between design alternative A and B. However, both methods contribute significantly to group decision-making in design by providing a forum for communication, a structured procedure for eliciting their preferences, a rigorous methodology for converting their expressions of preference into a quantitative measure of the relative merits of design alternatives, and a framework for analyzing possible courses of action and negotiating tradeoffs during the iterative design process.

1.4.1. Analysis of Design Decision Problem

Both MAUA and AHP solve a complex decision problem by disaggregating it into a set of subproblems, solving or reaching group consensus on each one, then reassembling them to obtain a solution to the larger decision problem. A correct and thorough application of either method to design problems has the following features:

0. *Overall goal*—An overall goal of the design process is established, such as "Design the best vehicle in the domestic mid-size sedan market segment."
1. *Decision makers*—A clear identification of the individuals or perspectives to be included in the group is made.
2. *Design alternatives*—A set of design alternatives is developed, most often from previous, similar applications.
3. *Multiple attributes*—Multiple attributes, subgoals or decision outcomes that are deemed to be important to one or more individuals are enumerated.
4. *Constraints*—A clear distinction is made between attributes with a defined range of acceptability, and those which are binary constraints. For example a deflection *constraint* would be "The beam MUST deflect EXACTLY 2 inches in response to load F in order to be considered feasible." The deflection *attribute* would be "We are willing to consider alternatives which deflect between 1 and 3 inches in response to load F."
5. *Relative value of achieving levels within each subgoal*—A procedure exists which can be used to model non-linearity of preference over the acceptable attribute range, if necessary. For example, the benefit gained from improving deflection from 3 to 2 inches might not be the same as that from improving deflection from 2 to 1 inch. In utility analysis, this is reflected in the degree of non-linearity over the single attribute utility

function for deflection. In AHP it is reflected in the relative priority of achieving various levels within each subgoal.

6. *Tradeoffs*—A measure of the relative value or importance of attributes, or a measure of the decision-makers' willingness to make tradeoffs between attributes is made. In utility analysis, tradeoffs are reflected in the scaling constants, and in AHP they are reflected in the relative priority or importance of each subgoal.
7. *Overall worth*—A mathematical procedure is carried out for combining the information obtained through features 1–5 above into a single numerical quantity which represents the overall utility or worth of a particular decision alternative.

1.4.2. Multiattribute Utility Analysis

Multiattribute utility analysis determines the worth of a design as a *combination* of attributes. Thurston (1991) describes how to formulate a multiattribute design evaluation problem and use the results to quantify beneficial attribute tradeoffs. Conflicting design attributes should be defined in such a way as to accurately reflect preferences while exploiting conditions of preferential and utility independence. These conditions do *not* refer to independence between the attribute levels but rather to the relative *worth* a designer places on individual attribute levels. For example, total manufacturing cost is clearly related to and dependent on weight, but the utility independence condition is satisfied if the relative worth to the designer over the range of acceptable levels of weight *alone* is independent of cost. This means that the general shape or degree of nonlinearity of the utility function is not altered by changes in levels of another attribute Y. The less restrictive preferential independence condition means that if a lower weight design is preferred to a higher weight design when the cost for both is Y1, then the lower weight design will still be preferred to the higher weight design *when the cost for both is some other value Y2*. These conditions are easily satisfied for design problems with proper definition of the attributes and their range of acceptability.

If the independence conditions are tested and satisfied, the number of preference statements required in the assessment procedure is minimized. In addition, the multiplicative multiattribute utility function in equation (1) is valid [Keeney and Raiffa (1976, pp. 290–291)], which permits calculation of a measure of the overall worth of a design, $U(X)$, ass a function of a set or combination of performance attributes:

$$U(X) = \frac{1}{K}\left[\left[\prod_{i=1}^{n} (Kk_i U_i(x_i) + 1)\right] - 1\right], \qquad (8.1)$$

where

$$U(X) = \text{overall utility of set of attributes } X$$
$$x_i = \text{performance level of attribute } i$$
$$X = \text{set of attributes at levels } (x_1, x_2, \ldots, x_n)$$
$$k_i = \text{assessed single attribute scaling constant}$$
$$U_i(x_i) = \text{assessed single attribute utility function}$$
$$i = 1, 2, \ldots, n \text{ attributes}$$
$$K = \text{scaling constant.}$$

The constant K is obtained by normalizing $U(X)$ between 0 and 1 in the standard way:

$$1 + K = \prod_{i=1}^{n} (1 + Kk_i). \tag{8.2}$$

where the more restrictive additive independence condition described by Fishburn (1965) is also satisfied, the scaling constants k_i sum to 1 and the utility function reduces to the additive form:

$$U(X) = \sum_{i=1}^{n} k_i U_i(x_i). \tag{8.3}$$

Each assessed single attribute utility function $U_i(x_i)$ is scaled so that where the attribute is at its worst (but acceptable) level, $U_i(x_{iw}) = 0$, and at it best level, $U_i(x_{ib}) = 1$. They can reflect nonlinearity of preference over the acceptable attribute range. The scaling constants k_i reflect the acceptable tradeoffs between attributes and, combined with K, scale $U(X)$ between 0 and 1.

Figure 8.1 shows two standard "lottery" questions to determine k_i and $U_i(x_i)$ for weight. The decision makers are asked to imagine two alternative designs, each alike in every respect except the attribute levels of the alterna-

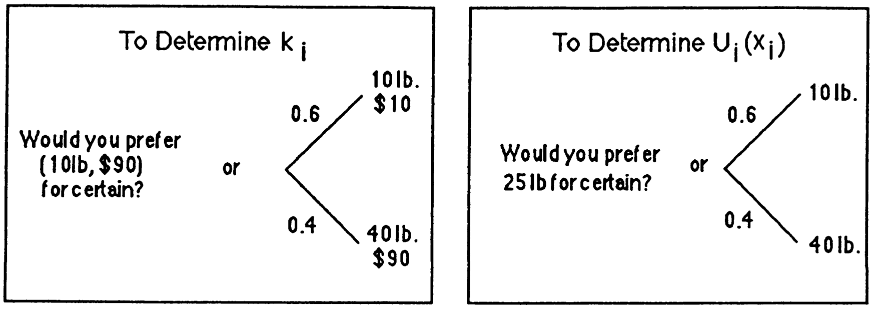


FIGURE 8.1. Lotteries to assess scaling constant k_i and utility function $U_i(x_i)$ for weight.

design in which there is uncertainty as to the attribute level(s). To determine k_i, subjects are queried as to whether they prefer 10 lb at a cost of \$90 for certain, or a 60% probability p that weight will be 10 lb and cost will be \$10, with a complementary probability $(1-p)$ of 40% that weight will be 40 lb and cost will be \$90. The value of p at which the subjects are indifferent between the "certain alternative" on the left and the uncertainty on the right is obtained by iteration between extreme values of p. The multiattribute utility when all attributes are at their best levels, $U(X_b)$, is set equal to 1, and where they are at their worst levels, $U(X_w)$, set equal to 0. By definition, the value of k_i is equal to the multiattribute utility where x_i is at its best level, x_{ib}, and all of the other attributes are at their worst levels. Since $U(x_{1w},\ldots,x_{ib},\ldots,x_{nw}) = k_i$, the value of k_i is determined by

$$
\begin{aligned}
U(x_{1w},\ldots,x_{ib},\ldots,x_{nw}) &= pU(X_b) + (1-p)U(X_w),\\
U(x_{1w},\ldots,x_{ib},\ldots,x_{nw}) &= pU(10\text{ lb}, \$10) + (1-p)U(40\text{ lb}, \$90),\\
U(10\text{ lb}, \$90) &= p(1) + (1-p)(0),\\
k_i &= p.
\end{aligned}
\tag{8.4}
$$

To determine the single attribute utility function $U_i(x_i)$ for weight, subjects are queried as to whether they prefer 25 lb for certain, or a 60% probability p that weight will be 10 lb and probability $(1-p)$ of 40% that weight will be 40 lb. When the probability p at which the subject is indifferent between these two choices is determined, the utility of 25 lb, which represents one point on the single attribute utility function, can be calculated:

$$
\begin{aligned}
U_i(x_i) &= pU_i(x_{ib}) + (1-p)U_i(x_{iw}),\\
U_i(x_i) &= pU_i(10\text{ lb}) + (1-p)U_i(40\text{ lb}),\\
U_i(25\text{ lb}) &= p(1) + (1-p)(0),\\
U_i(25\text{ lb}) &= p.
\end{aligned}
\tag{8.5}
$$

The conventional mechanical engineering design approach to multiple attribute design evaluation is to determine a "Figure of Merit" (FOM), which is essentially a weighted sum of each attribute level a design alternative exhibits. Each attribute's weighting factor is intended to reflect its relative importance. Utility analysis is superior to simple linear weighted average methods, as demonstrated by Thurston (1991). Utility analysis can more accurately measure designers' preferences over an acceptable attribute range, and allows for the possibility that preferences might not be linear with respect to attribute level. As a result, it can more accurately reflect the trade-offs between attributes the decision maker is willing to make. The FOM approach assumes linear preferences with respect to attribute levels and constant trade-offs, and can lead to erroneous results when used to rank alternatives and quantify desirable attribute tradeoffs.

Another advantage of utility analysis is its ability to explicitly model the decision-maker's attitude towards risk and uncertainty, and include the effect of uncertainty on the desirability of design alternatives. Decision aids for design that include manufacturing cost estimation uncertainty in calculations of expected design utility are found in Thurston and Liu (1991). Utility analysis is compared to fuzzy set analysis for design evaluation in Thurston and Carnahan (1990). They conclude that fuzzy set analysis is more appropriate at the very earliest stages of preliminary design, while utility analysis should be used as the design progresses to the stage where tradeoffs are to be evaluated.

1.4.3. Analytic Hierarchy Process

The Analytic Hierarchy Process (AHP) was developed by Saaty (1980). Compared with utility analysis, AHP's distinguishing features are that it structures the decision problem as a *hierarchy* of goals which each contribute to some overall goal (although hierarchies can also be used in utility theory), elicits preferences through qualitative pairwise comparisons, and uses the eigenvector of the pairwise comparison matrix to determine overall priorities.

A five-point (1, 3, 5, 7, 9) scale is used the describe intensity of relative importance. The decision-makers' priorities are elicited through indications of the relative importance of achieving each subgoal through a set of direct pairwise comparisons. For example, comparing profit to quality, 1 indicates equal importance, 3 indicates moderate importance of one over another, through 9 indicating extreme importance of one over another. Even numbers (2, 4, 6, 8) indicate intermediate values between the two adjacent judgments. Reciprocals of these numbers indicate reversal of relative importance. Pairwise comparisons are elicited for each possible pair at each level in the hierarchy. This set of pairwise comparisons forms a matrix shown later in Figure 8.5. Then, each design alternative is specified in terms of its impact on each of the sub-goals. The principal eigenvector is computed and used to provide a priority ordering of alternatives according to dominance, while the eigenvalue provides a measure of consistency of responses to the pairwise comparisons.

1.4.4. AHP vs. MAUA

There is a long-running and sometimes heated debate over the relative merits of classical utility analysis and AHP. Both camps have their dedicated followers. The literature is extensive, but summarizing positions have have been put forth by Dyer (1990a, 1990b), Saaty (1990), Harker and Vargas (1990), French (1986, pp. 357–361), Watson and Buede (1987). Summarizing this debate, much less trying to resolve it, is beyond the scope of this chapter. However, it is fair to say that the central issue for engineering design is disagreement over the proper degree of normative vs. descriptive roles of analytic decision-making tools. Some researchers have noted that utility analysis can sometimes fail as a descriptive decision aid under certain circumstances. That is, it does not always mimic the steps of the current decision making process nor accurately predict the decision maker's unaided choice. One example is "real-time" or emergency situations (Klein and Calderwood, 1991).

The criticism of AHP most relevant to engineering design is in regards to the assumption that qualitative indications of the "relative importance" of two attributes can be interpreted on a numeric scale of 1 to 9 to determine the ratio of the weights of the two attributes. For example, the statement that "weight is moderately more important than cost" is interpreted to imply that $w_{\text{weight}}/w_{\text{cost}} = 3$. This in turn implies that one unit of weight (say 1 lb) is three times as significant as one unit of cost ($). This is an especially important consideration for engineering design, when designer might wish to use the results of the analysis to calculate beneficial tradeoffs to guide the iterative design process. While AHP can contribute significantly by disaggregating a decision problem to enable consensus reaching for group negotiations, designers should be *extremely careful* if they wish to use the results to quantify beneficial tradeoffs.

However, judging a decision-aiding tool by whether or not it accurately describes the actual behavior and choices of human beings is not the appropriate question. Rather, one should ask "Does the tool help designers make better decisions?" If humans were always successful in instinctively making optimal decisions on their own, there would be no need for formal decision theory. The emergence of "design theory and methodology" as an important area of research is evidence that designers are indeed *not* satisfied with their current design decision-making methods.

1.4.5. Implementation Issues—Benevolent Dictator and Negotiation and Consensus

As mentioned earlier, neither MAUA nor AHP explicitly solves the problem of efficiency vs. equity. In practice, one of two approaches is commonly taken; assigning the "benevolent dictator" role to a group leader, or group negotiation and consensus. The first approach assigns the task of aggregating individual preferences to a third party such as a decision analyst or group leader who assumes a "benevolent dictator" role. The leader is benevolent in that he or she seeks to identify the course of action that is in some way the overall best choice for the group as a whole, and is not biased for or against any individual. The leader is a dictator, in that while individual preferences are elicited and used in the decision-making process, no group member(s) has veto power.

In group negotiation and consensus, group members first agree on the decision making procedure to be used. By disaggregating the decision problem and focussing attention on manageable sub-problems, either method provides an excellent forum for communication between individuals within a group and for enabling consensus-reaching. After consensus is reached on each sub-task, each method provides an explicit procedure for combining those results to compare each design alternative on the basis of its total relative merit to the group as a whole. By engaging individuals in the steps required for formulating the decision problem, practitioners have found it much easier to achieve consensus. For example, Sycara and Lewis (1991) use utility analysis as the basis of a method for negotiation in product design.

1.5. AHP Example: Group Decision-Making for Design Project Scheduling

1.5.1. Long-Range Plan for Project Scheduling

This section summarizes an application of AHP to group decision making in design project scheduling, where designers are only one part of a group comprised of design engineering, manufacturing, marketing, accounting and other sectors of the organization, as described by Thurston and Tian (1990). Deciding which family of products to design and when to introduce them to the marketplace is extremely important. This task is a multidisciplinary team effort, requiring the input of diverse parts of the corporation which have distinct and sometimes conflicting priorities.

The automotive, consumer electronics and other industries introduce new product features each year. These features are often modifications or additions to existing base products. In the automotive industry, even minor product modifications require large capital expenditures and several years of coordinated effort on the part of large engineering design teams. Each team consists of subgroups, such as powertrain or electrical system specialists. Orchestrating the engineering design process for each engineering specialty, for each planned modification, and for each product is a complex scheduling task. A long-range plan (LRP) specifies a schedule for product design, allocating manpower and capital resources to each project to ensure completion by the planned launch date.

The long-range plan must often be changed in response to the marketplace or actions by the competition. However, moving the launch date up on one design project can require delays in other projects due to manpower and capital resource constraints. This has different impacts on different parts of the organization. Launching a product earlier might prevent the engineering design group from achieving a quality goal. However, the marketing group might prefer to introduce the product as-is in order not to lose market share; they know from historical data that once market share is lost, it is extremely difficult to regain. In addition, a delay in a low-mileage, high volume vehicle might prevent compliance with regulatory corporate average fuel economy (CAFE) standards.

Design project managers need a method by which they can facilitate communication across disciplinary boundaries within the organization. However, it is not enough just to break down communication barriers so that the interests and preferences of each sector of the organization can be known. There is also a need for a method to integrate this information into design project management in a meaningful and fruitful way.

1.5.2. Hierarchy

The overall goal here is to minimize the total detrimental impact of necessary changes to the long range product design plan. The hierarchy is shown in Figure 8.2. Each of the six major factors represents the primary interests of one part of the group, who have not traditionally worked closely together at the decision making level. The reason that each major factor occupies the same (first) hierarchical level is that the corporation is trying to break down traditional organizational barriers between these groups.

The structure is intended to stimulate input and encourage interaction between groups by making it clear that no group is "more important" than another. Members of these groups would be sensitive to implications of a "chain of command" if their major factor, such as "regulatory compliance,"

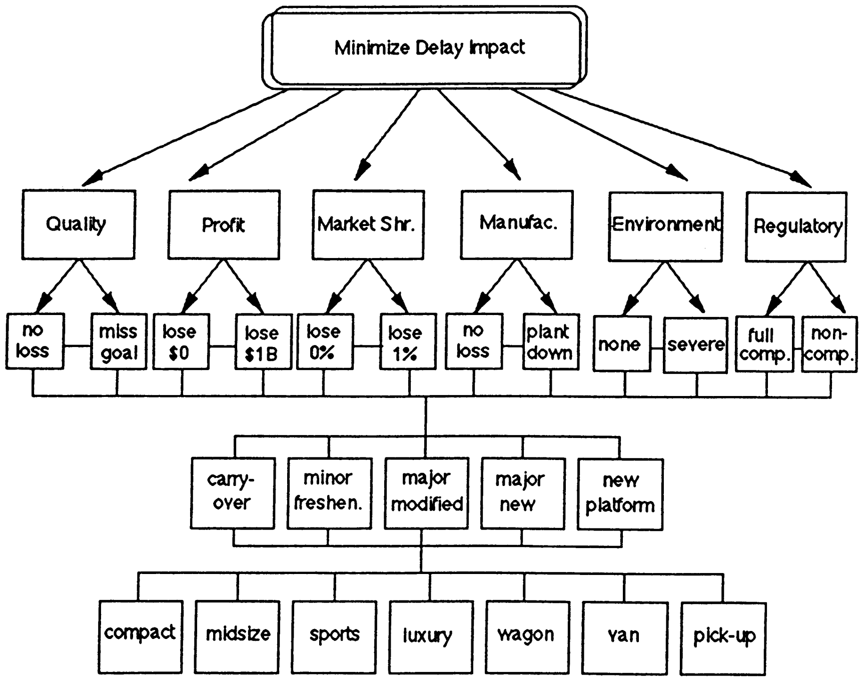


FIGURE 8.2. AHP hierarchy for design project delay impacts.

	Quality	Profit	Mkt Shr.	Manufg.	Environ.	Regulat.
Quality	1	2	5	6	7	9
Profit	1/2	1	2	5	7	8
Mkt Shr.	1/5	1/2	1	2	3	5
Manufg.	1/6	1/5	1/2	1	2	3
Environ.	1/7	1/7	1/3	1/2	1	2
Regulat.	1/9	1/8	1/5	1/3	1/2	1

FIGURE 8.3. Pairwise AHP comparison matrix.

appeared in a lower level of the hierarchy. In order to gather input from each group literally on its own terms, impacts were not converted to some common metric such as indirect cost. Figure 8.3 shows the pairwise comparison matrix. For example, market share loss was deemed to be "moderately more important" than environmental impact, so a value of 3 was entered in row 3, column 5 of the matrix. The following factors reflect the major interests in the group decision-making process:

1. *Quality*—Design engineers are most concerned with the overall quality of the product they produce. The time and resources allocated to the design task determine the degree to which they are able to achieve their goals. These goals might be in the area of using the Taguchi method to develop robust designs which are affected as little as possible by uncontrollable deviations in the manufacturing and assembly process, or in improving component tolerancing and "fit and finish" after the assembly process.

2. *Profit Loss*—Accounting is most concerned with the expected loss in short term profit due to delaying the launch date of a particular project by one time period, or 1/2 year. Units of measurement for expected profit loss are dollars, ranging from a worst case scenario of 1 billion dollars to a best case scenario of no profit loss due to the ability to continue sales of the "old" product.

3. *Market Share Loss*—Marketing is more concerned with longer term impacts; uninterrupted product availability can be an important factor in maintaining corporate image in a market segment and customer brand loyalty. Being "out of market" for even a short time can have a long term detrimental effect on the overall market share for that model. The detrimental effect of a project delay on market share is defined as the expected loss in market share for that particular model due to a 1/2 year launch date delay. Units of measurement are percentage of total market share, ranging from a worst case loss of 1% of the total market share to a best case of no market share loss.

4. *Manufacturing Impact*—Manufacturing is concerned with production

plant capacity utilization. The worst case scenario is a total plant shutdown, and the best case is no negative effect on plant utilization. A "minor effect" is characterized as one or more of the following: some layoffs, some downtime, and/or a 2 week shutdown for inventory adjustment. A "moderate effect" is characterized by a layoff of 1 shift, a 2 week downtime period, the addition of robots, and/or a small facility readjustment. A "major effect" is characterized by a 25% decrease in capacity utilization, up to a 12 week downtime period, or the addition of a second body shop.

5. *Environmental Impact*—This attribute includes the environmental impact throughout the product's manufacture, consumer-use and disposal life-cycle. The manufacturing process generates byproducts that are harmful to the environment, such as air emissions, wastewater, and solid, toxic and hazardous waste. During the useful life of the product, more waste might be generated during operation, such as vehicle emissions. At the end of the useful life of the product, it must be disposed of.

6. *Regulatory Compliance*—The regulatory group is concerned with satisfying a diverse set of federal regulations for both individual vehicles and the entire fleet. These include Corporate Average Fuel Economy (CAFE) standards, Federal Motor Vehicle Safety Standards (FMVSS) and vehicle emissions standards. Noncompliance can result in fines and/or harm to corporate image. While the corporation does not deliberately plan to violate regulatory requirements, non-compliance might occur due to unforeseen circumstances. With this category in the hierarchy, the impact of potential noncompliance is included in the group decision-making process.

1.5.3. Integrating AHP Results into Design Project Scheduling

The AHP analysis results in a score for each project that reflects the preferences and priorities of each member of the decision-making group. The higher the score, the greater the overall detrimental impact to the group that would result from a delay of the desired launch date of that product. The project scheduling problem is formulated as an integer program with linear constraints. The objective function is the minimization of the detrimental impact of all project delays as shown in equation (8.6). The AHP scores serve as the objective function coefficients C_{ij}, the total overall group impact of delaying the launch date of project j for body style i for one time period. Binary decision variables (D_{ij1}, D_{ij2}) permit the option of delaying any project j for any body style i for either one or two time periods. $(D_{ij1}, D_{ij2}) = (0,0)$ corresponds Ko no delay, $(1,0)$ corresponds to a 1 time period delay, and $(1,1)$ corresponds to a 2 time period delay. See Thurston and Tian (1990) for details on formulation of the constraints on manpower and capital resources in each engineering group:

$$\text{minimize} \quad \sum_{i=1}^{z} \sum_{j=1}^{n} C_{ij}(D_{ij1} + D_{ij2}), \tag{8.6}$$

where

C_{ij} = delay index; the overall impact of delaying the launch data of project j for body style i for one time period

$D_{ij1} = 0$ if project j for body i is not delayed first time period

$= 1$ if project j for body i is delayed first time period

$D_{ij2} = 0$ if project j for body i is not delayed second time period

$= 1$ if project j for body i is delayed second time period

and

$i = 1, 2, \ldots, z$ body styles

$j = 1, 2, \ldots, n$ proposed project for a particular body style

1.6. Discussion

This section has described a tool for group decision-making across traditional disciplinary boundaries. AHP provides a forum for design engineers to interact directly with personnel from marketing, accounting and other parts of the organization. The analysis is structured to reflect issues of concern to each constituency, and permit the direct use of terminology normally used to discuss possible ramifications of delaying particular projects. Participants gain a better appreciation for each other's priorities and concerns, and are better able to reach consensus. Design quality is considered alongside profit margin, market share and environmental impact.

This is not intended to be a descriptive decision-aiding tool. Our goal is not to replicate past choices; a primary motivation for the project is that the decision-makers are not satisfied with the results of their ad hoc or nonexistent group decision-making procedures, and do not wish to replicate them. AHP allows us to determine the diverse effects of project delays, and for the first time quantify them with a single parameter. Without AHP, the modeler might be forced to assume that the impact of delaying a project is the same for all vehicle types and project categories.

Part 2. A Communications-Based Method for Design Team Management

2.1. Communication in Interdisciplinary Engineering Design Groups

It has been estimated that 50% to nearly 100% of design failures are attributable to commonly understood mechanisms (Marriott and Miller, 1981).

This suggests that many failures are due not to a lack of expertise in the design team, but rather to nonrepresentation of existent expertise at key decision points. Interdisciplinary teams are desirable because of *resource additivity*, where group performance increases with the sum of combined member abilities (Hill, 1982; Shaw, 1976). However, barriers to coordination (Allen, 1986) of a pool of specialists exist. Nonadditivity has been explored by Steiner (1972) and Hackman and Morris (1983) under the label *process loss*. Steiner proposed that process loss always occurs in real groups, and stems from informational, behavioral, and organizational factors.

The design team manager defines, assigns, and coordinates tasks (Thamhain, 1983). *Task definition* partitions the project into a number of simpler subtasks. *Task assignment* is the appointment of individuals or subgroups to each subtask. *Task coordination* is the management of the exchange of information to ensure that the necessary design information is represented at the proper decision points. Coordination can become exceedingly difficult in an interdisciplinary project. The manager may have only limited ability to understand, communicate with, or control personnel of diverse expertise and rank (Thamhain, 1977, 1983). Faced with the difficulty of communication and control *during* the design process, the manager should "design in" a resistance to coordination difficulties during the Task Definition and Assignment (TDA) stage *before* the design process begins.

On what criteria should the manager make TDA decisions? Engineering design has been shown to be more susceptible to communication variables than research or technical service efforts (Allen, et al. 1980). Task achievement is related to communication, particularly in complex tasks (Bales and Slater, 1955; Hare, 1976). The TDA imposed on a group determine effectiveness of communication, and hence the effectiveness of the design team. The nature of the task affects the communication skills required to solve the problem (Barge and Hiokawa, 1989; Fiedler, 1967). The modes of communication required for problem solving will affect the group's ability to communicate (Carzo, 1963; Chapanis, et al., 1972). Changes in the information to be communicated has been shown to influence choice of communication channels (Wolek, 1970), which display varying probabilities of communication error (Chakrabarti et al., 1983; Mehrabian and Reed, 1968).

Traditionally, team building techniques have addressed either the characteristics of the artifact, or the psychological needs of design personnel, but not both. A common artifact basis is the division of tasks for minimum interdependence (Steward, 1981). However, full independence of tasks is seldom possible (Finger and Dixon, 1989), and this in reality is an indirect attempt at evading communication rather than treating it directly. Tools that attend to personnel needs (Kiggundu, 1983; Roby and Lanzetta, 1956) are aimed toward optimization of job satisfaction, motivation, or other personnel-related variables. Some investigators have noted that design techniques that are successful in continuous solution spaces fail at disciplinary discontinuities (Ullman, 1989), presumably because of coordination difficulties at interdisciplinary boundaries. For this reason some tools are domain specific, limiting their usefulness across interdisciplinary projects.

The remainder of this chapter presents a tool for managing interdisciplinary engineering design teams which is communications-based. A cognitive model for design communication is presented which is used to develop strategies for defining and assigning design tasks. These strategies are then incorporated into an extension to Failure Mode Effects Analysis (FMEA) to minimize the effect of communication errors in the design process. An example of the design of a program for computer-assisted instruction is presented.

2.2. Cognitive Model of Engineering Design Communication

2.2.1. *Lexicon*

Cognitive models of design describe the mental processes of the designer (Finger and Dixon, 1989; Perlman, 1989). Here we define a lexicon of design information processing which provides the basis for our cognitive model. Figure 8.4 illustrates the following concepts. Design is an activity of information processing for the purpose of decision making:

Engineering design is a process performed by humans aided by technical means through which information in the form of requirements is converted into *information* in the form of descriptions of technical systems (Eder, 1989).

The design process is essentially an information processing activity, usually undertaken by a team of people, with its progression depending on the decisions made (Wallace and Hales, 1989).

Information is knowledge of any design-related objects or events and their relationships that is pertinent to the making of any design decision (Eder, 1989; Stomph-Blessing, 1989). Thamhain (1983) used the term *technical expertise*:

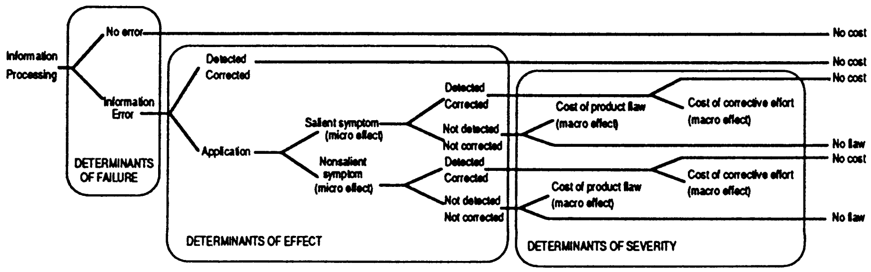

Information Processing
No error
Information Error
DETERMINANTS OF FAILURE
Detected Corrected
Application
Salient symptom (micro effect)
Nonsalient symptom (micro effect)
Detected Corrected
Not detected Not corrected
Detected Corrected
Not detected Not corrected
DETERMINANTS OF EFFECT
Cost of product flaw (macro effect)
Cost of corrective effort (macro effect)
Cost of product flaw (macro effect)
Cost of corrective effort (macro effect)
DETERMINANTS OF SEVERITY
No cost
No cost
No cost
No flaw
No cost
No flaw

FIGURE 8.4. Detailed model of design information processing.

Technical expertise ... includes an understanding of the technology and underlying concepts, theories, and principles, the design methods and techniques, and the functioning and interrelationship of the various components which make up the total system.

... It is necessary not only for proper analytical and development work, but equally important for evaluating technical solutions and tradeoffs, to communicate effectively within the engineering team, assess risks, participate in search of integrated solutions, and make tradeoffs among various alternatives.

The information sphere is where information resides. Spheres can include fields of physical knowledge, experience, and cognitive fields (skills), or any other knowledge source. For example, mechanical engineers use primarily visual and spatial reasoning in conceptualizing their designs (Earle, 1985).

Information processing is the generation, access, transfer, and application of information to the making of decisions. It consists largely of transferring information between information spheres and decision points.

Communication is the exchange of information between people, defined by the presence of an encoder (sender), a decoder (receiver), a channel (means of transmission), a referent (topic), and a message (Mehrabian and Reed, 1968).

Information availability does not refer to the *existence* of information in the team, but to the ability of decision makers to *retrieve* it. Engineers waste large amounts of time tracking down design information, a search which may inflate design costs. Some personnel may simply proceed without adequate information rather than expend the effort to find it (Liker and Hancock, 1986).

Information error is erroneous or incomplete information at a decision point.

Availability and communication errors are errors that arise from information unavailability. Some examples are the use of tenuous information due to anticipation of difficulty in obtaining more complete information, or the lack of important information because the decision maker did not know it to exist or know its significance. A *communication error* is the event in which the decoded meaning does not match the encoded meaning.

Information error effects. We distinguish between the occurrence of an information error and the resulting effect:

- *Micro Information Error Effect*—the temporary, unexpected symptom that serves to flag the error *during* the design process, and thereby prompts corrective effort. An example is the nonfit of a prototype part that was dimensioned incorrectly.
- *Macro Information Error Effect*—the effect on either the design process or the artifact, expressed through two types of corrective action cost (Ireson and Coombs, 1988):
- *Corrective Effort*—If a micro error effect occurs, its correction entails the expense of corrective effort. For example, if a transposed digit in the value

of pi is not discovered until after it has been used to build a model of a circular gear, the macro error effect is the cost of rebuilding the model correctly. The cost of corrective effort during early design of concept, development, and pre-production are relatively small.

- *Product Flaw*—If an information error did not cause a visible micro effect during design, or it went uncorrected, a product flaw might emerge. For example, inadequate information about ergonomic requirements might result in a passenger compartment with insufficient leg-room. Costs associated with product flaws include redesign, erosion of product reputation, etc. The cost of correcting product flaws increases exponentially during the later design stages of production and field service.

2.3. Task Definition and Assignment (TDA) Strategies

This section presents strategies for defining and assigning design tasks to improve communication. They reduce the risk associated with information errors by controlling communication accuracy, information availability, error detection and corrective effort.

2.3.1. Control of Communication Accuracy

Mehrabian and Reed (1968) proposed hypotheses that relate communication accuracy to attributes of the communicator, addressee, channel, message, and referent. We describe them here, along with corresponding TDA strategies.

Decentering Hypothesis: Communication accuracy is directly correlated with the communicator's or addressee's ability to explain their body of knowledge in "layman's term." *Strategy*: Assign critical tasks to members who have displayed decentering ability, or define critical tasks so as not to require communication between dissimilar fields

Cognitive Similarity: Accuracy is correlated with the degree of similarity between the communicator's and addressee's coding rules and modes of cognition (Rinkel, 1959). *Strategy*: Assign critical tasks to team members who are cognitively similar.

Rate Hypothesis: Accuracy is inversely correlated with the rate of information processing. *Strategy*: Assign critical task to team members with similar information processing capacities, or define individual tasks so as not to require excessively high information processing rates.

Rate Control Hypothesis: Accuracy is correlated with the degree to which the rate of transmission can be modified by the decoder. *Strategy*: Assign a critical task so that communication rates are likely to be controllable by the decoder (e.g., interpersonal communication).

Channel Availability Hypothesis: Accuracy in decoding increases with the degree to which all of the communication channels used by the encoder ate made available to the decoder. *Strategy*: Assign a critical task to individuals who have equal access to available communication channels.

Habitual Channel Hypothesis: Accuracy is correlated with the degree to which the communication channels typically employed by the encoder for that kind of communication are available. *Strategy*: Assign tasks to individuals who habitually use similar channels.

Feedback Hypothesis: Accuracy is correlated with the degree of feedback available to the encoder. *Strategy*: Assign critical tasks for maximal feedback in information exchange.

Message Complexity Hypothesis: Message accuracy is inversely correlated with the degree of complexity of the message. *Strategy*: Define critical tasks in such a way that messages between individuals will be minimally complex.

Organization Hypothesis: The accuracy of a message is directly correlated with its degree of organization. *Strategy*: Assign critical tasks such that organization of information is improved, or define a critical task so that information that is likely to be communicated is highly organized.

Objectivity Hypothesis: Accuracy is directly correlated with the degree of objectivity of the message. *Strategy*: Define and assign critical tasks so that communicated information is objective, leaving exchange of subjective information to occur between cognitively similar members.

Ambiguity Hypothesis: Accuracy is directly correlated with the degree to which the coding rules for the referent are well defined. *Strategy*: Define and assign tasks in such a way that referents that are likely to require communication are concrete and unambiguous.

Referent Complexity Hypothesis: Accuracy is inversely correlated with the complexity of the referent. *Strategy*: Define and assign tasks in such a way that only simple referents are likely to be involved in communication events.

2.3.2. Control of Information Availability

Information availability depends on the relationship between the location of the information and location of the requestor.

Member Centrality. Centrality (Carzo, 1963, p. 400) is indicated by the total number of communication links needed to interact with all other members (Bavelas, 1948, 1950), and is a convenient indicator of the member's information availability (Leavitt, 1951). The lower the number of links, the more available the information possessed by other group members. Groups which have the fewest communication links between the point at which information is received and the actual decision point exhibit the best performance (Roby and Lanzetta, 1956). One might assign a critical task to a person in a highly central position in the team structure, or adjust the team structure so that this person has high centrality.

Information Centrality and Initiation. Alternatively, increasing the amount of information initially possessed by a member, independent of the actual position in the communication net, has an effect on performance similar to that of increasing centrality (Shaw, 1954), by reducing the role of error-prone external communication links. If the decoder has some personal knowledge of the subject matter, availability of the information is facilitated. Availability errors arise from ignorance of the need for certain pertinent information, and/or non-use of the information due to difficulty in obtaining it. One might assign a critical task to a person who has expertise broad enough to complete the task independently, or to recognize the need for information and know where to obtain it.

2.3.3. Control of Error Detection

Very little research has been devoted to communication error detection in design. What does exist (Gilchrist et al., 1954; Leavitt, 1951; Shaw, 1954) describes it only as a dependent variable of the communication net, although other factors might affect it. Intuitively, error detection depends on the receiver's knowledge of the subject matter and a clear understanding of the relationship between the information error and its effect on the artifact.

Suppose that the value of pi is communicated to a decoder who will use it to fabricate a prototype. In the encoding process, one of the digits is transposed. The use of this erroneous value may not display a micro effect until far down the design process, perhaps when prototype parts are first assembled. But if the receiver had personally known the value of pi and the fact that the message was supposed to represent the value of pi, the error would have been more likely to be detected and corrected on receipt. If it had not been corrected on receipt, the error effect still would be quite salient on application, providing another means of detection. One might assign a critical task to an individual or small group of similar expertise so that they are likely to mutually understand the purpose, meaning, and significance of messages they exchange. Or one may define a critical task to encourage salient application of externally communicated information, or define it in such a way that erroneous critical information will result in a salient micro error effect.

2.3.4. Control of Corrective Effort

Research on corrective effort in design is sparse. The severity of corrective effort depends on the amount of work that must be done to replace erroneous information with correct information. Replacement may call for redesign of the component, as well as other components designed after it. Hence, the extent of corrective effort is determined by the nature and sequence of the tasks that make up the design project. One strategy recognizes that detection is facilitated by the appearance of a micro error effect. The sooner the error effect appears, the less severe the effort of correction. One might define a critical task so that application of externally communicated information, and hence a micro error effect, is likely to occur soon after reception.

2.4. Extension of FMEA to Design Team Analysis

This section presents a method for designing the design team itself to minimize the effect of communication errors. The cognitive model and TDA strategies are integrated into a tool which is traditionally used to evaluate the physical design artifact, Failure Modes and Effects Analysis (FMEA) (Ireson and Coombs, 1988). FMEA focuses on weaknesses in the artifact and makes them the object of design modifications. A full understanding of potential physical failure mechanisms of the artifact is required. Here, communication failure mechanisms are the object of FMEA. First, we must develop a functional concept of *component* that simultaneously represents both the team members and the artifact.

2.4.1. Definition of Component

Information components are defined as pieces of information that must be represented at design decision points. A piece of information always concerns some referent, sender and receiver, so a physical artifact and one or more team members are included. An information component fails when it is not represented at the decision point, or represents erroneous information.

Identification of significant components. Many information exchanges are only momentary and leave no physical evidence. Some important information might be transmitted subliminally, or might represent common domain knowledge between design participants and thus need not be explicitly exchanged. The importance of such information might not be obvious if the information does not travel along observable communication channels. One might notice them by the effect of their failure on the system (Blakar, 1973). Functional FMEA has been used to identify critical components as evidenced by their failure. This relieves us from exhaustively identifying all possible information components.

Reduction in number of components. Rather than attempting to exhaustively enumerate all information exchanges, we treat similar messages as units which have the same likelihood of information error and detection, and severity of effect. Referent, message, encoder, decoder, channel, and availability are constant. For example, the information needed by a structural engineer in determining the type of steel to fabricate a beam includes the strength-stiffness ratio of several different grades of steel. This information would likely come from the same source, arrive via the same communication channel, occur between the same encoder and decoder, and comprise information of equal complexity. By applying a TDA strategy to one specific information component in a class, all other components in its class should also experience reduced risk of information error.

2.4.2. Definition of Evaluation Function

The Risk Priority Number (RPN) reflects the detrimental effect of information errors. It is a function of the likelihood of the error and its detection, and the severity of corrective efforts or product flaws, as shown in equations (8.7) and (8.8). For information errors that are detected and require corrective effort during the sign process:

$$\mathrm{RPN}_{ce} = \mathrm{LIE} \times \mathrm{LD} \times \mathrm{SCE}. \tag{8.7}$$

where

RPN_{ce} = risk priority number associated with corrective effort

LIE = relative likelihood of information error, represented on a 1–9 scale

LD = relative likelihood of detection of information error, 1–9 scale

SCE = severity of corrective effort, 1–9 scale.

For information errors that are undetected and result in a product flaw:

$$\mathrm{RPN}_{pf} = \mathrm{LIE} \times \mathrm{LND} \times \mathrm{SPF} \tag{8.8}$$

where:

RPN_{pf} = risk priority number associated with product flaw

LND = likelihood of nondetection of information error, 10 − LD

SPF = severity of product flaw, 1–9 scale

In standard FMEA it suffices to estimate the severity of component failure on the performance of the artifact. In design team analysis, we are also concerned with the severity of mid-course corrective efforts. Table 8.4 shows subjective estimates for Likelihood of Information Error (LIE), Likelihood of Detection (LD), Severity of Corrective Effort (SCE) and Severity of Product Flaw (SPF) on a 1–9 adapted from Bajaria (1983) and Kapur (1988):

TABLE 8.4. Scales for likelihood of information error and detection and severity of corrective effort and product flaw.

Scale	LIE & LD	SCE	SPF
1	Extremely rate	No effect	No degradation
3	Not likely	Some backtracking	User annoyance
5	50/50 chance	Significant delay	User dissatisfaction
7	Likely	Redesign	Inoperation
9	Certain	Scrap project	Safety hazard

2.5. Example: Design Team for Computer-Assisted Instruction (CAI)

We illustrate the technique by analyzing an interdisciplinary team which designs software for computer-assisted instruction (CAI) programs on conic sections in analytic geometry, and dimensional analysis in chemistry (Safoutin, 1991). Three distinct sources of expertise are required; a classroom instructor, a CAI specialist and a computer programmer. The instructor contributes knowledge of the subject matter, and provides input at the beginning of the project, periodic review during the design process, and occasional consultation on details of the subject matter. The CAI specialist provides expertise in techniques for student interaction and presentation of material in a computer-based learning environment, which are very different from those of classroom instruction. A computer programmer determines the practicality of implementing the techniques developed by the CAI specialist and writes the computer code. The programmer has had informal exposure to CAI design principles, and feels comfortable with the task of instructional design and topic selection. Design is joint by review, where a preliminary design is conceived and implemented by the programmer, then evaluated by the CAI specialist and less frequently the instructor. It is assumed that the programmer will initiate a search for CAI methods and subject matter information when needed. The worksheet depicted in Figure 8.5 helps to elicit and organize information used to analyze a design team configuration.

A potential product flaw is first selected for analysis: the student cannot exit the computer interaction routine without providing the correct answer, a very frustrating experience if the student does not know the correct answer.

PRODUCT FLAW: Student cannot exit interaction routine without answering correctly

PRODUCT COMPONENT	COMPONENT FUNCTION	INFORMATION COMPONENT	INFORMATION FUNCTION	POSSIBLE INFORMATION ERROR
Presentation technique		Fact that the student		Designer of interactive
		will become frustrated		sequence does not
		without an exit option		comprehend the import-
				ance of an exit option

COMMUNICATION ACCURACY					AVAILABILITY
Referent	Message	Encoder	Decoder	Channel	
Student,	Exit option is	CAI designer	Programmer	External	Low. Little motivation on part
interaction	desirable to				of programmer to seek this
sequence	prevent frustra-				information. Little likelihood of
	tion				volunteer until review.

LIKELY ERROR MECHANISMS	LIKELIHOOD OF INFORMATION ERROR	LIKELIHOOD OF DETECTION	SEVERITY OF CORRECTIVE EFFORT	LIKELIHOOD OF NON-DETECTION	SEVERITY OF PRODUCT FLAW	CORRECTIVE EFFORT RISK	PRODUCT FLAW RISK
Lack of centrality, availability	5	3	4	7	6	60	210
Message objectivity							
Message ambiguity							
Initiation							

FIGURE 8.5. FMEA table for design team analysis.

The component function whose loss or degradation causes the flaw is then identified: the segments of computer code, procedures, algorithms, and presentation technique. In column 3 the most important information components are listed. Entered in column 4 is the function of the information which, if not fulfilled, results in physical component failure. The possible failure modes (information errors) are listed in column 5.

In the "Communicaton Accuracy" section the existing communicaton environment is described. The referent is the topic of the information component, in this case the relationship between the student and the interaction routine. The message is specific information about the referent, here that the student should have a means of exiting the interaction routine short of answering correctly, or he or she might become frustrated. The encoder is the team member who initially possesses the information, the CAI specialist. The decoder, the programmer, requires the information. The channel is the internal or external channel along which the message is likely to be exchanged.

Blocks to availability of the information component are recorded in "Availability" column. Currently the programmer is the initial designer of the presentation technique. Information must be obtained from the CAI specialist, and there are several obstacles. The CAI specialist is not always available for immediate consultation, and is not likely to offer CAI expertise unless the programmer-designer recognizes a need and requests information. The programmer's limited understanding of CAI design principles can impair the ability to recognize when CAI knowledge is called for before he or she has created a preliminary design. The programmer might not know or believe that an exit option is always important no matter how simple the question, and will likely proceed with a design based on incomplete or incorrect information which will not be corrected until review.

These observations are used to indicate in the next column that *availability* of expertise, *initiation* of information search, referent ambiguity and low message objectivity are likely error mechanisms. The referent is ambiguous because it involves the relationship between a hypothetical average student and an interaction routine, subjects that are somewhat ambiguous themselves. The message that the student will become frustrated without an exit option is subjective. It involves the personal perceptions of the student as envisioned by both the CAI specialist and the programmer, perceptions which are likely to be somewhat different. The CAI specialist is trained in the reasons why a student would become frustrated, but the programmer might not expect or believe that frustration would be significant. A programmer might object that an exit option makes the question too easy, or is not really necessary because the question seems simple enough to answer.

In the next block of columns, estimates for the likelihood of information error, detection, and the severity of corrective effort and product flaw are entered. Then, Corrective Effort Risk, RPN_{ce} is calculated using equation

(8.7) and Product Flaw Risk, RPN_{pf} is calculated using equation (8.8). Only the first figure of an RPN is significant and should be used only for comparative purposes in evaluating one information error against another. The analyst has made estimations that compute to $RPN_{ce} = 60$ and $RPN_{pf} = 210$.

In the strategies column, TDA strategies as defined earlier that can reduce corrective effort risk and product flaw risk are identified. Corrective effort risk can be reduced by decreasing the likelihood of an information error (LIE) or the severity of of corrective effort (SCE). Product flaw risk can be reduced by decreasing the likelihood of an information error (LIE), its nondetection (LND) or the severity of the product flaw (SPF).

The communication failure is that provisions might not be specified for the student to be able to exit an interaction routine without answering correctly. To prevent this flaw, the designer must know that the student will experience frustration if the correct answer is required for exit but is unknown. If this fact is not known to the designer, it is assumed that he or she will not provide an exit option.

Review and detection is the most likely means by which CAI expertise will make its way into the design. Error detection is estimated to be not likely (at 3). The analysis has revealed that reliance on detection is not wise given the current TDA. Rather than rely on periodic review, one should strive to decrease the relatively high (at 5) likelihood of information error LIE and increase likelihood of detection LD by modifying the TDA.

When an information error is detected, the corrective effort required depends on the amount of time and work that proceeds past the point of use of the erroneous information. Currently, severity of corrective effort is estimated at 4. To decrease this measure our only means is to define the task or make an assignment so that application of externally communicated information is likely to occur soon after reception.

It had been believed wise to assign the design of the presentation technique to the programmer. Our analysis shows that representation of CAI expertise in the presentation technique is very tenuous, relying too heavily on error detection. If the design of the presentation technique was instead assigned explicitly to the CAI specialist, CAI expertise would be more available. A CAI specialist directly designing the presentation technique is more likely to give full credit to the importance of an exit option. This assignment would also reduce losses due to referent ambiguity and message objectivity.

If the CAI specialist explicitly specifies that the presentation technique include an exit option, the programmer provides one. However, if there is only a *recommendation* that an exit option be included, the programmer might decide that the extra programming effort is not justified by the benefits (which he perceives to be minimal). Upon review, the programmer might be instructed to add the option, but this entails corrective effort. Accurate receipt and implementation of the message would no longer depend on its interpretation by the programmer.

Risk Reduction. Formerly,

$$\begin{array}{ccccccc} \text{LIE} & \times & \text{LND} & \times & \text{SPF} & = & \text{RPN}_{pf} \\ 5 & \times & 7 & \times & 6 & = & 210 \end{array}.$$

With the new assignment, the analyst estimates that LIE = 2 and LND = 3, significantly lowering RPN_{pf} to 36. Similarly, RPN_{ce} would change. Formerly,

$$\begin{array}{ccccccc} \text{LIE} & \times & \text{LD} & \times & \text{SCE} & = & \text{RPN}_{ce} \\ 5 & \times & 3 & \times & 4 & = & 60 \end{array}.$$

With the new assignment, LIE = 2 and LD = 10 − LND = 7, lowering RPN_{ce} to 56.

2.6. Discussion

When all TDA modifications have been specified, the analysis can be repeated until RPNs have been reduced to an acceptable level. Even if TDA adjustment proves impractical, high risk areas can be identified for increased monitoring during the design process. The analysis pointed out inadequate availability of CAI expertise and an over reliance on error detection. One might naturally suspect that a programmer without formal CAI training might not be adequate to ensure a quality presentation technique. However, it might also seem apparent that periodic review by a CAI specialist would uncover and correct any such deficiency. We saw that this was not the case. Our analysis broke the problem down into distinct components relating to the likelihood of an information error, its detection, the severity of the effort spent to correct the error during the design process and the seventy of a product flaw if the error goes undetected. The example leads to the conclusion that assignment directly to a CAI specialist is more effective because it reduces reliance on detection, increases availability, and increases communication accuracy.

3. Summary

Group decision making is difficult because of problems of communication between individuals, and because individuals often have conflicting interests. All of the methods presented in Part 1, including voting and ranking schemes, provide a forum for helping individuals communicate their interests and preferences to one another. Multiattribute utility analysis and the analytic hierarchy process also provide a structured procedure for eliciting preferences, and a rigorous methodology for converting expressions of preference into a quantitative measure of the relative merit of design alternatives. While these methods do not resolve the impossibility of interpersonal comparison of utility, they do provide a framework around which individuals can provide input, express preferences, analyze possible courses of action and negotiate tradeoffs.

Part 2 presented a technique for interdisciplinary design team management which focuses on communication errors. Mechanisms of information processing errors and their role in requiring corrective effort during the design process and causing product flaws were identified. We have employed this understanding toward a new methodology for analyzing the effectiveness of design teams which focuses directly on task definition and assignment strategies. It pinpoints the location, nature, and prevention of information-related sources of design error and over-reliance on error detection. It also provides documentation of potential sources of error and a means of remedy where only gut feelings existed before.

Each of the methods presented here disaggregate a previously intractable problem into subproblems on which team members can communicate and/or reach consensus. These approaches are normative in that they are intended to indicate how design teams *should* make decisions, rather than automate their current decision making processes. The analyst must think hard about which aspects of current procedures should be retained, and which should be replaced with methods which are intended to improve the outcome of the decision making process.

Acknowledgment

The author gratefully acknowledges the support of the National Science Foundation under PYI award DDM-8957420.

References

Allen, Thomas J., "Organizational Structure, Information Technology, and R&D Productivity," *IEEE Trans. on Engineering Management*, EM-33:4, 212–217, 1986.

Allen, T. J., D. M. S. Lee and M. L. Tushman, "R&D Performance as a Function of Internal Communication Project Management, and Nature of the Work," *IEEE Trans. on Engineering Management*, EM-27:1, 2–12, 1980.

Arrow, K., *Social Choice and Individual Values*, New York: Wiley, 1951.

Barge, J. K. and R. Y. Hirokawa, "Toward a Communication Competency Model of Group Leadership," *Small Group Behavior*, 20:2, 167–189, 1989.

Bales, R. F. and P. F. Slater, *Role Differentiation in Small Decision Decision Making Groups*, *Family*, *Socialization and Interaction Process*, 259–306, New York: Free Press, 1955.

Bajaria, H. J., "Integration of Reliability, Maintainability, and Quality Parameters in Design," 29th L. Ray Buckendale Lecture, SAE, 1983.

Bavelas, A., "A Mathematical Model for Group Structures," *Applied Anthropology*, 7 (1948), 16–30, 1948.

Bavelas, A., "Communication Patterns in Task-Oriented Groups," *Journal of the Acoustical Society of America*, 22, 725–730, 1950.

Blakar, R. M., "An Experimental Method for Inquiring into Communication," *Eur. J. Soc. Psychol*, 3(4), 415–425, 1973.

Carzo, Jr., R., "Some Effects of Organization Structure on Group Effectiveness," *Admin. Sci. Quarterly*, 393–424, 1963.

Chakrabarti, A. K., S. Feineman, and W. Ruentevilla, "Characteristics of Sources, Channels, and Contents for Scientific and Technical Information Systems in Industrial R and D," *IEEE Trans. on Engineering Management*, EM-30:2, 83–88, 1983.

Chapanis, A. et al., "Studies in Interactive Communication: I. The Effects of Four Communication Modes on the Behavior of Teams During Cooperative Problem Solving," *Human Factors*, 14:6, 487–509, 1972.

Condorcet, Marquis de, "Essai sur l'application de l'analyse a la probabilite des decisions rendues a la pluralite des vois", Paris, 1785.

DeSanctis, G. and R. B. Gallupe, "A Foundation for the Study of Group Decision Support Systems," Management Science, Vol. 33, No. 5, May 1987.

Dyer, J. S., "A Clarification of 'Remarks on the Analytic Hierarchy Process'," *Management Science*, Vol. 36, No. 3, March 1990.

Dyer, J. S., "Remarks on the Analytic Hierarchy Process," *Management Science*, Vol. 36, No. 3, March 1990.

Earle, J., *Engineering Design Graphics*, New York: John Wiley, 1985.

Eder, W. E., "Information Systems for Designers," *Proceedings of the Institution of Mechanical Engineers, International Conference on Engineering Design*, Vol. 2, 1307–1319, 1989.

Fiedler, F. E., *A Theory of Leadership Effectiveness*, New York: McGraw Hill, 1967.

Finger, S. and J. R. Dixon, "A Review of Research in Mechanical Engineering Design. Part I: Descriptive, Prescriptive, and Computer-Based Models of Design Processes," *Research in Engineering Design*, 1, 51–67, 1989.

Fishburn, P. C. "Independence in Utility Theory with Whole Product Sets," *Operations Research*, Vol. 13, 1965, pp. 28–45.

French, S., *Decision Theory*: *An Introduction to the Mathematics of Rationality*, New York: John Wiley and Sons, 1986.

Gebala, D. A., Eppinger, S. D., "Methods for Analyzing Design Procedures," *Proceedings of ASME Conference on Design Theory and Methodology*, 1991.

Gilchrist, J. C., et al., "Some Effects of Unequal Distribution of Information in a Wheel Group Structure," *J. Abn. Soc. Psych.*, 49, 554–556, 1954.

Hackman, R. J. and C. G. Morris, "Group Tasks, Group Interaction Process, and Group Performance Effectiveness," *Small Groups and Social Interaction*, New York: John Wiley, 1983.

Hare, P. A., *Handbook of Small Group Research*, Second Edition, New York: The Free Press, 330–355, 1976.

Harker, P. T., Vargas, L. G., "Reply to 'Remarks on The Analytic Hierarchy Process' by J. S. Dyer," *Management Science*, Vol. 36, No. 3, March 1990.

Hauser, J. R. and D. Clausing, "The House of Quality," *Harvard Business Review*, 66:3, 1988, 63–73.

Hill, G. W., "Are n + 1 heads better than 1?," *Psychological Bulletin*, 91, 517–539, 1982.

Hogarth, R., *Judgement and Choice*, New York: Wiley, 1980.

Ireson, W. B. and C. F. Coombs (eds.), *Handbook of Reliability Engineering and Management*, New York: McGraw-Hill, 13.5–13.33 and 18.11–18.25, 1988.

Kapur, Kailash C., "Techniques of Estimating Reliability at Design Stage," Chapter 18 in Ireson, W. G. *Handbook of Reliability Engineering and Management*, New York: McGraw-Hill, 1988.

Keefer, D. L., "Allocation Planning for R & D with Uncertainty and Multiple Objectives," *IEEE Transaction on Engineering Management*, Vol. EM-25, No. 1, February 1978.

Keeney, R. L., Raiffa, H., *Decisions with Multiple Objectives: Preferences and Value Tradeoffs*, New York: Wiley and Sons, 1976.

Kiggundu, M. N., "Task Interdependence and Job Design: Test of a Theory," *Organizational Behavior and Human Performance*, 31, 145–172, 1983.

Kirkwood, C. W., "Pareto Optimality and Equity in Social Decision Analysis," *IEEE Transactions on Systems, Man and Cybernetics*, Vol. SMC-9, No. 2, February, 1979.

Klein, G. A., R. Calderwood, "Decision Models: Some Lessons from the Field," *IEEE Transactions on Systems, Man, and Cybernetics*, Vol. 21, No. 5, 1991.

Krishnan, V., Eppinger, S. D., Whitney, D. E., "Towards a Cooperative Design Methodology Analysis of Sequential Decision Strategies," *Proceedings of ASME Conference on Design Theory and Methodology*, 1991.

Leavitt, H. J., "Some Effects of Certain Communication Patterns on Group Performance," *J. Abn. Soc. Psych.*, 46, 38–50, 1951.

Liker, J. K. and W. M. Hancock, "Organizational Systems Barriers to Engineering Effectiveness," *IEEE Trans. on Engineering Management*, EM-33:2, 82–91, 1986.

Luce, R. D., and H. Raiffa, *Games and Decisions*, New York: Wiley, 1957.

Marriott, D. L. and N. R., "Materials Failure Logic Models: A Procedure for Systematic Identification of Material Failure Modes in Mechanical Components," *Proceedings, Conference on Failure Prevention and Reliability*, ASME, Hartford, CT, Sept. 1981.

McMahon, E. H., "Evaluation of Group Design in Engineering," *Proceedings of ASME Conference on Design Theory and Methodology*, 1991.

Mehrabian, A. and H. Reed, "Some Determinants of Communication Accuracy," *Psychological Bulletin*, 70:5, 365–381, 1968.

Nunamaker, J. F., L. M. Applegate and B. R. Konsynski, "Computer-Aided Deliberation: Model management and Group Decision Support," *Operations Research*, Vol. 36, No. 6, Nov-Dec. 1988.

Perlman, G., "Descriptive Models of Cognitive Aspects of the Engineering Design Process," *Design Theory '88: Proceedings of the 1988 NSF Grantee Workshop in Design Theory and Methodology*, New York: Springer-Verlag, 1989.

Pugh, S., "Concept Selection—A Method that Works," *Proceedings ICED*, Rome, 1981, pp. 497–506.

Pugh, S., *Total Design*, Reading, MA: Addison-Wesley, 1990.

Roby, T. B. and Lanzetta, J. T., "Work Group Structure, Communication, and Group Performance," *Sociometry*, 19:1956, p. 105–113.

Runkel, P. J., "Cognitive Similarity in Facilitating Communication," *Sociometry*, 19, 189–191, 1956.

Saaty, T. (1980), *The Analytic Hierarchy Process*, New York: McGraw-Hill (revised and extended, 1988).

Saaty, T., "An Exposition of the AHP in Reply to the Paper 'Remarks on The Analytic Hierarchy Process'," *Management Science*, Vol. 36, No. 3, March 1990.

Safoutin, M. J. and Thurston, D. L., "A Communications-Based Technique for Interdisciplinary Design Team Management," *IEEE Transactions on Engineering Management*, Vol. 40, No. 4, 1993.

Savage, L. J., *The Foundations of Statistics*, New York: Wiley, 1954.

Shaw, M. E., *Group Dynamics: The Psychology of Small Groups*. New York: McGraw Hill, 1976.

Shaw, M. E., "Some Effects of Unequal Distribution of Information Upon Group Performance in Various Communication Nets," *J. Abn. Exp. Psych.*, 49:1954, p. 547–553.

Steiner, I. D., *Group Process and Productivity*, New York: Academic Press, 1972.

Steward, D. V., "The Design Structure System: A Method for Managing the Design of Complex Systems," *IEEE Trans. on Engineering Management*, EM-28:3, 71–74, 1981.

Stomph-Blessing, L. T. M., "Analysing an Engineering Design Process in Industry," *Proceedings of the Institution of Mechanical Engineers, Int'l Conference on Engr. Design,* v. 1 p. 57–64. August 1989.

Sullivan, L. P., "Quality Function Deployment," *Quality Progress*, 1986, 39–50.

Sycara, K. P. and C. M. Lewis, "Modeling Group Decision Making and Negotiation in Concurrent Product Design," *Systems Automation: Research and Applications*, Vol. 1, No. 3, 1991.

Thamhain, H. J., "Managing Engineers Effectively," *IEEE Trans. on Engineering Management*, EM-30:4, 231–237, 1983.

Thamhain, H. J. and D. L. Wilemon, "Leadership Effectiveness in Program Management," *IEEE Trans. on Engineering Management*, EM-24:3, 102–108, 1977.

Thurston, D. L., and Tian, Y. Q., "Integration of the Analytic Hierarchy Process with Integer Linear Programming for Long Range Product Planning," *Mathematical and Computer Modelling*, Vol. 17, No. 4/5, 1993.

Thurston, D. L., "A Formal Method for Subjective Design Evaluation with Multiple Attributes," *Research in Engineering Design*, Volume 3, Number 2, 1991.

Thurston, D. L. and Carnahan, J. V., "Fuzzy Ratings and Utility Analysis in Preliminary Design Evaluation of Multiple Attributes," *ASME J. of Mechnical Design*, Vol. 114, No. 4, December 1992.

Thurston, D. L. and Liu, T., "Design Evaluation of Multiple Attribute Under Uncertainty." *Systems Automation: Research and Applications*, Vol. 1, No. 2, 1991.

Ullman, D. G., "A Taxonomy of Mechanical Design," 1989 ASME Technical Conference: 1st International Conference on Engineering Design Theory and Methodology, Montreal, September 1989, pp. 23–36.

Wallace, K. M. and Hales, C., "Engineering Design Research Areas," *Proceedings of the Institution of Mechanical Engineers, International Conference on Engineering Design*, Vol. 1, Aug. 1989, p. 555–562.

Wolek, F. W., "The Complexity of Messages in Science and Engineering: An Influence on Patterns of Communication," *Communication Among Scientists and Engineers*, 233–337, 1970.

von Neumann, J. and O. Morgenstern, *Theory of Games and Economic Behavior*, 2nd ed. Princeton, NJ: Princeton University Press, 1947.

Watson, S. R. and D. M. Buede, *Decision Synthesis: The Principles and Practice of Decision Analysis*, Cambridge: Cambridge University Press, 1987.

9
Routineness Revisited

David C. Brown

Abstract. In the current research literature on the use of artificial intelligence (AI) in design, we find many terms for types of design. In particular, the term *routine design* is often used, with a variety of definitions. The goal of this chapter is to discuss routine design, and to contrast it with some of the other types of design. We will attempt to clarify the definition of routineness, and point out what is missing from existing definitions. We will also consider definitions of, and comments about routine design from other authors, as a contrast to our definition. In conclusion, we relate the notion of class 1, 2, and 3 types of design, introduced by Brown and Chandrasekaran (1985), to ideas presented in this chapter.

9.1. Introduction

In books and papers about design problem-solving we find many terms for types of design [for example, see AAAI (1990) and Finger and Dixon (1989)]. These include *preliminary*, *conceptual*, *functional*, *innovative*, *creative*, *routine*, *embodiment*, *parametric*, *detailed*, *redesign*, *nonroutine*, and *configuration*.

The goal of this chapter is to discuss routine design and to contrast it with some of the activities suggested by the other terms given above.

As Gero (1990, p. 34) says, "There seems to be a general acceptance of the classification of design into routine, innovative, and creative (Brown and Chandrasekaran, 1985)...." Unfortunately, many people have used the term *routine* in slightly different ways, often without understanding the key points of the original description. In this chapter we will try to point to the sources of confusion, and will try to clarify the definition of the term.

9.1.1. Three Classes of Design

Let us start by considering the following passages from Brown and Chandrasekaran (1985):

Class 1 Design

The average designer in industry will rarely if ever do class 1 design, as we consider this to lead to major inventions or completely new products. It will often lead to the formation of a new company, division, or major marketing effort. This is extremely innovative behavior, and we suspect that very little design activity is in this class. For this class, neither the knowledge sources nor the problem-solving strategies are known in advance.

Class 2 Design

This is closer to routine, but will involve substantial innovation. This will require different types of problem-solvers in cooperation and will certainly include some planning. Class 2 design may arise during routine design when a new requirement is introduced that takes the design away from routine, requiring the use of new components and techniques. What makes this class 2 and not class 1 is that the knowledge sources can be identified in advance, but the problem-solving strategies, however, cannot.

Class 3 Design

Here a design proceeds by selecting among previously known sets of well-understood design alternatives. At each point in the design the choices may be simple, but overall the task is still too complex for it to be done merely by looking it up in a database of designs, as there are just too many possible combinations of initial requirements. The choices at each point may be simple, but that does not imply that the design process itself is simple, or that the components so designed must be simple. We feel that a significant portion of design activity falls into this class.

Class 3 Complexity

While class 3 design can be complex overall, at each stage the design alternatives are not as open-ended as they might be for class 2 or 1, thus requiring no planning during the design. In addition, all of the design goals and requirements are fully specified, subcomponents and functions already known, and knowledge sources already identified. For other classes of design this need not be the case.

9.1.2. The Key Points

Let us now discuss the key points of that definition, and add some of the refinements which appeared in that paper and in subsequent papers.

The main point (due mainly to Chandrasekaran), which is often overlooked, is summarized in the following table:

	Knowledge sources	Problem-solving strategies
Class 1	Not known	Not known
Class 2	Known	Not known
Class 3	Known	Known

For class 3 design, this means that everything about the design process, including the knowledge needed (i.e., knowledge sources), must be known in advance. Note that this does not mean that the specific design (i.e., the solution) is known in advance. Nor does it mean that the pattern of use of the knowledge (i.e., the design trace) is completely known in advance.

9.1.3. "Known" Knowledge

There is some ambiguity in the use of the word *Known* in the above table. We will discuss this in terms of knowledge sources, with obvious extension to problem-solving strategies.

By referring to a knowledge source as "known", we mean:

- that it is known in advance that the knowledge source will be needed to make that decision or set of decisions, and
- that the knowledge source is "immediately available" for use—that is, it does not have to be reasoned out or transformed from some other knowledge.

9.1.4. The Implications for Class 3 Design

The implications for class 3 design are:

- Use of a fixed set of well-understood design plans.
- No planning is required, only plan selection.
- Plan selection is fairly simple, with known criteria.
- Plans are probably not very long, or they would not be easily remembered.
- Possible problem decompositions are known in advance, while the actual decomposition to be used is not.
- Dependencies between subproblems are known and, for the most part, can be compensated for in advance.
- Subproblems can usually be solved in a fixed order with little or no backtracking, due to the anticipated dependencies.
- All possible subcomponents of the object being designed are known in advance.
- The particular configuration of subcomponents chosen for a design in response to a given set of requirements is not known before the design activity starts. However, that configuration of subcomponents is a previously known configuration (i.e., the designer could identify it as a candidate solution for that type of design problem).
- All attributes or parameters (e.g., length) of the design of a subcomponent are known (i.e., their names, *not* their values).
- The knowledge needed to calculate or select a value for each attribute is known in advance.
- Appropriate ranges of values are known for most attributes.

- There exist "expectations" about a typical value for an attribute in a particular design situation.
- The types of requirements given for a design problem are all known in advance.
- Many common failures during the design process will be recognizable.
- There exist suggestions about how to make changes to parameter values in order to fix failures.

9.1.5. AIR-CYL

The AIR-CYL system (Brown and Chandrasekaran, 1989) that designed air cylinders is an example of a class 3 design system. The possible configurations are known in advance and are selected at run-time as a side-effect of plan selection; the possible plans for each subproblems are all available; and the parameters to be given values are all known in advance, as is the knowledge used to produce those values.

The system, written in DSPL, a language for constructing design expert systems, also satisfies all of the other criteria in the list above. AIR-CYL is a system that does routine design. DSPL has been used to build systems for a variety of domains, such as operational amplifiers, gear pairs, distillation columns, and commercial buildings.

In the following sections we will attempt to clarify the definition of routineness, and point out what is missing from the presentation above. Then we will consider definitions of and comments about routine design from other authors, as a contrast to the definition presented here.

9.2. A Second Axis

The author's recent work (Brown, 1991) addresses a form of learning known as *compilation*, in which knowledge is transformed and reorganized in order to produce more efficient problem-solving.

The thesis that underlies the work in knowledge compilation during design is that design tasks become routine due to learning. This learning is brought about by repetition of similar problem-solving. That is, routineness is a direct reflection of experience. Routine designs are done more efficiently.

In order to avoid unwanted connotations, we will use the term *nonroutine* as the opposite of *routine*. The level of experience with a certain type of design will be reflected by a position on a routine → nonroutine axis—with "very experienced" at one end and "inexperienced" at the other.

As this axis has nothing to do with *what* is being decided, this suggests the need for another axis that describes what sort of decisions are being made at various points during a design. We will use a conceptual → parametric axis for that. The intuition is that the axis shows the abstractness of the decisions being made, and reflects the notion that more constraints are added to the solution as the design activity progresses.

By *conceptual* design we mean that the kind of things being decided at that point in the design are abstract (conceptual). For example, that the design requirements can be satisfied by a design that provides a particular function, or by one that has a particular pattern of subfunctions.

This is quite compatible with Dixon's very useful taxonomy of design problems (Dixon et al., 1988). His levels are named *functional*, *phenomenological*, *embodiment*, *attribute*, and *parametric*. Clearly, these levels correspond to portions of the conceptual → parametric axis, even though this axis is less specific about the content of the decisions being made.

Dixon goes further, and states that "*conceptual* design is often used to describe the Embodiment of a design from Function" (p. 43). He considers *preliminary* design to be an extension of conceptual design to another of his levels of specificity, i.e., to artifact type.

By *parametric* design we mean that the things being decided are values for a prespecified set of attributes, and that providing values for these attributes fully specifies the design. In Dixon's terms, the design goes from artifact type level to the artifact instance level.

For many design problems, the conceptual → parametric axis represents the flow of time during the design activity, with earlier decisions falling toward the left and later decisions falling toward the right.

However, not all design problems have to begin with vague functional requirements and conclude with a fully specified design. For example, Dixon et al. (1988) point out that a design activity can start at any level of abstraction and finish at any one of the more specific levels.

9.2.1. *Four Categories of Design Activity*

We consider the routine → nonroutine and conceptual → parametric axes to be orthogonal (see Figure 9.1).

The space produced is naturally divided into four categories of design activity. They are represented by the four extreme points at the limits of the axes:

RC	routine, conceptual design
RP	routine, parametric design
NRC	nonroutine, conceptual design
NRP	nonroutine, parametric

These will each be discussed below, in Sections 9.3 and 9.4.

9.2.2. *Concerns About the Analysis*

At this point it is appropriate to discuss several concerns about this two-axis analysis.

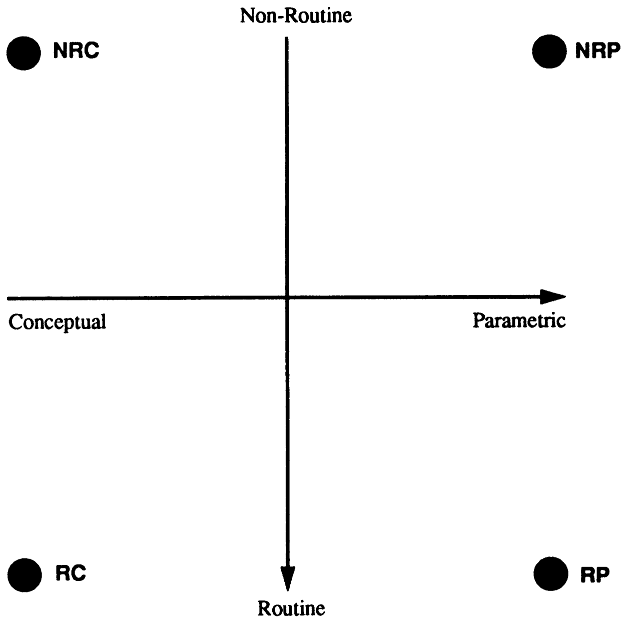


FIGURE 9.1. Orthogonal axes.

Relative measure: As already stated, the routineness of a particular design problem depends on the experience of the problem-solver. Therefore, routineness is a relative measure. What is routine for one designer is not routine for another. What is routine for a designer today, may not have been two years ago. Routineness is in the brain of the beholder. It is an *individual's standard.*

In addition, there is also a *community standard.* The professional engineering design community may consider a design problem routine—meaning that there is an expectation that the problem will be routine for each member of the community. This may be because the specific knowledge and problem-solving for that problem is taught in college.

This community standard is probably easier to see at the Nonroutine end of the axis. Suppose we associate Nonroutine design activity with "innovation." It is easy to see that the community standard for a particular design problem is represented by the existing design solutions. Thus, a design can be innovative relative to that pool of existing designs.

Of course, it is perfectly possible for a design to be innovative relative to the individual's standard, but not innovative relative to the community standard. Design problems by themselves are not innovative, only in context. This demonstrates some of the danger in using the term *innovative design.*

The routineness axis: First, as routineness is expressed on an axis, with the possibility of different degrees of routineness, one should not assume that there are only four categories of design activity (i.e., it is closer to being continuous than discrete). Routine and nonroutine are the extremes. We will try to restrict our focus to the extremes of both axes, in order to simplify the analysis.

What was learned: The routine → nonroutine axis is supposed to reflect the level of experience with a particular type of design. The more routine a problem is, the more knowledge is already "known" and is ready for immediate use. In the table at the beginning of the chapter we separated what could be known in advance of carrying out a design into knowledge sources and problem-solving strategies. The routine → nonroutine axis is concerned with how much is known, but does not distinguish between these two types of knowledge. A more refined analysis probably should make this distinction.

Subproblem type: Up to this point we have assumed that all subproblems of a design problem are of the same class. This is not always realistic. In complicated problems some subproblems will be quite new, and will be nonroutine, whereas other subproblems will lead to very well-known components needing routine design, or even merely selection from a catalog. Clearly, this makes any model of design more complex.

Nonlinear progress: The reader should not assume that the nice, linear progress through a design problem that is "suggested" by the conceptual → parametric axis is correct. Different subproblems can be at different points on the axis at any point in time. Problem-solving can jump from one point on the axis to another—for example, when a decision about using a certain type of component suggests a simplification of the functional design (perhaps through function sharing). Also, failures during design, due perhaps to incompatible choices, can lead to redesign (making changes to something already designed) or to re-design (doing whole portions of the design again from scratch). Nonlinear progress should not affect the arguments presented in this chapter.

Other axes: This two axis analysis ignores other dimensions. Several people, such as Chandrasekaran (1990), Brown (1992), and Hayes-Roth (1990), have discussed the need for multiple mechanisms, or methods, for design tasks. For example, the use of constraint satisfaction or case-based reasoning to produce a design candidate. Our analysis does not reflect that dimension, and does not require it. The analysis also ignores the effect of the Domain (e.g., mechanical versus electrical) on the design activity [for example, see Brown (1990) or Waldron (1990)].

In the next two sections we will examine the four extreme categories of design activity (RC, RP, NRC, and NRP), giving examples of each.

9.3. Routine Design

In this section we will examine two of the four extreme points defined by the two axes. They are at the routine end of the routine → nonroutine axis.

9.3.1. Routine, Parametric Design

At the RP point the designer is deciding values for parameters (parametric), and has well-formed methods for deciding them (routine).

This is a typical routine situation, where a designer uses well-known methods to decide values for parameters. Several existing knowledge-based systems are capable of doing this category of design activity, such as AIR-CYL (Brown and Chandrasekaran, 1989), and PRIDE (Mittal et al., 1986).

9.3.2. Routine, Conceptual Design

At the RC point the designer is making very abstract decisions (conceptual) and has well-formed methods for deciding them (routine).

This category of design is done by a designer who often designs complex things given a rich but fixed set of requirements. For example, the designer of low-cost office buildings needs to decide which of a standard set of designs to use, and what type of structural system to use, given the type of equipment and numbers of people to be placed in the building. He or she also needs to consider the geological information about the site, as well as other factors such as the weather. The decisions made are not final values of parameters, but rather attributes of the design that will allow a list of parameters to be formed so that parametric design can be done.

The best known knowledge-based system that is close to this kind of design is HI-RISE (Maher and Fenves, 1985). HI-RISE acts as an assistant to a designer for the preliminary structural design of high-rise buildings. It generates "feasible alternatives for two functional systems," in the form of structural systems. As these functional systems are known in advance, and the methods for selecting and checking the compatibility of the structural systems are also known in advance, then the system is doing routine design. As many of its decisions are fairly abstract, such as "braced frame" versus "shear wall" construction, the system belongs toward the conceptual end of the conceptual → parametric axis.

A correction: In Section 9.1.4 we presented a list of implications of the earlier definition of class 3 design. Unfortunately, a few of the points refer to "subcomponents." The RC category of design activity need not decide subcomponents. Consequently, those points should be changed to include more abstract decisions, such as "subfunctions."

9.4. Nonroutine Design

In this section we will examine the other two of the four extreme points defined by the two axes. They are at the nonroutine end of the routine → nonroutine axis.

9.4.1. *Nonroutine, Conceptual Design*

At the NRC point the designer is making very abstract decisions (conceptual), and does not have any well-formed approach to making them (nonroutine).

This is the aspect of design about which we know the least. It is easy to think of the most abstract decisions being nonroutine. A typical early design task might be deciding the full functionality of the object to be designed given the requirements. Those aspects of design that we normally consider to be most creative are precisely those in the NRC category. This is the sort of activity we associate with the initial stages of architectural design, for example.

There have been some attempts to produce design systems in this category [see Gero and Maher (1989) and Joskowicz et al., (1992)]. For example, Ulrich and Seering (1989) describe a system that generates graphs of functional elements (i.e., schematic descriptions) that produce a required relationship between a given input and a given output. They call this process *schematic synthesis*. The descriptions consist of idealized elements, such as pumps, or springs, which contain no information about geometry or materials. Descriptions can then be used to generate a physical description.

9.4.2. *Nonroutine, Parametric Design*

At the NRP point the designer is deciding values for parameters (parametric), and does not have any well-formed approach to making them (nonroutine).

One can easily imagine a new designer in industry being given the final step of a design project, where the rest had been completed by a senior designer. It is clearly possible for the naive designer to know all of the parameters to be decided, but not know how to go about deciding them. This would result in nonroutine behavior, such as analyzing the dependencies between the parameters in order to determine an appropriate order in which to decide them, or searching textbooks for appropriate methods or equations.

Another more complex example can be found in the task of designing hulls for racing yachts, as described by Gelsey in Joskowicz et al. (1992, p. 44). The hull's shape can be described by a grid of planar panels. The problem can be viewed as that of finding sizes for those panels, i.e., finding values for parameters. In actual fact, the grid may need to be changed, in order to improve expected performance. That sort of change will produce a *new* set of parameters. Producing this new set is *not* a parametric design task.

The statement in Section 9.2 that "routineness is a direct reflection of experience" is not meant to imply that all problems can produce the same degree of routineness with experience. For example, some problems have a dependency structure that is much too complex to be properly analyzed by the designer. Even if the dependencies were known *a priori*, the ordering of the tasks may not be, as in the tasks Balkany et al. (1991) label as Type 2.

Such complexity might lead to use of the iterative refinement approach to parametric design (Orelup et al., 1988; Ramachandran et al., 1988). One can argue that this is not routine design, as it is not possible to anticipate the order of decisions, therefore preformed plans cannot exist and cannot be used. In addition, an iterative refinement approach will decide the value of some parameters more than once, nudging them gradually to their final acceptable position.

9.5. Related Work

In this section we will consider some recent definitions of, and comments about, routine design from other authors. This is by no means an exhaustive review. It does not concentrate on definitions that we consider to be totally wrong. It is merely intended to show the range of variation in the literature. Many authors use the term *routine* with no associated definition.

Gero (1990) proposes a model of design based on the retrieval and instantiation of "design prototypes" that bring "all the requisite knowledge appropriate to the design situation together in one schema." This knowledge includes function, behavior, structure, relational, qualitative, computational, constraints, and context.

"Routine design can be defined as the design that proceeds within a well-defined state space of potentials designs. That is, all the variables and their applicable ranges, as well as the knowledge to compute their values, are all directly instantiable from existing design prototypes" [p. 34].

On the surface this appears to be quite compatible with our definition. However, he also states that "instances are refined in two ways. The first way is by pruning the set of variables to the applicable set through a specification of applicable functions, structures, or behaviors and propagating this specification. The second way is by determining the values of the applicable set of variables using the available knowledge."

This "second way" is clearly routine, but the "first way" implies that the set of variables to be given values needs to be determined via propagation. This would mean that neither the variables nor the methods for giving them values are "known," in the sense already defined. Consequently, there is some conflict here with our view.

Tomiyama (1990) lists classes of design as "Creative, New, Combinatory, Routine, Parametric, and Redesign." Although he avoids the confusion between routine and parametric, the relationship between routine and the other types is unclear (especially for "New").

Tomiyama also presents "Attribute Modeling," where design objects only have attributes/parameters, design objects do not change their structure, and "Well Formalized" design processes act on attributes. This, he claims, is used to deal with "Routine-Type Design." However, "constraint solving" is allowed as a design process. If this were to be done by the usual constraint satisfaction methods this would not be routine given our definition, as plans are not available and some search is required.

Agogino, in her presentation at a recent AAAI workshop (Joskowicz et al., 1992), argued that routine design introduced "no new variables," while in nonroutine design "new variables are created." This definition is in terms of the conceptual → parametric axis, and not strictly in terms of routineness.

Snavely et al. (1990) present four "mutually exclusive types of design," called invention, innovation, routine, and procedural. For them, routine design "is the process of filling the slots of a fixed topology (or predetermined set of fixed topologies) with catalog entries." A "catalog" is a database with multiple levels of abstraction, with the lowest being typical part catalog entries (e.g., a particular spring). As they allow a catalog entry to be a "dimension," this appears to overlap parametric design. Although one could argue that the topology is "fixed" because it is "known" to be the solution, that does not appear to be their claim. Their main criterion for distinguishing between their types is the variability of the topology—the higher the variability, the more inventive.

Sriram and Tong (1990) provide a formal definition of design as (S, C, A, K, Δ), where S is the set of solutions, C is the set of constraints that need to be satisfied, A is the set of artifacts, K is the knowledge used to develop S, and Δ is the set of transformation operators. They list "design activities" as creative, innovative, and routine. They distinguish between them by which of the ingredients of the formal definition are known. Thus, their definition is one of the few that have activities arranged solely along the routine → nonroutine axis.

9.6. Summary and Confusion

Where do the three classes presented in Section 9.1.1 fit into this new two axis analysis? As the classes are concerned with how much is already *known*, as opposed to what is decided, it is clear that they are positioned along the routine → nonroutine axis (see Figure 9.2).

The figure is *not* intended to be taken too literally. The rectangles representing the classes are put in representative places. Class 3 covers all the routine cases. Class 1 covers all the nonroutine cases. Class 2 is between them. Where exactly are the boundaries? This isn't clear. Does class 2 cover the rest of the space? Yes, if these are really supposed to cover *all* the possibilities.

One could even propose a new class, perhaps Class 2a, with knowledge sources "not known" and problem-solving strategies "known"—where the approach to solving the problem was known in advance but the knowledge to be used wasn't. It isn't clear that this situation would often occur. As exactly *what* (i.e., knowledge vs. strategy) is known is yet another dimension, class 2 and 2a would occupy approximately the same position in Figure 9.2, as in both cases only one of the two types of knowledge is known.

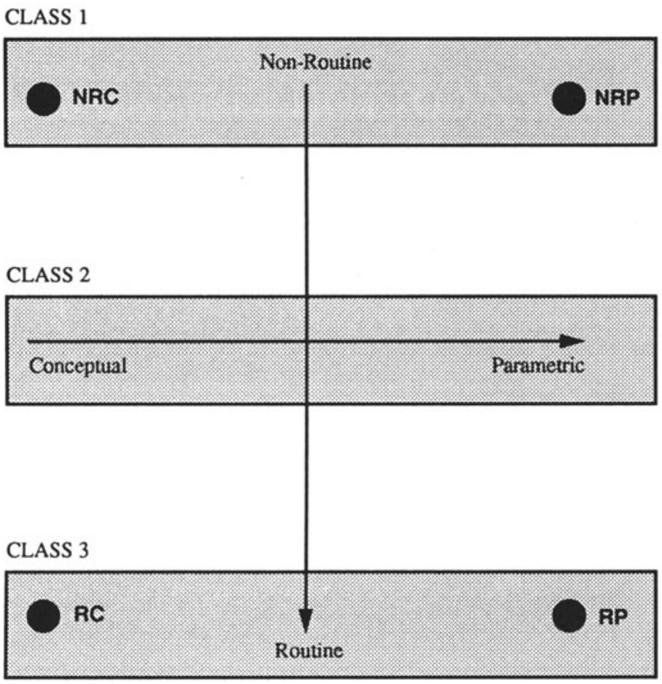


FIGURE 9.2. Three classes of design activity.

9.6.1. Confusion

At the start of this chapter it was pointed out that there has been some confusion about routine design. The most common confusion is that routine design equals parametric design. One major reason for this is that it is very likely that parametric design problems are routine. This is not merely because they usually represent the most "automatic" aspects of design (i.e., the use of equations to produce values). Another reason is due to the following argument.

As stated earlier: "For many design problems, the conceptual → parametric axis represents the flow of time during the design activity, with earlier decisions falling towards the left and later decisions falling towards the right." Because of the impact of experience, repetition of the design problem with similar but different requirements will cause the amount of reasoning needed to be reduced.

Thus, conceptual design effort will gradually be reduced, and will become unnecessary, as it will be the same or similar for every design with similar requirements. Eventually, all that will be required is parametric design, and that will become routine.

Thus, in this situation, design problems will gradually take less time and will require less reasoning. Consequently, it is natural to associate routineness with parametric design, just as it is natural to associate nonroutineness with conceptual design. This is the source of the confusion.

9.6.2. Summary

In this chapter we have examined routineness, have provided a cleaner definition, have introduced four extreme categories of design using two orthogonal axes, have related this analysis to the three classes of design, and have explained a source of confusion.

Acknowledgments

This work was supported in part by NSF grant DDM-8719960. The author would like to acknowledge the support and contributions of past and present members of the WPI AI in Design Research Group, as well as the WPI AI Research Group.

References

AAAI (1990). *AI Magazine*, Special Issue on Design, Winter, Vol. 11, No. 4, American Association for Artificial Intelligence.

Balkany, A., Birmingham, W. P., and Tommelein, I. D. (1991). A knowledge-level analysis of several design tools. In J. Gero (Ed.). *Artificial Intelligence in Design '91*. Butterworth Heinemann, pp. 921–940.

Brown, D. C. (1990). Research into knowledge-based design at WPI. In J. S. Gero (Ed.). *Applications of Artificial Intelligence in Engineering, Vol. 1, Design*. London/Berlin: Computational Mechanics Publications & Springer-Verlag.

Brown, D. C. (1991). *Compilation: The hidden dimension of design systems*. In H. Yoshikawa & F. Arbab (Eds.), *Intelligent CAD, III*, Amsterdam: North-Holland.

Brown, D. C. (1992). Design. *Encyclopedia of Artificial Intelligence, 2nd Edn.*, S. C. Shapiro (Ed.), J. Wiley, pp. 331–339.

Brown, D. C. and Chandrasekaran, B. (1985). Expert systems for a class of mechanical design activity. In J. S. Gero (Ed.). *Knowledge Engineering in Computer-Aided Design*. Amsterdam: North Holland, pp. 259–282.

Brown, D. C., and Chandrasekaran, B. (1989). *Design Problem Solving: Knowledge Structures and Control Strategies*. Research Notes in Artificial Intelligence Series, Morgan Kaufmann Publishers, Inc.

Chandrasekaran, B. (1990). Design problem solving: a task analysis. *AI Magazine*, Special Issue on Design, Winter, Vol. 11, No. 4, American Association for Artificial Intelligence, pp. 59–71.

Dixon, J. R., Duffey, M. R., Irani, R. K., Meunier, K. L., and Orelup, M. F. (1988). A proposed taxonomy of mechanical design problems. *Proceedings of the ASME Computers in Engineering Conference*. San Francisco, CA, Vol. 1, p. 41.

Finger, S., and Dixon, J. R. (1989). A review of research in mechanical engineering design, Part I: descriptive, prescriptive, and computer-based models of design. *Research in Eng. Design*, *1*(1), 51.

Gero, J. S. (1990). Design prototypes: A knowledge representation schema for design. *AI Magazine*, Special Issue on Design, Winter, Vol. 11, No. 4, American Association for Artificial Intelligence, pp. 26–36.

Gero, J. S., and Maher, M. L., Eds. (1989). *Proc. of the Workshop on Modeling Creativity and Knowledge-Based Creative Design*. University of Sydney.

Hayes-Roth, B. (1990). Three topics for discussion. *Working Notes for the Workshop on Creating a Scientific Community at the Interface between Engineering Design and AI*. Engineering Design Research Center, Carnegie-Mellon University, Pittsburgh, PA, p. 12.

Joskowicz, L., Williams, B., Cagan, J., and Dean, T. Eds. (1992). Symposium: Design from Physical Principles, Working Notes, AAAI Fall Symposium, October, Cambridge, MA.

Maher, M. L., and Fenves, S. J. (1985). HI-RISE: A knowledge-based expert system for the preliminary structural design of high rise buildings. Report No. R-85-146, Dept. of Civil Engineering, Carnegie-Mellon University, Pittsburgh, PA.

Mittal, S., Dym, C. L., and Morjaria, M. (1986). PRIDE: An expert system for the design of paper handling systems. *IEEE Computer Magazine*, Special Issue on Expert Systems for Engineering Problems.

Orelup, M. F., Dixon, J. R., Cohen, P. R. and Simmons, M. K. (1988). Dominic II: Meta-level control in iterative redesign. *Proc. 7th National Conf. on Artificial Intelligence*, AAAI, St. Paul, MN.

Ramachandran, N., Shah, A., and Langrana, N. A. (1988). Expert system approach in design of mechanical components. *Proc. ASME Int. Computers in Engineering Conf.*, San Francisco, CA.

Snavely, G. L., Pomrehn, L. P., and Papalambros, P. Y. (1990). Toward a vocabulary for classifying research in mechanical design automation. *First International Workshop on Formal Methods in Engineering Design, Manufacturing and Assembly*.

Sriram, D., and Tong, C. (1990). AI and Engineering Design. Notes for Tutorial WP2, *AAAI-90*: *the 8th Nat. Conf. on Artificial Intelligence*, Boston, MA.

Tomiyama, T. (1990). Intelligent CAD Systems. Notes for Tutorial, *Eurographics '90*, Montreux, France.

Ulrich, K. T., and Seering, W. P. (1989). Synthesis of schematic descriptions in mechanical design. *Research in Engineering Design*. New York: Academic Press, Vol. 1, No. 1, p. 3.

Waldron, M. B. (1990). Understanding design. In H. Yoshikawa & T. Holden (Eds.). *Intelligent CAD, II*. Amsterdam: North-Holland, pp. 73–87.

10
A Comparative Analysis of Techniques in Engineering Design

SRIKANTH M. KANNAPAN AND KURT M. MARSHEK

Abstract. This chapter describes the application of seven approaches to support three basic task types of design (design selection, parametric design, and design synthesis). The specialization of these approaches to practical design techniques is analyzed and illustrated with examples.

1. Introduction

The engineering design process critically influences factors of lead time and cost in product development; factors that frequently determine the profitability of products in competitive markets. To reduce the lead time and cost of product development, it is important to first characterize the design process so that new technologies and methodologies can be developed to improve its efficiency.

Characterization of the design process requires both a *natural process* view of design and an *artificial process* view of design (Kannapan and Marshek, 1992a). The natural process view emphasizes cognitive and social processes such as identification of customer needs, and team design with multiple perspectives. The artificial process view emphasizes symbol representation and manipulation processes such as modeling, computation, and reasoning. Development of design methods, tools, and environments to aid design teams require careful attention to how the two views interact and complement each other.

In this chapter we focus on basic task types in design from an artificial process view and analyze the application of alternative approaches to support the tasks. The organization of the chapter is as follows. Section 2 characterizes *task types* of design selection, parametric design, and design synthesis. Section 3 summarizes alternative *approaches* to support these tasks. Sections 4, 5 and 6 analyze and compare specializations of these approaches into practical design *techniques*. Selection of mechanical transmissions, parametric design of a relief valve, and synthesis of a rotary actuator are used as illustrative examples. Section 7 summarizes the analysis.

2. Basic Task Types in Design Activity

Three basic task types in design are identified here (Kannapan and Marshek, 1992a). The overall goal of the three tasks is as follows: given a description of a design (requirements), produce new descriptions of the design (implementations) by making decisions that ultimately reduce the space of possible realizations of the design. The task types are illustrated in Figure 10.1 and characterized below:

Design Selection

This task involves selecting a design object that satisfies design requirements from a specified set of alternatives. Design selection requires the knowledge of attributes of the alternatives, attributes defining requirements, and choice criteria to define optimality in selection. The term design object is used in a general sense, covering, for example, the selection of a working principle for a device, a material type for a component, a functional module, or a completed design.

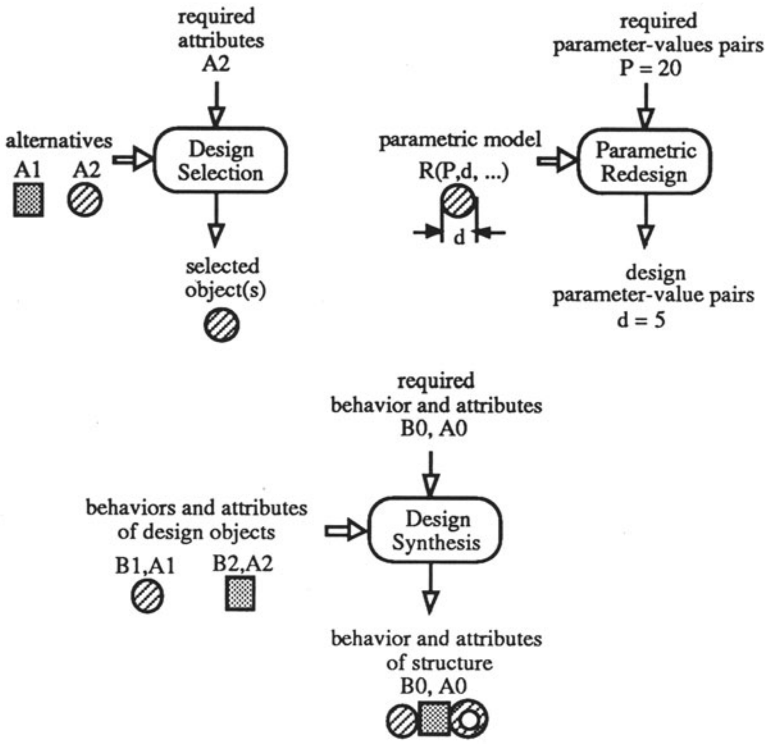


FIGURE 10.1. Basic design task types illustrated with a beam cross-section example; A's are attributes; P,d are parameters; R is a relationship; B's are behaviors.

Parametric Design

This task involves determining values or sets of values for a specified set of variables, called design parameters, that define a design (geometry, material, etc.) so as to achieve optimal values for parameters that represent design requirements (behavior, cost, etc.). Values for design parameters must satisfy constraints among variables arising from specialized engineering and other disciplines. The form of the constraints varies in complexity from heuristics, to logical relations, to partial differential equations.

Design Synthesis

This task involves configuring entities such as geometric primitives, machine components, lumped parameter models, or abstracted principles of a domain to define a system structure that satisfies design requirements. Design requirements specify the required behavior of the design, and criteria for evaluation of optimality (such as minimum cost) of a system structure. Knowledge of previous designs and principles from engineering and other disciplines specify realizability constraints for behavior and structure.

Design synthesis is distinguished from parametric design in that parametric design only permits variation of values for a prespecified set of variables. In design synthesis, introduction of new entities and variables are permitted in defining a configuration of entities. (We use "configuration" and "structure," and "design synthesis" and "configuration design" synonymously.) In effect, a synthesized configuration defines the space of parametric design variation.

Real-world design activity usually involves multiple instances of all three task types at different levels of abstraction and decomposition, with complex interactions among them. For example, in designing an automobile, the headlights might be selected from vendor catalogs, the engine and transmission might be selected from manufacturing lines of different manufacturing divisions, wheel hubs and bumpers might be parametrically redesigned, while a new anti-lock brake might be synthesized using first principles and knowledge of previous designs. Wheel hub design and brake design interact strongly in this case.

3. Approaches to Design

Table 10.1 summarizes a variety of approaches for supporting design processes in general; independent of the types of tasks described in the previous section. The following three sections analyze how these approaches can be combined and specialized to produce practical techniques for supporting tasks of design, selection, parametric design, and design synthesis.

TABLE 10.1. Approaches to supporting design processes, and theories and techniques offered (Kannapan and Marshek, 1992a).

Approach	Theories and techniques offered
Algorithmic	Finite deterministic processes: —equation solving —optimization —grammars and language compilation
Axiomatic	Axiomatization of general intuitively powerful design guidelines; proof of design theorems.
Database	Logically centralized design models and a collection of design processes: —relational, hierarchical and network data models —object hierarchy, methods, inheritance —blackboards, demons, message-passing
Machine learning	Knowledge acquisition from instruction, examples, analogy, observation, and experimentation: —inductive generalization —explanation-based generalization —case-based reasoning
Problem solving/planning	Knowledge representation; reasoning: —recursive problem decomposition —state-space search, search control strategies —rule-based and model-based reasoning —constraint reasoning —blackboard systems
System science	Identification and modeling of system, environment and interactions: —black box theory, state theory, component integration theory —decision theory —task planning —hierarchical control
Transformational	Language transformation and translation: —logical expressions and inference rules —algebraic expressions and rewrite rules

4. Design Selection

The basic mechanism needed for design selection is that of attribute matching. A design is selected when the attributes of a design in a design library match the required attributes optimally, according to specified criteria. The deeper issues in supporting the executive mechanism are (a) how design objects and their attributes are represented and organized in the design library, and (b) what processes are used to partially or totally match required attributes to the attributes of stored objects.

We will use an organization of rotary power transmissions (Kannapan, Marshek, and Gerbert, 1991a) as an illustrative example. Figure 10.2 shows an hierarchic organization of classes of transmissions organized on the basis

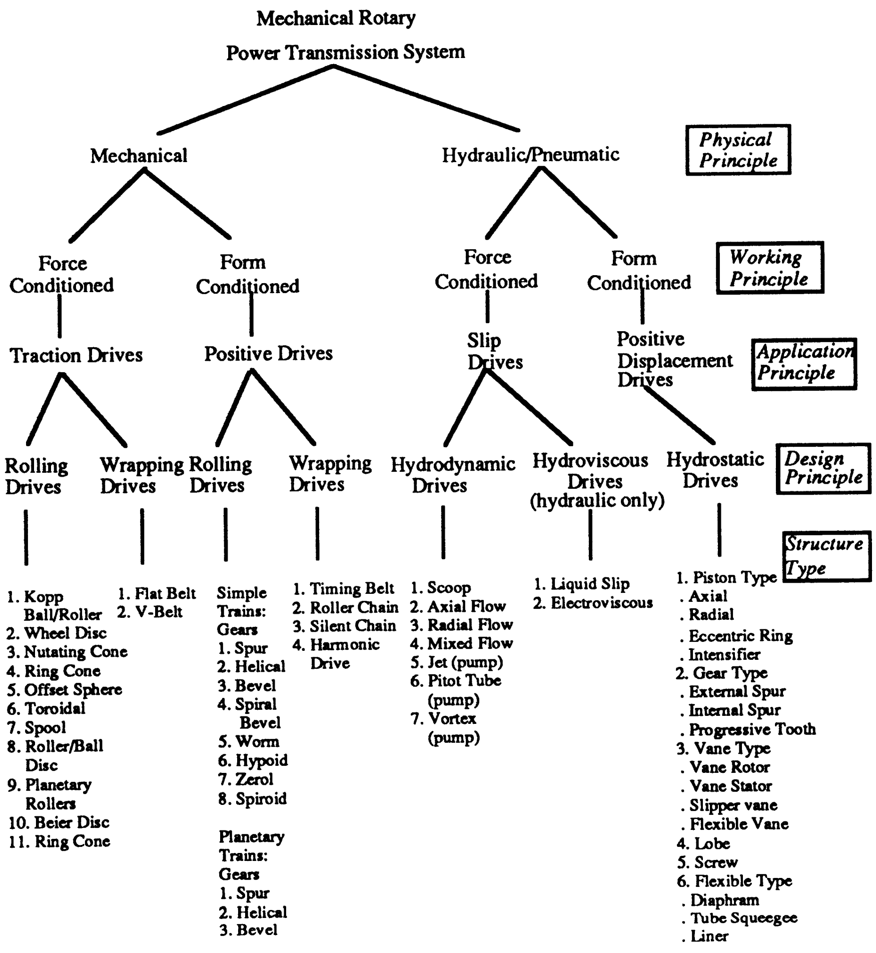


FIGURE 10.2. A classification of rotary power transmission systems.

of physical principles, working principles, application principles, design principles, and structure types.

Developing such an organization of objects requires detailed knowledge of engineering principles and prototypical artifacts of a domain as well as the choice of bases for classification. For example, in Figure 10.2, the term *form conditioned* refers to cases where the working principle for transmission of mechanical power relies on the physical form of the transmission components, and the load normal to their contact surface. The term *force conditioned* refers to cases where the working principle for transmission of mechanical power relies on the sliding friction or fluid force arising as a result of the load normal to the contact surface between transmission components. This distinction is useful in correlating structural properties with behavioral properties of mechanical transmissions.

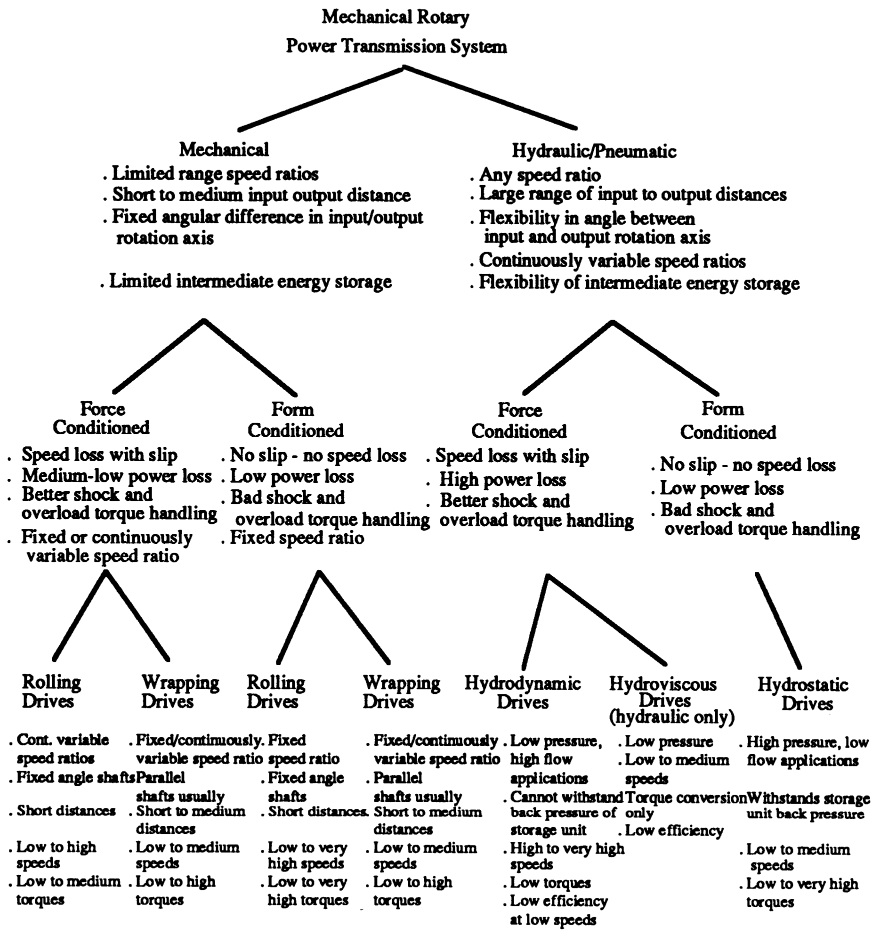

Mechanical Rotary Power Transmission System
Mechanical
. Limited range speed ratios
. Short to medium input output distance
. Fixed angular difference in input/output rotation axis
. Limited intermediate energy storage
Hydraulic/Pneumatic
. Any speed ratio
. Large range of input to output distances
. Flexibility in angle between input and output rotation axis
. Continuously variable speed ratios
. Flexibility of intermediate energy storage
Force Conditioned
. Speed loss with slip
. Medium-low power loss
. Better shock and overload torque handling
. Fixed or continuously variable speed ratio
Form Conditioned
. No slip - no speed loss
. Low power loss
. Bad shock and overload torque handling
. Fixed speed ratio
Force Conditioned
. Speed loss with slip
. High power loss
. Better shock and overload torque handling
Form Conditioned
. No slip - no speed loss
. Low power loss
. Bad shock and overload torque handling
Rolling Drives
. Cont. variable speed ratios
. Fixed angle shafts
. Short distances
. Low to high speeds
. Low to medium torques
Wrapping Drives
. Fixed/continuously variable speed ratio
Parallel shafts usually
. Short to medium distances
. Low to medium speeds
. Low to high torques
Rolling Drives
. Fixed speed ratio
. Fixed angle shafts
. Short distances
. Low to very high speeds
. Low to very high torques
Wrapping Drives
. Fixed/continuously variable speed ratio
. Parallel shafts usually
Short to medium distances
. Low to medium speeds
. Low to high torques
Hydrodynamic Drives
. Low pressure, high flow applications
. Cannot withstand back pressure of storage unit
. High to very high speeds
. Low torques
. Low efficiency at low speeds
Hydroviscous Drives (hydraulic only)
. Low pressure
. Low to medium speeds
Torque conversion only
. Low efficiency
Hydrostatic Drives
. High pressure, low flow applications
Withstands storage unit back pressure
. Low to medium speeds
. Low to very high torques

FIGURE 10.3. Attributes of classes of rotary power transmissions.

Design object hierarchies like Figure 10.2 provide a framework for attaching attributes of function, performance, and cost to design objects. Figure 10.3 shows some of the attributes of rotary transmissions attached to corresponding nodes of the class hierarchy in Figure 10.2. To associate attributes to design objects, modeling decisions must be made as to what attributes are of interest that cover a range of requirement descriptions at an appropriate level of abstraction. For example, an important behavior characteristic of hydraulic/pneumatic physical principles that must be represented is that almost any speed ratio of rotary motion within some range can be achieved because of the ease of storage of the intermediate fluid energy and control of the fluid pressure and flow rate by means of valves.

There is a disadvantage in representing design alternatives as class hierarchies with associated attributes as in Figures 10.2 and 10.3. The representation is biased towards a selection process that considers attributes in a specific order starting from the root of the class hierarchy. For example, speed ratios must be considered before speed loss, and physical principles must be selected before working principles. Biases can be avoided by representing alternatives simply as flat structures as in a relational database, but at a cost of reduced search efficiency where the bias is appropriate. Representation of multiple class hierarchies on top of flat representations retain advantages of both but at the cost of additional storage and maintenance requirements.

The question now is how required attributes can be matched to attributes of design objects to enable selection. Class hierarchies described above can be used to incrementally narrow the space of alternatives in all the techniques described below.

Decision Theory

A decision problem (system science approach) can be posed and solved by weighting the relative importance of required attributes, evaluating alternatives by ranking their degree of acceptability with respect to each attribute, and computing their cumulative rankings by a formula such as a weighted sum. See Kuppuraju, Ittimakin, and Mistree (1985), for example. If the design objects are hierarchically organized as described above, design objects can be searched incrementally starting from the root(s) of the hierarchy. For example, using Figure 10.3, a large input–output distance may be ranked as the most important attribute to consider for an application which may narrow possibilities to hydraulic/pneumatic drives. Next, no-slip requirements may limit selections to hydrostatic drives. Evaluation of alternative structure types of hydrostatic drives by weighted attributes of speed and torque ranges, cost, and reliability may result in the selection of geared hydrostatic drives.

The computation of a cumulative ranking for an alternative is simple by this technique and works well with qualitative information. However, this technique typically presumes independence between attributes and relies on designer intuition in assignment of weights and ranks, although methods exist for normalizing weights and eliminating biases when ranking alternatives on multiple attributes.

Classification Rules

Techniques from expert diagnostic systems (problem solving/planning approach) can be used in design selection. One technique is to encode classification knowledge as "if–then" rules. See Sim and Chan (1991), for example. Activation of the rules with respect to a class hierarchy of design objects incrementally reclassifies the requirements so that it migrates from the most general to the most specific subclass that satisfies the requirements.

Domain-specific rules can be organized by attaching sets of rules relevant to a subclass to the corresponding object in the class hierarchy. For example, from requirements for a rotary transmission with a tolerable speed loss of 0.5%, the activation of a rule attached to the class of mechanical physical principles, such as "if speed-loss tolerance is $<1\%$, then reclassify as form-conditioned drive," can reclassify the requirements under the form-conditioned subclass.

Another technique is to implement explicit choice rules which "rule-in" or "rule-out" design objects on the basis of their attributes. See McDermott (1978), for example. An example of a choice rule for transmissions is "if input–output axis angle = 90 deg, then rule-in worm-gear pair." Rule-based selection techniques successfully encode selection knowledge of specialized design domains, although the acquisition of the rules can be laborious and their ranges of validity tend to be narrow.

Query Languages

Query languages (database approach) can be used to select objects from relational, object-oriented, and network data models. For example, it is possible to query a database for rotary transmissions that have attributes of speed-ratio in the range 10 to 20 and have power losses less than 5%. See Eastman and Bond (1991) for an example of a design data model. Query languages provide means for indexed retrieval of design objects but do not reason with selection knowledge. Also, query operations are mathematically well founded and closed only for relational data models.

Index-transformation and Analogy

When design requirements do not exactly match an index, rules of index transformation (machine learning approach) can be used to reformulate the design requirement into other requirements that can be exactly matched by indexes. For example, if no transmission fits in the available space for an application, an index transformation rule for space partitioning may result in the selection of a piston pump at the input and a axial-flow motor at the output by instantiation of objects under different subclasses. See Hundal (1990) and Navinchandra, Sycara, and Narasimhan (1991) for examples of this technique.

Instead of transforming the index, analogical reasoning provides a means of associating terms used across different design contexts, and analyzing a causal explanation of required functions so as retrieve a design whose function is "close" to what is required. For example, knowledge of the context and manner of use of a transmission for a hand-drill suggests attributes of low input power and short distance power transmission, thus eliminating hydraulic/pneumatic transmissions. See Dyer, Flowers, and Hodges (1986) and Goel and Chandrasekaran (1990) for examples of this technique.

The ability to select design objects by transformed indexes and by "close" matches is potentially of great value since design objects exactly matching the requirements might be unavailable. Furthermore, requirements themselves might be negotiable or subject to change. However, rules for index transformation and "closeness" tend to be domain- and context-specific.

Interval and Qualitative Reasoning

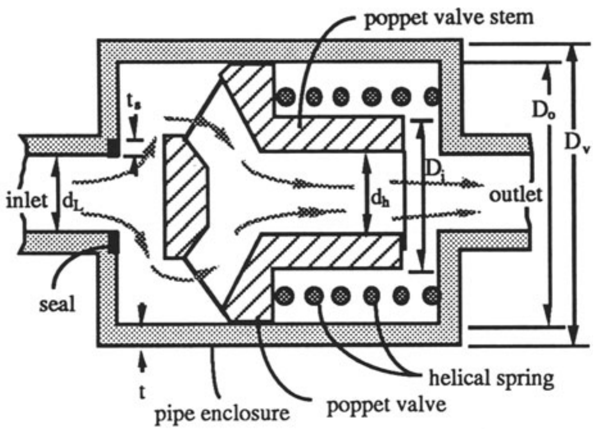
Interval reasoning (algorithmic) and qualitative reasoning techniques (problem solving/planning) can be used to represent sets of design alternatives concisely. For example, requirements on operating speed and power may be specified as intervals such as [100, 2000] rpm and [50, 500] W that implicitly represent sets of relevant design objects in a library of transmissions. The sets of alternatives can be incrementally narrowed either by specifying intervals on additional attributes or by computing tighter intervals on attributes by functional composition. See Ward (1989) for an example of this technique. Alternatively, qualitative representation of machine behavior together with simplification and abstraction operators can dynamically classify designs for selection based on their behavior properties (Joskowicz, 1990). For example, simplification and abstraction of configuration space representations of the kinematics of transmissions can be used to dynamically create an equivalence class of self-locking transmissions.

The advantage here is that requirements can be specified and interpreted at varying levels of abstraction. However, interval representations are direct only when attributes can be modeled as variables with ordered values such as real numbers; while qualitative representations of sets of values may not narrow alternatives sufficiently for selection.

5. Parametric Design

The abstract formulation of a parametric design task follows a familiar pattern:

1. a symbolic model of a prespecified design configuration is created where parameters defining the design, the design requirements, the context of use, and other attributes of interest in the product life cycle are identified;
2. constraints and goals from domain theories and processes of manufacturing (and other product life-cycle concerns) applicable to the components and subsystems are used to derive a mathematical model;
3. from known parameter values representing design requirements, the mathematical model is solved to determine optimum or acceptable values for unknown parameters defining the design, its behavior and its attributes. In special cases, parameters defining the design are directly computable, while in most cases they have to be estimated initially, and iteratively improved by evaluating its effects against requirements.



A	pipeline cross-sectional area (in.*in.)	Lf	helical spring free length (in.)
C	helical spring index	n	helical spring factor of safety
Cv	valve configuration factor	N	number of helical spring coils
Cf	orifice co-efficient	P	maximum fluid pressure in pipeline (psi)
d	helical spring wire diameter (in.)	Pc	cracking pressure (psi)
do	orifice diameter (in.)	P	pressure drop from valve inlet to outlet (psi)
dL	flow line diameter (in.)	Q	fluid flow rate (gal/min)
D	mean helical spring diameter (in.)	S	fluid specific gravity
Fc	cracking force on helical spring (lb)	Sp	allowable stress for pipe material (psi)
Fd	dynamic fluid force (lb)	t	pipe thickness (in.)
Ft	total force on helical spring (lb)	ts	seal thickness (in.)
G	shear modulus of spring material (psi)	tv	valve cylinder thickness (in.)
ks	computed spring rate (lb/in.)	δ	max deflection of valve and spring due to fluid force (in.)
kact	actual spring rate (lb/in.)	τ	helical spring maximum stress (psi)
K	flow co-efficient (head loss factor)	τ_{all}	allowable helical spring material stress (psi)
Kw	Wahl spring factor		

FIGURE 10.4. Schematic and selected parameters of a relief valve design.

We will use a problem of relief valve design (Lyons, 1982) as an example to illustrate the application of alternative approaches to the above formulation of the parametric design task. Figure 10.4 shows a schematic of the relief valve configuration. When the pressure of a fluid at the inlet of the valve equals or exceeds a specified "cracking" pressure, the fluid pushes open the poppet valve and flows to the outlet while holding the valve in force equilibrium against a helical compression spring. At pressures below the cracking pressure, the valve is pressed against a seal by the spring and fluid flow is cut off.

The first step of formulation of the parametric design problem for the relief valve identifies parameters defining the design (e.g., geometry and material properties), the context of use (e.g., specific gravity of fluid, flow rate, acceleration due to gravity), requirements (e.g., cracking pressure, factor of safety), and behaviors of interest (e.g., natural frequency of valve vibration). Some of the parameters are shown in Figure 10.4.

The second step in this case uses domain theories of fluid mechanics and solid mechanics to develop a mathematical model that constrains the parameters through relationships. In general, a variety of models can be developed at different levels of abstraction and approximation of the domain theories with consequences on the tractability of the solution process and the accuracy of results. The key to this choice is to develop the most efficient model (in terms of cost and time for model creation and solution) that validates the achievement of requirements within an interval of tolerance and confidence. This process in practice is based on specialized human expertise, experience and intuition. Development of computational tools for modeling is an area of research—see Falkenhainer and Forbus (1991) for example. Once a mathematical model is formulated, a variety of techniques may be applied as described below.

Analytical/Numerical Methods

A direct algorithmic approach can be applied where all the constraints are collected and unknown parameter values are solved from given parameter values based on standard analytical (e.g., algebraic manipulation, substitution) and numerical methods (e.g., gradient methods, finite difference, and finite element methods).

However, four issues need to be addressed in scaling up such direct solutions:

1. the intractability of simultaneously satisfying multiple goals and solving large numbers of constraints of varying form and complexity (some may be as simple as rules of thumb and curve-fitted data while others may be partial differential equations that require specialized solution techniques);
2. the distributed nature of specialized knowledge on parametric models and solution methods; creation of a centralized collection of parameters, constraints, goals, and solution methods is impractical;
3. the need to solve problems in parallel to reduce design time;
4. the need for design decisions; when context and domain information does not fix a sufficient number of parameters and leads to under-constrained formulations with large solution spaces.

Task Decomposition

Intractability of simultaneous solution of constraints can be avoided by problem decomposition (problem solving/planning approach). Subproblems are defined on the basis of clustering of goal and constraint knowledge in the domain. For examples, see Brown and Chandrasekaran (1983), Kannapan and Marshek (1992b), and Bowen and O'Grady (1990).

In relief valve design, clustering of constraint knowledge corresponds to subproblems of valve-flow, valve-cracking, helical-spring, and pipe-enclosure design when patterned after different chapters in a relief valve design handbook (Kannapan and Marshek, 1992b; Lyons, 1982)—see Figure 10.5. In-

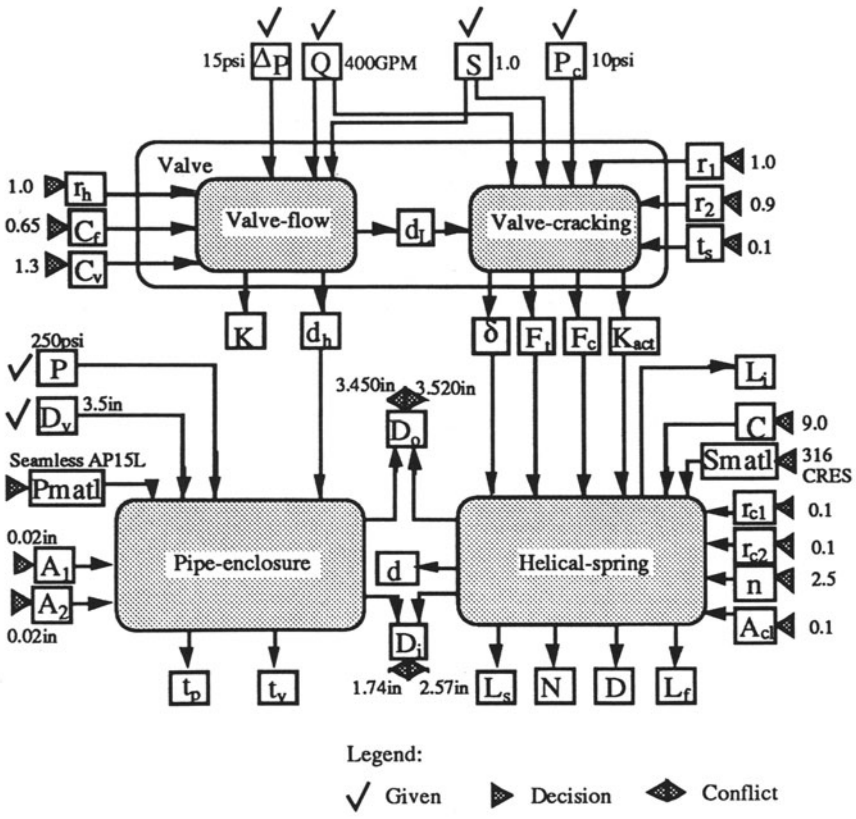


FIGURE 10.5. Relief valve design problem decomposition.

teractions between subproblems are represented by shared parameters. To handle underconstrained subproblems, parameters that are decided by human designers according to typical design procedures are identified, and preferred values encoded as default values or utility functions (Kannapan and Marshek, 1992b).

An execution scenario that solves the valve-flow subproblem first, then the pipe-enclosure subproblem in parallel with the valve-cracking and helical-spring subproblems results in values for parameters shown in Figure 10.5 for a representative design. The arrows indicate directions of data flow. The subproblem solution processes themselves can take a variety of standard forms (e.g., symbolic algebra, numerical methods, constraint propagation) and are not considered here.

One issue that arises here is that of planning and controlling an efficient execution order for subproblem solutions. Search techniques from problem-solving/planning approaches, task management techniques from system science (e.g., PERT, CPM), and optimization techniques such as dynamic programming are applicable. As an alternative to pre-planning the order of solution, blackboard systems can be used to execute design processes opportunistically when triggered by value binding events of parameters. See Sriram et al. (1991) for an example.

Another issue that arises in managing solution processes is conflict detection and resolution. The individual decisions made by designers in this example lead to conflicts in values for the parameters D_i and D_o. One possible protocol to negotiate such conflicts is to propagate utility functions on decision parameters to the conflict parameters, and to propose agreement values based on the propagated utility functions and axiomatic theories of bargaining—Kannapan and Marshek (1992b) contains details. The resolution of conflicts by negotiation can also be formulated and solved as case-based (machine learning) or rule-based inference (problem solving/planning). See Sycara (1990), Klein (1992), and Lander (1989) for examples.

Optimization

A variety of multi-objective optimization techniques can be applied by associating objective functions to individual subproblems, and ranking or weighting the relative importance of objectives. For example, the objective of maximizing the factor of safety for the helical-spring may be ranked higher than the objective of minimizing the external diameter D_v of the pipe-enclosure. See Gero (1985) and Karandikar et al. (1989) for example techniques.

Qualitative reasoning and monotonicity analysis can be combined with optimization techniques to guide the search for improved objective function values. For example, we can determine by qualitative reasoning and monotonicity analysis of constraints and utility functions that both the helical-spring solution process and the pipe-enclosure solution process drive the value of the shared parameter D_i in the same direction (Kannapan and Marshek, 1992b). We can then reason with their active constraints to determine that the corrosion resistance constraint for the pipe-enclosure determines the optimal value of D_i. See Agogino and Almgren (1987) for an example of this technique.

Optimization techniques work well for linear objectives and constraints but find it difficult to search beyond the neighborhood of point design solutions when nonlinearities are involved. Techniques like simulated annealing attempt to "jump" beyond local optima but their processes and results are probabilistic.

Coordinated Subspace Optimization

Optimization methods can be combined with system science techniques to coordinate the solution of coupled subproblems. Subproblems interacting through shared parameters can be coordinated with the solution of the global problem by exchanging parameter sensitivity information through penalty functions. For example, if a goal of the pipe subproblem is to minimize the external diameter of the valve, and the spring subproblem has a goal of keeping the spring index ($C = D/d$) close to 9, a penalty function for the spring subproblem incorporates the sensitivity of the spring index to the diameter of the valve. Terms can also be added to the constraints of the spring subproblem to incorporate penalties for violating constraints of the pipe subproblem, and vice versa. These penalties include coefficients modeling the responsibilities and trade-offs of different subproblems in achieving objectives and satisfying constraints. The responsibility and trade-off coefficients are themselves optimized using sensitivity information from the subproblems. See Sobieszczanski-Sobieski (1988) for an example of this technique.

An alternative to using penalty functions is to coordinate solution processes by hierarchic model-based or feedback control: (a) input parameters and default values for control variables of the solution processes are used to initiate subproblem optimizations, (b) outputs and process variables are sensed, and (c) control variables are updated by a control law that uses sensed information or models of the process. For example, the shared variable D_o may begin with a default value, then the pipe enclosure design and spring design subproblems optimize on their local goals. After a specified threshold of resource usage or objective improvement rate is sensed, the sensitivities of the objectives with respect to D_o are estimated. A control law updates the value of D_o using, for example, an average value of the sensitivities so as to initiate another cycle of subproblem optimizations. See Bell, Kannapan, and Taylor (1992) for an example.

While the basic limitations of optimization techniques still hold, techniques for coordinated subspace optimizations handle subproblem interaction effectively, and permit parallel subproblem solution. If the dynamics of solution processes are analyzed and possibly predicted, control loops can maintain process stability and improve process efficiency.

Set-based Reasoning

An alternative to propagating values and utility functions is to propagate sets of values between subproblems either as intervals or fuzzy sets. For example, Wood and Antonsson (1990) propagate imprecision (uncertainty in choice of values) in parameters based on a numerical method (fuzzy weighted average algorithm), while Ward (1989) develops a labeled interval propagation calculus for propagating value intervals through functions. For relief-valve design, by these techniques, the effect of choosing different materials for the spring wire on the outside diameter of the valve (D_i) can be represented by computing a set of values for D_i from a set of values for spring wire strength.

Propagating sets of values has advantages of being able to compactly represent and reason with a subspace of instantiated designs, although some approximations have to be introduced into constraint reasoning. On the other hand, value and utility propagation do not introduce approximations but are only capable of reasoning in the neighborhood of a point design solution.

Constraint and Rule-based Reasoning

Instead of an optimization formulation where subproblems are completely formulated before solution, a "least commitment" style of design can be supported using a problem-solving/planning approach by handling interactions between subproblems as constraints to be formulated and propagated (Stefik, 1981; Sussman and Steele, 1980). For example, with the given parameter values in Figure 10.5, if constraints of different subproblems are introduced incrementally, constraints may be evaluated to either determine some parameter values from givens, or to symbolically manipulate, propagate and simplify other constraint expressions. When constraints can no longer be propagated, the location and ordering of design decisions that will restart constraint propagation can be precisely identified.

However, this technique becomes infeasible when constraint expressions are too complex to symbolically propagate through other constraints. Where constraint propagation is infeasible, parametric design heuristics can be encoded as implementation rules or sensitivity rules using a problem-solving/planning approach. For example, an implementation rule for designing the orifice of the relief valve may be encoded as "if sharp-edge orifice is selected and pressure-ratio < 0.9, then set orifice coefficient $(Cf) = 0.65$." Alternatively, a sensitivity rule can specify that a certain fractional change in orifice coefficient is expected to result in a certain fractional change in pressure ratio. For examples of these techniques see Dixon and Simmons (1984) and Dixon et al. (1987). Heuristic implementation rules tend to have limited ranges of validity and unforeseen interactions with other heuristics and constraints.

Rule Generalization and Process Replay

Machine learning techniques can be used to generalize implementation rules acquired through experience, and to adapt and replay existing design process plans.

Deductive generalizations of implementation rules are based on a formal explanation of how the implementation (the "then" part) satisfies the requirements (the "if" part). The generalized "then" part comprises those preconditions that must be met for the explanation to hold. The generalized "if" part comprises the specification explained by the generalized "then" part together with preconditions for proper component behavior. An example of generalization of "if sharp-edge orifice is selected and pressure-ratio < 0.9 and flow rate < 500 gal/min, then set orifice coefficient $(Cf) =$

0.65" would be to replace the flow rate condition to that of "laminar flow." Such a generalization would be valid if a formal explanation of how the choice of $Cf = 0.65$ satisfied requirements only assumed that the flow was laminar and not necessarily that the flow rate < 500 gal/min. See Mitchell, Mahadevan and Steinberg (1985) for an example of this technique.

Histories of parametric design processes can be stored as a sequence of decisions that resulted in a successful design starting from requirements. Parameters for variation in such a process are those parameters that do not affect the acceptability of a stored decision sequence. Parametric design with new instantiations of these parameters then simply corresponds to a replay of the process history. For example, if a process history for relief-valve design corresponds to a sequence of valve-flow design, valve-cracking design, helical-spring design, and pipe-enclosure design, assumptions in the models may indicate that variation in the flow-rate within the laminar range does not affect the sequence. The process history can therefore be replayed with changed flow-rate requirements. See Mostow and Barley (1987) for an example of this technique.

6. Design Synthesis

Design synthesis is the task of configuring entities of a domain to construct a realizable system structure that satisfies design requirements. A variety of design situations are covered by defining entities in the domain at different levels of abstraction (e.g., geometric objects, lumped parameter models, functional modules, and physical principles) and decomposition (e.g., airplanes, device level artifacts, material, and chemical structures). "Original design" is subsumed by design synthesis when original designs are viewed as non-obvious combinations and utilizations of known objects, models, or principles (Kannapan and Marshek, 1991). (The *discovery* of new objects, models, or principles is not considered part of a *design* process.) In addition, design situations where a faulty or suboptimal subsystem of an existing system must be replaced by an improved one are covered by redefining the scope of the task to synthesis of the subsystem.

Since a synthesized configuration defines the space for parametric variation, it represents a substantial commitment to the cost and performance targets achievable when the design is completed. However, this task is the hardest to support by traditional applications of symbolic and numeric processing techniques. An exception is the field of digital electronics (VLSI design) where substantial progress has been made in developing synthesis techniques and tools. Although synthesis techniques for digital electronics are candidates for generalization to other domains, many of the inherent advantages of this domain are difficult to find in other domains of physical systems. Inherent advantages in the domain of digital electronics are: the number of component types are small (in fact a NAND gate by itself is functionally complete), relationships between components are simple (e.g. wires), behaviors are easily representable (e.g., boolean functions with time delays), many implementation side-effects can be controlled (e.g., by keeping wires sufficiently apart), and functional modularity is feasible and acceptable (e.g., separate implementation of memory and processing functions). Some of the techniques applicable to synthesis of physical systems beyond VLSI are described in this section.

Consider the task of synthesizing a configuration of mechanical parts to gradually actuate a flap on the wing of an airplane from a high-speed motor (MacDonald, 1973). The kinematic behavior requirement may be expressed as that of constant ratio input-output rotation between fixed angles with respect to a fixed reference frame of a housing. The MacDonald device (Figures 10.6 and 10.7) is used here as an example of a design configuration that can be synthesized to satisfy requirements. The MacDonald device uses a sliding joint between the actuator and the housing, and a helical spline between the actuator and an output member to convert rotary motion of a threaded shaft to slower rotary motion of the output member between fixed angular ranges (MacDonald, 1973)—see Figure 10.7. In what follows we neglect the fixed angular range requirement.

Four types of knowledge are involved in such a design synthesis task: (1)

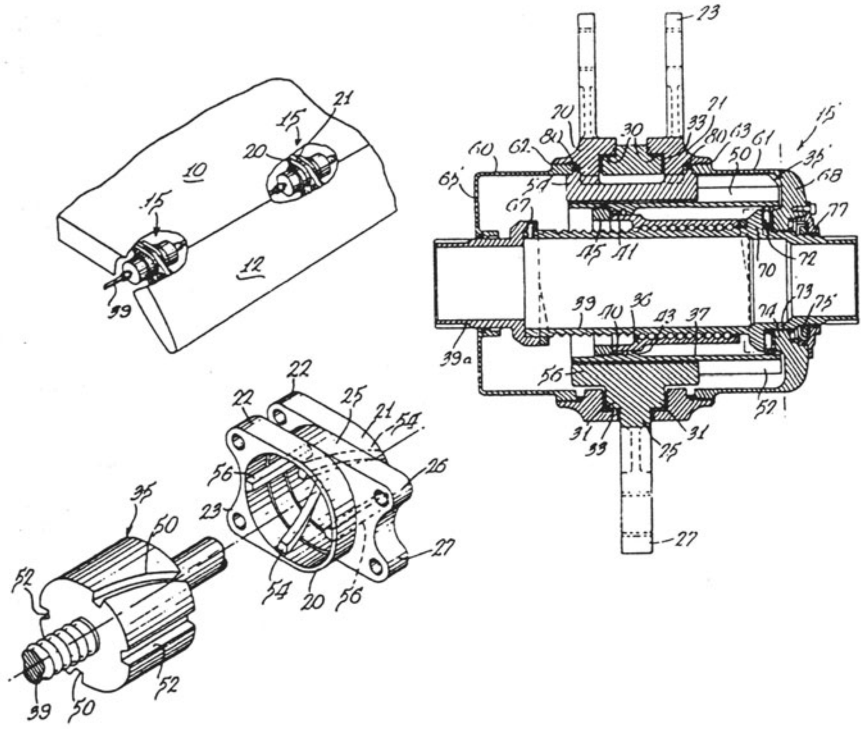


FIGURE 10.6. Schematics of MacDonald rotary actuator (MacDonald, 1973).

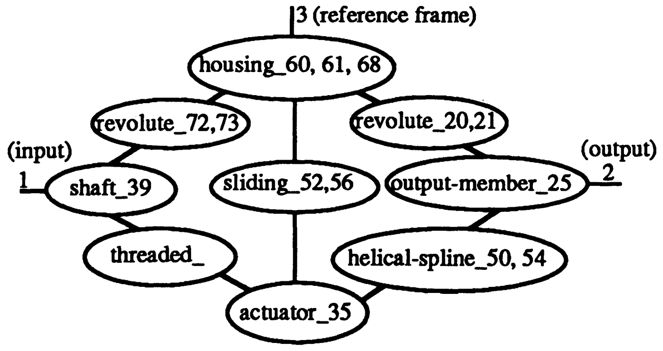


FIGURE 10.7. Structure of MacDonald rotary actuator.

knowledge of design requirements, (2) knowledge of previous successful and failed designs, (3) principles of relevant engineering and other disciplines, and (4) how to use the knowledge in (1), (2), and (3) to construct structures that satisfy design requirements. In the following discussion, we focus on techniques for (4) with only indirect reference to representation issues in (1), (2), and (3). We do not discuss control strategies required in (4).

Structure Enumeration

A direct algorithmic technique to support a synthesis task of this type is to generate structures and test them for acceptability. First, a set of component types and component relationship types are selected from a library and a fixed number of instances of the selected types are created. Second, the instantiated components and relationships are used to exhaustively generate all structures feasible by engineering principles. Third, the generated structures are evaluated with respect to design requirements. See Buchsbaum and Freudenstein (1970) for an example of this technique.

An alternative to this technique is to define a grammar that represents component types as terminal symbols of a language, and systems of components as nonterminal symbols of the language. Productions of the language (typically context-free) can be used to generate a space of allowable configurations of components and relationships. See Mullins and Rinderle (1991) for example.

For the problem of rotary actuation, a space of possible structures that contains the MacDonald device structure can be generated by selecting a few instances of rigid members and sliding, revolute, threaded, and helically splined pairs. Configuring the instances of rigid members and kinematic pairs in all kinematically feasible ways, and analyzing the resulting structures determines if any of them can be used for rotary actuation.

The advantage of this technique is that the feasibility of a structure is tested during generation. But there are two important disadvantages in this technique. First, the generative process is not directed by the goal of satisfying design requirements; although attribute grammars can help in testing preconditions before activating productions (Rinderle, 1991), and optimization methods can progressively eliminate redundant components of a structure after it is generated (Topping, 1983). Second, the types and number of instances of components to be used must be selected at the beginning of the synthesis process thus *a priori* limiting the space of structures generated.

Function/Structure Variation

One way to avoid these disadvantages is to begin with functional requirements, and combine system science and database approaches to convert the synthesis task to selection and variation tasks. First, a design library is created where designs are represented hierarchically as functional block diagrams. Components are associated with the functional blocks. Second, a system satisfying requirements is selected from the database if possible. If this is not possible, the technique resorts to the user for interactive construction of a system from available functional blocks and associated components.

Selection or construction of one design satisfying requirements affords possibilities for synthesis of other designs by function and structure variation. Function variation systematically replaces one or more functional blocks by others of identical function, while structure variation replaces one or more components in the structure by others of identical function. For example, from the MacDonald device, other devices may be generated by exchanging functions of the sliding and helically splined pairs, or by choosing different bearing types and spline profiles. See Hundal (1990) for an example of this technique.

The space of structures that one can generate by this technique is limited by the initial configuration that is selected or constructed as well as the functional blocks represented in the library. Opportunities for implementing several functional blocks with one component (function sharing) are difficult to exploit unless they are already implicit in the function block definition. Also, physical realizability of a generated configuration cannot be ensured.

Case Adaptation

The need for exact matching of requirements in function/structure variation can be relaxed by retrieving partial matches and "close" matches as described earlier for design selection. A retrieved design case can then be adapted if possible to satisfy requirements.

When multiple component selections are made through index transformation rules, the unification of index variables in rules implicitly determine how selected components can be configured and adapted. For the rotary actuator synthesis problem, if indexed retrieval of rotation-to-rotation conversion fails, transformed indexes of rotation-to-translation conversion and translation-to-rotation conversion implicitly specify a rigid connection between the output of the threaded pair and the input of the helically splined pair. See Navinchandra, Sycara, and Narasimhan (1991) for an example of this technique.

When design objects are retrieved by analogical reasoning, the differences between the requirements and the capabilities of the retrieved design can be analyzed on the basis of functional and causal representations of the design and its behavior. Results of this analysis are used to augment or mutate the design using context specific plans and domain specific rules. For example, it is possible to reason that rotation-to-rotation conversion is "close" to existing capability of rotation-to-translation conversion by means of a threaded pair. Augmenting a threaded pair to satisfy requirements in this case involves detecting that a translation-to-rotation conversion is also required, selecting the helical-spline for this purpose, and adding it to the threaded pair with appropriate interfaces. See Dyer, Flowers and Hodges (1986), Goel and Chandrasekaran (1990), and Murthy and Addanki (1987) for examples of this technique.

Adaptation techniques exploit knowledge of past designs without necessitating initial selection by exact matching. However, the emphasis remains on reusing known designs and not on synthesizing new designs using task and domain knowledge.

Task Decomposition

Task and domain knowledge can be exploited by using a problem-solving/planning approach. The overall synthesis task is recursively decomposed into primitive tasks that are directly solvable, and the overall solution is composed from the primitive solutions. The decomposition is on the basis of the structure of task knowledge represented as goals, constraints, and rules. Plans are associated with tasks at each level of decomposition to control the execution of its subtasks.

Execution of a top-level plan applies the task knowledge of rules and constraints by means of rule-based inference engines or constraint propagation tools. The primitive subtasks select design objects from a library, or parametrically redesign models of prototypical artifacts. The design objects and models themselves may be organized separately as class hierarchies.

The results of executing lower level plans are composed by higher level plans while incorporating constraints that incorporate intertask couplings at each level of decomposition. In special cases it may be possible to order subtasks that extending partial design configurations so that little or no backtracking is necessary (McDermott, 1982). In general, plans may fail. When a plan fails, attempts are made to repair the plan, or alternative plans are explored by backtracking. Kota and Lee (1990b) and Maher and Fenves (1985) are examples of this technique.

For example, based on task knowledge, one decomposition of the task of synthesizing a rotary actuator that corresponds to the MacDonald device is: input–rotation–support, rotation–to–translation-conversion, translation–to–rotation-conversion, output–rotation–support. Plans associated with each of these subtasks may select bearings, threads, helical splines, and so on, while the top-level plan enforces constraints on the connections between the selected components to produce the structure of the MacDonald device.

Task decomposition and functional decomposition (used in function/structure variation and adaptation techniques) tend to be complementary; one exploiting knowledge of previously known design artifacts and the other exploiting both declarative and procedural designing knowledge. Task decomposition techniques are also goal directed and begin with design requirements.

Graph Transformation and Augmentation

Relying on task or functional decompositions implicitly limits the space of exploration to previously known decompositions of the task for which plans succeeded, or to variations and adaptations of known designs. An alternative is to modify the structure enumeration technique described earlier to make it begin with graph representations of behavioral design requirements.

The first step is to express (a) behavior requirements and primitive behavior fragments of an implementation domain as graphs (e.g., bond graphs, constraint nets) and (b) rules of graph transformation that encode properties of graph well-formedness and knowledge of the domain. The second step is to apply transformations to the requirement graph so as to enumerate all transformed and augmented graphs that retain the intent of the original requirements. Transformations match and replace subgraphs, while augmentations introduce new nodes or new paths between nodes.

The third step is to match subgraphs of the transformed and augmented graphs to behavior fragments that correspond to components and component relationships stored in a library. The connectivity among matched behavior fragments specify a configuration of components and relationships. The graph corresponding to this configuration is further augmented by the unused behaviors of the selected components so as to consider possible side-effects. The final step is to determine if the generated graph satisfies behavior requirements. If it does not, a rule-based (problem solving/planning) technique can be used to debug the design by substituting faulty fragments of the structure by other fragments.

The above steps can of course be interleaved to varying degrees depending on the control strategy. See Ulrich (1988), Prabhu and Taylor (1989), Finger and Rinderle (1989), and Williams (1989) for varieties of these techniques.

The rotary actuation design requirement can be represented as a constraint graph; where nodes are rotation velocity variables of the input, output, and housing, and a hyperedge is a relation of constant velocity ratio between input and output relative to the housing. This graph can be transformed by replacing the constant ratio rotation–to–rotation conversion requirement by fragments of behavior that convert rotation to translation and vice versa. Augmentations introduce nodes for translation velocity variables for the reference frame, input, and output; and introduce hyperedges to establish equality relationships between translation velocities. Such transformations and augmentations permit matching of subgraphs to behaviors associated with rigid members and bearing components, as well as threaded, helical-splined, and sliding relationships. Connecting components by relationships as specified by the behavior graph produces the structure of the MacDonald device.

The main advantage of graph-based techniques is that the space of structures explored is not limited by existing functional or task decompositions even though it is directed by behavioral design requirements from the start. Thus opportunities for exploiting and sharing functionality of components can be detected and realized. The costs of creating these advantages is the possibility of reinventing designs that are previously known, or designs that can be more efficiently created from known decompositions of task or artifact knowledge. Also, only main signal and power flows of required behavior are considered for structure generation in graph transformation and augmentation; many generated designs may be later found unacceptable due to destructive side-effect behaviors of selected components, or nonbehavioral design requirements.

Algebraic and Logical Transformation

Transformations and augmentations of graphs tend to be local in effect whereas transformations of symbolic expressions can make it easier to recognize and replace patterns that are noncontiguous.

Given a behavior requirement as a symbolic expression of a language, algebraic manipulation rules can be used to repeatedly transform the expression. The intent of the transformations is to enable subexpressions of the requirement to be matched to behaviors of components and relationships in a library.

Kota (1990a), for example, uses matrices to represent and transform qualitative behaviors. Application of such a matrix algebraic method to the rotary actuation example will express the design requirement as a conversion of input rotation to output rotation by means of a motion transformation matrix concatenated to constraint matrices on the forms of motions to be transmitted (e.g., one-way or reversible). The motion transformation matrix can be rewritten by rules for row/column manipulations and decompositions to match the motion transformation matrices of rigid members, and threaded, helically splined, revolute, and sliding pairs creating the structure of the MacDonald device.

Unlike graph-based and algebraic languages, predicate logic provides a basis for expressing conjunctive, disjunctive, conditional, and negated behaviors as well as type information on behavior variables in a natural manner. Inference rules provide a formal mechanism for reasoning either in the *forward* direction by producing new expressions from expressions known to be true, or in the *backward* direction by reducing a goal expression to expressions known to be true. A variety of sound and complete inference rules exist such as *modus ponens*, resolution, and natural deduction that apply to different forms of expressions. Other inference techniques such as abduction are also relevant to design processes since they allow the hypothesis of a design that logically *implies* the design requirement but is not logically *equivalent* to it. See Dietterich and Ullman (1987), Kannapan and Marshek (1991b), and Takeda, Tomiyama and Yoshikawa (1992) for examples.

Application of predicate logic-based methods to the rotary actuation example begins with the definition of a vocabulary of predicates that represent primitive behaviors of a domain. For example, kinematic behaviors of rigid members and bearing components, as well as revolute, threaded, helically splined, and sliding relationships can be defined as predicates that relate translation and rotation velocities. Logical equivalences and implications are defined to formally encode (a) composability of behaviors (e.g., bearing behavior logically equivalent to a conjunction of revolute and rigid behaviors), (b) domain theories (e.g., translation velocities add when reference frames change), (c) properties of the language (e.g., conjunction is symmetric), and (d) mathematical properties of terms (e.g., transitivity of =). A library of components and relationships with associated behaviors is defined to enable reuse of previously known designs.

Now, the required behavior is defined as a logic expression, in this case as a predicate prescribing constant ratio of rotation velocity between input and output relative to the housing. A successful sequence of applications of selected inference rules to the required behavior produces a transformed behavior expression that is a conjunction of behaviors of rigid members and bearings configured as in the MacDonald device using threaded, sliding, and helically splined relationships. Opportunities for sharing functionality of components can be exploited in this process (Kannapan and Marshek, 1991b).

Properties of algebraic and logical transformational techniques are very similar to graph augmentation and transformation techniques. One difference is that algebra and logic provide a richer language for representation and reasoning but at the cost of less tractable means for controlling the reasoning process. For logic-based languages, the existence of mathematical foundations for truth (model theory) and logical entailment (proof theory) bring problems of formally dealing with time, uncertainty, and monotonicity of inferences to the fore.

Variable Expansion and Optimization

Representation and reasoning with languages based on graphs, algebra, and logic is difficult when models of physical phenomena lead to complex mathematical expressions that require numerical methods for their solution.

Symbolic and numerical methods can be used to evolve structures from prespecified physical domains by introducing new design variables and optimizing over them. A boundary variational technique is developed by Bendsoe and Kikuchi (1988) for structural design where the design synthesis problem is posed as the determination of the optimum distribution of holes in a structural material. Cagan and Agogino (1992) show how by optimization on an expanded domain of variables structural changes in the design can be obtained. It is unclear how these techniques would apply to synthesis of the MacDonald device; but see Gupta and Jakiela (1992) on how new variables can be introduced to simulate kinematics and generate shapes by discretizing geometric boundaries.

Variable expansion and optimization techniques address the need to represent and reason with complex behavioral expressions and geometric detail when required. One disadvantage is that the methods are sensitive to the types of domains (e.g., solid mechanics, kinematics) to which they are applied. The other disadvantage is shared with methods of structure enumeration described earlier: a structural domain from which the design is to be created (e.g., a block of material) has to be specified beforehand.

7. Summary

This chapter focused on three basic types of design tasks (design selection, parametric design, and design synthesis) and seven approaches to support these tasks from an artificial process view. The specialization of these approaches into techniques to support each task were analyzed, compared, and illustrated with examples. A summary of the techniques applied to each task is given below:

Design Selection Techniques

- decision theory
- query languages
- interval and qualitative reasoning
- classification rules
- index transformation and analogy

Parametric Design Techniques

- analytical/numerical methods
- optimization

- set-based reasoning
- rule generalization/process replay
- task decomposition
- coordinated subspace optimization
- constraint and rule-based reasoning

Design Synthesis Techniques

- structure enumeration
- case adaptation
- graph transformation/augmentation
- variable expansion/optimization
- function/structure variation
- task decomposition
- algebraic/logical transformation

References

Agogino, A. M., A. Almgren, 1987, Symbolic Computation in Computer Aided Optimal Design, *Expert Systems in Computer Aided Optimal Design*, J. S. Gero (Editor), North-Holland, Amsterdam, pp. 267–284.

Bell, D. G., S. Kannapan, D. L. Taylor, 1992, Product Development Process Dynamics, *Proc. ASME Design Theory and Methodology Conf.*, Scottsdale, AZ, pp. 257–266.

Bendsoe, M. P., N. Kikuchi, 1988, Generating Optimal Topologies in Structural Design Using a Homogenization Method, *Computer Methods in Applied Mechanics and Engineering*, Vol. 71, pp. 197–224.

Bowen, J., P. O'Grady, 1990, A Technology for Building Life-Cycle Advisers, *Proc. of Computers in Engineering 1990*, Vol. 1, ASME, Boston, MA, August 5–9, pp. 1–7.

Brown, D. C., B. Chandrasekaran, 1983, An Approach to Expert Systems for Mechanical Design, *IEEE Computer Society Trends and Applications '83*, NBS, Gaithersburg, MD, May 1983, pp. 173–180.

Buchsbaum, F., F. Freudenstein, 1970, Synthesis of Kinematic Structure of Geared Kinematic Chains and Other Mechanisms, *J. of Mechanisms*, Vol. 5, pp. 357–392.

Cagan, J., A. M. Agogino, 1992, Dimensional Variable Expansion—A Formal Approach to Innovative Design, *Research in Engineering Design*, Vol. 3, pp. 75–85.

Dietterich, T. G., D. G. Ullman, 1987, FORLOG: A Logic-Based Architecture for Design, *Expert Systems in Computer-Aided Design*, John Gero (Editor), IFIP, Amsterdam, North Holland, pp. 1–24.

Dixon, J. R., A. Howe, P. R. Cohen, M. K. Simmons, 1987, Dominic I: Progress Toward Domain Independence in Design by Iterative Redesign, *Engineering with Computers*, Vol. 2, pp. 137–145.

Dixon, J. R., M. K. Simmons, 1984, Expert Systems for Design: Standard V-Belt Drive Design as an Example of the Design-Evaluate-Redesign Architecture, *Proc. ASME Computers in Engineering Conf.*, Las Vegas, NV, August 12–16.

Dyer, M. G., M. Flowers, J. Hodges, 1986, Edison: An Engineering Design Invention System Operating Naively, *Proc. of the 1st Intl. Conf. on Applications of AI to Engineering Problems*, Southhampton, U.K., Vol. 1, pp. 327–341.

Eastman, C. M., A. H. Bond, 1991, Application and Evaluation of an Engineering Data Model, *Research in Engineering Design*, Vol. 2, No. 4, pp. 185–207.

Falkenhainer, B., K. D. Forbus, 1991, Compositional Modeling: Finding the Right Model for the Job, *Artificial Intelligence*, Vol. 51, pp. 95–143.

Finger, S., J. Rinderle, 1989, A Transformational Approach to Mechanical Design Using Bond Graph Grammars, *Proc. ASME Design Theory and Methodology Conf.*, Montreal, Canada, pp. 107–116.

Gero, J. S., 1985, *Design Optimization*, Academic Press Inc., New York, NY.

Goel, A., B. Chandrasekaran, 1990, A Task Structure for Case-based Design, *Proc. IEEE Intl. Conf. on Systems, Man and Cybernetics*, November, pp. 587–592.

Gupta, R., M. J. Jakiela, 1992, Qualitative Simulation of Kinematic Pairs via Small-scale Interference Detection, *Proc. ASME Design Theory and Methodology Conf.*, Scottsdale, AZ, pp. 351–363.

Hundal, M. S., 1990, A Systematic Method for Developing Function Structures, Solutions and Concept Variants, *Mechanism and Machine Theory*, Vol. 25, No. 3, pp. 243–256.

Joskowicz, L., 1990, Mechanism Comparison and Classification for Design, *Research in Engineering Design*, Vol. 1, pp. 149–166.

Kannapan, S., K. M. Marshek, G. Gerbert, 1991a, A Framework for a Design Library for Mechanical Transmissions, *Proc. of 1991 NSF Design and Manufacturing Systems Conf.*, Austin, TX, January 9–11, pp. 1079–1088.

Kannapan, S., K. M. Marshek, 1991b, Design Synthetic Reasoning, Parts I, III and III, *Mechanism and Machine Theory*, Vol. 26, No. 7, pp. 711–739.

Kannapan, S., K. M. Marshek, 1991c, Evaluating the Patentability of Engineered Devices, *Proc. Artificial Intelligence in Design*, Edinburgh, Scotland, Butterworth-Heinemann, pp. 683–701.

Kannapan, S., K. M. Marshek, 1992a, Engineering Design Methodologies: A New Perspective, In *Intelligent Design and Manufacturing*, ed. A. N. Kusiak, John Wiley and Sons, pp. 3–38.

Kannapan, S., K. M. Marshek, 1992b, A Schema for Negotiation between Intelligent Design Agents in Concurrent Engineering, In *Intelligent Computer Aided Design*, eds. D. C. Brown, M. B. Waldron and H. Yoshikawa, IFIP Transactions B, Elsevier Science, Amsterdam, pp. 1–25.

Karandikar, H., R. Srinivasan, F. Mistree, W. J. Fuchs, 1989, Compromise: An Effective Approach for the Design of Pressure Vessels using Composite Materials, *Computers and Structures*, Vol. 33, No. 6, pp. 1465–1477.

Klein, M., 1992, Detecting and Resolving Conflicts among Cooperating Human and Machine-based Design Agents, *Artificial Intelligence in Engineering*, Vol. 7, pp. 93–104.

Kota, S., 1990a, Qualitative Motion Synthesis: Towards Automating Mechanical Systems Configuration, *Proc. of the 1990 NSF Design and Manufacturing Systems Conf.*, Arizona State University, Tempe, AZ, pp. 77–91.

Kota, S., C-L. Lee, 1990b, A Computational Model for Conceptual Design: Configuration of Hydraulic Systems, *Proc. of NSF Design and Manufacturing Systems Conf.*, Arizona State University, Tempe, AZ, Jan. 8–12, pp. 93–104.

Kuppuraju, N., P. Ittimakin, F. Mistree, 1985, Design through Selection: A Method that Works, *Design Studies*, Vol. 6, No. 2, pp. 91–105.

Lander, S. E., 1989, Knowledge-based Systems for Cooperating Experts, *Computer and Information Sciences Technical Report 91-28*, University of Massachusetts, Amherst.

Lyons, J. L., 1982, *Lyons' Valve Designer's Handbook*, Van Nostrand Reinhold Publishing Co.

MacDonald, J. G. F., 1973, Power Operable Pivot Joint, *United States Patent 3, 731, 546*, May 8.

Maher, M., S. Fenves, 1985, HI-RISE: An Expert System for the Preliminary Structural Design of High Rise Buildings, *Knowledge Engineering in Computer Aided Design*, (Editor) J. S. Gero, North-Holland, Amsterdam, pp. 125–135.

McDermott, D., 1978, Circuit Design as Problem Solving, *Proc. IFIP Workshop on AI and Pattern Recognition in CAD*, ed. J-C. Latombe, North Holland, Amsterdam, pp. 227–259.

McDermott, J., 1982, Rl: A Rule-Based Configurer of Computer Systems, *Artificial Intelligence*, Vol. 19, pp. 39–88.

Mitchell, T. M., S. Mahadevan, L. I. Steinberg, 1985, LEAP: A Learning Apprentice for VLSI Design, *Proc. IJCAI*, Los Angeles, California, pp. 573–580.

Mostow, J., M. Barley, 1987, Automated Re-Use of Design Plans, *Proc. Intl. Conf. on Engineering Design*, Boston, Aug. 17–20, ASME, Vol. 2, pp. 632–647.

Mullins, S., J. R. Rinderle, 1991, Grammatical Approaches to Engineering Design, Part 1: An Introduction and Commentary, *Research in Engineering Design*, Vol. 2, No. 3, pp. 121–135.

Murthy, S. S., S. Addanki, 1987, PROMPT An Innovative Design Tool, *AAAI-87*, pp. 637–642.

Navinchandra, D., K. P. Sycara, S. Narasimhan, 1991, A Transformational Approach to Case-based Synthesis, *Artificial Intelligence in Engineering Design and Manufacturing*, Vol. 5, No. 1, pp. 31–35.

Prabhu, D. R., D. L. Taylor, 1989, Synthesis of Systems from Specifications Containing Orientations and Positions Associated with Flow Variables, *1989 ASME Design Automation Conf.*, Montreal, Canada, September 17–21, pp. 273–280.

Rinderle, J. R., 1991, Grammatical Approaches to Engineering Design, Parts II: Melding Configuration and Parametric Design by Attribute Grammars, *Research in Engineering Design*, Vol. 2, No. 3, pp. 137–146.

Sim, S. K., Y. W. Chan, 1991, A Knowledge-based Expert System for Rolling Element Bearing Selection in Mechanical Engineering Design, *Artificial Intelligence in Engineering*, Vol. 6, No. 3, pp. 125–135.

Sobieszczanski-Sobieski, J., 1988, Optimization by Decomposition: A Step from Hierarchic to Non-hierarchic Systems, *Technical Report TM-101494*, September, NASA Langley Research Center, Hampton, VA.

Sriram, D., R. Logcher, A. Wong, S. Ahmed, 1991, An Object-Oriented Framework for Collaborative Engineering Design, In *Computer-Aided Cooperative Product Development*, eds. D. Sriram, R. Logcher, S. Fukuda, Springer-Verlag, pp. 51–92.

Stefik, M., 1981, Planning with Constraints (MOLGEN: Part 1), *Artificial Intelligence*, Vol. 16, pp. 111–140.

Sussman, G. J., G. L. Steele, 1980, CONSTRAINTS—A Language for Expressing Almost-Hierarchical Descriptions, *Artificial Intelligence*, Vol. 14, pp. 1–39.

Sycara, K. P., 1990, Negotiation Planning: An AI Approach, *European Journal of Operations Research*, Vol. 46, pp. 216–234.

Takeda, H., T. Tomiyama, H. Yoshikawa, 1992, A Logical and Computable Framework for Reasoning in Design, *Proc. ASME Design Theory and Methodology Conf.*, Scottsdale, AZ, pp. 167–174.

Topping, B. H. V., 1983, Shape Optimization of Skeletal Structures: A Review, *Journal of Structural Engineering*, Vol. 109, No. 8, pp. 1933–1951.

Ulrich, K. T., 1988, *Computation and Pre-Parametric Design*, PhD Dissertation, Massachusetts Institute of Technology, Cambridge, MA.

Ward, A. C., 1989, *A Theory of Quantitative Inference for Artifact Sets, Applied to a Mechanical Design Compiler*, PhD Dissertation, Massachusetts Institute of Technology, Cambridge, MA.

Williams, B., 1989, *Invention from First Principles via Topologies of Interaction, PhD Thesis*, Massachusetts Institute of Technology, Cambridge, MA.

Wood, K. L., E. K. Antonsson, 1990, Modeling Imprecision and Uncertainty in Preliminary Engineering Design, *Mechanism and Machine Theory*, Vol. 25, No. 3, pp. 305–324.

11
A Data Representation for Collaborative Mechanical Design

Richard L. Nagy, David G. Ullman, and Thomas G. Dietterich

Abstract. Collaborative design projects place additional burdens on design documentation practices. The literature on group design has repeatedly documented the existence of problems in design decision making due to the unavailability of design information. This paper describes a data representation developed for collaborative mechanical design information. The data representation is used to record the history of the design as a sequence of design decisions. The resulting database records the final specifications, the alternatives that were considered during the design process, and the designers' rationale for choosing the final design parameters. It is currently implemented in a computerized data base system under development by the Design Process Research Group (DPRG), at the authors' institution (Oregon State University).

1. Introduction

The data representation described in this chapter was developed to record design information from collaborative mechanical design projects. This data representation is implemented in computerized design history tool (DHT). It is the authors' opinion that mechanical design practice is moving toward collaborative design efforts involving interdisciplinary design teams. A consequence of this shift toward collaborative design is an increase in the problems associated with managing the information generated in the design process. Problems in managing design information for collaborative design projects have been identified in other design process research efforts. Several examples from the literature are

The three most salient problems found across different design projects, were:
1. the thin spread of application domain knowledge
2. fluctuating and conflicting design requirements.
3. communication and coordination breakdowns (Curtis 1988)

The rationale and context for key design decisions and assumptions becomes lost and confused, both over time and between development groups (Curtis 1988).

> Critical errors are commonly made in the formulation and resolution of design decisions (Yakemovic 1989).

> A substantial portion of design related information from group meetings is not easily shared with other members to aid in decision making or problem solving (Morjaria 1989).

The above quotations are not all from research directly related to mechanical engineering design. However, it seems reasonable to assume that collaborative design projects in general will share the type of problems identified in the above quotations. It is the opinion of the authors that, with the exception of the problem concerning "the thin spread of application domain knowledge," these problems demonstrate an ever-increasing need for information management tools to aid designers in collaborative design projects. Information management tools can aid designers by organizing design documentation, including new types of design information not traditionally recorded, and integrating all recorded information in a shared database.

The traditional methods of recording mechanical design information are in design drawings, plans, and specification sheets. These recording methods do not represent the decision process, and only the end result of design decisions are recorded. Other traditional forms of recorded design information, such as design notebooks, logbooks, meeting notes, and revision drawings, often exist, but this information is generally not available to all the design participants, nor is it in a form easily shared between design groups. The alternative proposals considered during the decision process and the reasons for rejecting the alternatives and accepting a particular proposal are either not recorded or that information is difficult to access.

For recorded design information to be useful during the design process, it must be readily accessible to the designers. It is the opinion of the authors that a computer database is currently the most suitable tool to organize design process information. The traditional methods of recording design information are too poorly integrated to serve effectively as a shared database for collaborative design. This chapter describes a data representation developed for collaborative mechanical design information. The data representation is used to record the history of the design as a sequence of design decisions. The resulting database records the final specifications, including a three-dimensional rendering of the design. The database contains information on the alternatives that were considered for each design parameter during the design process. This includes the arguments for and against the alternate proposals and the designers' rationale for choosing the final design parameters and rejecting the alternatives.

Section 2 of this chapter summarizes research related to this topic; Section 3 describes the data representation; Section 4 briefly describes the implementation of the data representation in a computer database; and Section 5 states some conclusions drawn from current implementation.

2. Background

2.1. *Related Internal Research*

The context of the research described in this chapter is the ongoing effort at the authors' institution to understand and develop tools to support the mechanical design process. As part of this effort a model of the mechanical design process was developed based on studies of single engineers performing novel (nonroutine) mechanical designs of moderately complex components (Ullman et al., 1988). This model is referred to as the task/episode accumulation model.

A conclusion, drawn from the model and analysis of the protocol data, is that the temporal history of a mechanical design process can be mapped by recording the individual episodes as a sequence of decisions. A computerized DHT was developed based on this model, which is capable of recording design process information from individual design projects (Chen, 1990).

2.2. *Related External Research*

The collaborative design history data representation developed in this research, as well as several other recent efforts to develop design history tools, is based on the IBIS (Issue-Based Information System) method. IBIS was developed by Horst Rittel for organizing the deliberation process that occurs during complex decision making (Rittel et al., 1973). The IBIS method organizes the deliberation process into a network of three data elements: *issues*, *positions*, and *arguments*. An *issue* is an identified problem to be resolved by deliberation. Each issue can have many *positions* that are proposed solutions developed to resolve the issue. Each position can have any number of *arguments* that support or oppose that position. Using the IBIS model, a deliberation is started when someone creates an initial data element and others respond with additional data elements based on one of the defined legal inter-element relationships. Figure 11.1 is a network diagram of the IBIS method with data elements depicted as network nodes and relationships depicted as arrows. IBIS is a general model of the deliberation process. It does not directly provide a way to indicate a successful issue resolution or, which position was finally accepted by the participants in the deliberation process. Nor does it incorporate a method of representing the temporal sequence of the deliberation process.

One computer application based on the IBIS model is gIBIS (graphical IBIS)[1] (Conklin and Begeman, 1988). The gIBIS tool was developed as a computer aided design tool to capture design histories and support

[1] gIBIS was developed at Microelectronic and Computer Technology Corp.

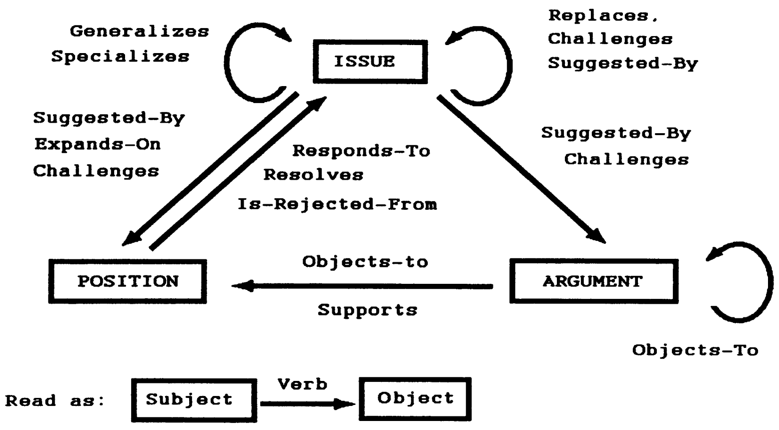


FIGURE 11.1. IBIS network.

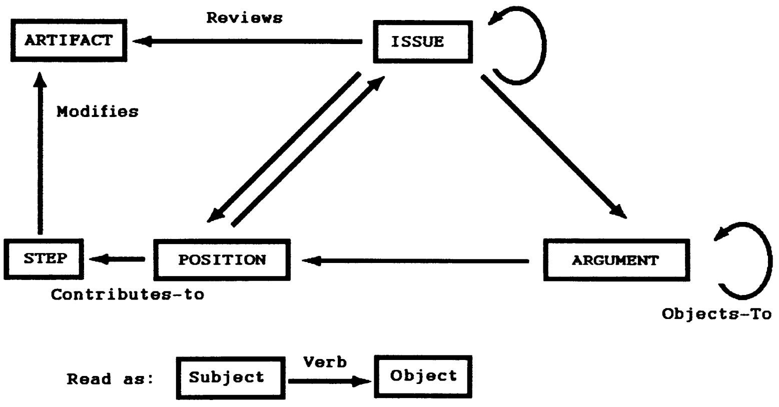


FIGURE 11.2. gIBIS network.

computer-mediated teamwork (groupware). Groupware refers to "*computer-based systems that support two or more users engaged in a common task, and that provide an interface to a shared environment*" (Ellis et al., 1988). The gIBIS system uses an extension of the IBIS model, developed by Collin Potts, that includes *artifacts* and *steps* (see Figure 11.2). *Artifacts* represent whatever documents and standard notations are used to represent the *steps* and *steps* represent the changes that are made to *artifacts* to revise them toward correctness or completeness. The gIBIS system employs a commercial relational database system (that supports report generation) as the output interface, and it uses a unique hypertext input interface. The hypertext system supports multiple users, via a computer network. In the gIBIS system the latest agreed-upon position to resolve a particular issue is marked. By recording the current accepted position on each issue of a design process, the gIBIS system can identify the current state of the design, but it does not record the temporal history of the design process.

Another computer-aided design history tool based on the IBIS method is the MIKROPLIS hypertext system. MIKROPLIS is a computer program for managing textual design information, representing designers' reasoning (McCall, 1989). It is based on PHI (procedural hierarchy of issues), developed by Raymond McCall, and it is an extension of Rittel's IBIS system. PHI extends IBIS's network structure by allowing each primary structure to be decomposed as subissues, subpositions, and subarguments to any level of granularity. PHI imposes a quasihierarchial structure on the recorded information, as opposed to the IBIS multilinked network. Whereas IBIS supports several inter-issue relationships, PHI connects issues only by a *serve* relationship. Issue-2 serves Issue-1 if answering (resolving) Issue-2 is useful for answering Issue-1. This simpler structure alleviates some problems in retrieving information from large, complex databases. Like gIBIS, MIKROPLIS records only the current state of the design process and does not record the temporal history of the design process.

2.3. Data Source

The design process data representation was developed by the author based on both a review of related research, and by examining videotaped protocol data of engineers doing mechanical design. The protocol data used in this research is from three sources. One source was the protocol data recorded earlier by researchers at the authors' institution. This data is of five individual professional designers performing mechanical design, and includes all phases of the design process: conceptual, layout, and detail design.

The other two sources of videotaped protocol data are from outside researchers, which record students performing collaborative design.[2] One set of data recorded at Stanford University, involved group design sessions recorded to understand collaborative workspace activity (Tang and Leifer, 1988). The video data from the University of Queensland was recorded to support developing statistical methods to quantify the quality and effectiveness of mechanical designs (Radcliffe and Lee, 1989). These three sources of protocol data were studied to determine the kinds of information, generated in mechanical design processes that the design history representation would need to embody. As the representation evolved it was continually tested using design episodes from this protocol data. The tests were conducted by entering a representation of design episodes into a computer database.

[2] The author is indebted to Dr. John C. Tang, currently at Sun Microsystems, Inc., and Drs. Tat Y. Lee, and David F. Radcliffe from the University of Queensland, Australia, for the loan of video tapes of collaborative, mechanical design sessions.

3. Design Process Data Representation

3.1. Data Elements

The objective of this research was to develop a data representation for collaborative mechanical design. The data representation is used to represent design information in the DHT computer database. The representation developed in this research is composed of four data-elements: *issues*, *proposals*, *arguments*, and *decisions* (see Figure 11.3). A design process history is recorded in the database by representing the actual design process with these four data-elements. The representation is based on an idealized four-step model of mechanical design. In the first step designers identify a design *issue* or issues. During the second step *proposals* are developed to resolve particular design issues. In the third step the designers formulate *arguments* either supporting or opposing specific proposals, and in the fourth step a design *decision* is made to accept or reject the proposals.

The data representation is developed to record the results of the creative design activity. It does not represent the creative process itself, which is internal to the designer's mind, and not articulated during the design process. Although developed to record collaborative design processes, the data representation is also suitable to record the design history of individual designers. In this report the use of the words *group* or *designers* should, in general, be interpreted as meaning one or more participants in a particular design process.

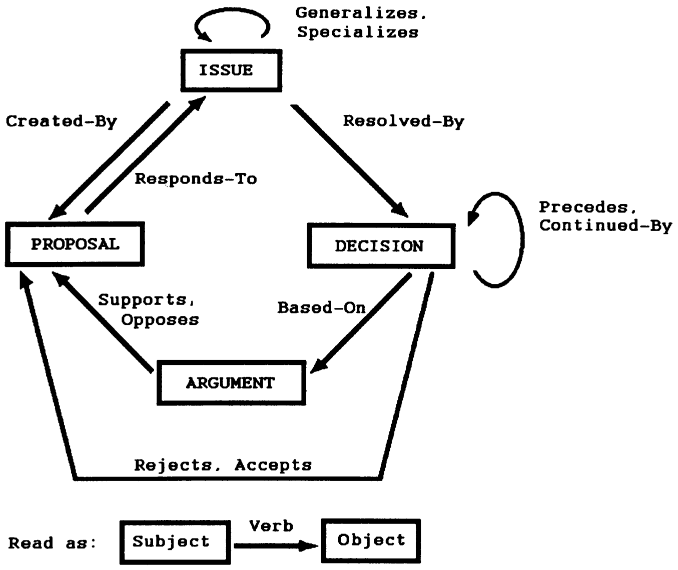


FIGURE 11.3. Data representation network.

The four principle data-elements are linked to form compound networks that serve to organize the design history data in the computerized database. The design process data-elements and networks used in this representation are described below.

A design *issue* is any question or problem identified by a designer or design team, that will need to be resolved to complete the design process. An issue may be the desire to satisfy a design requirement, the need to establish the value for some design object parameter, or any other design related question or requirement identified by a design participant.

A *proposal* is a suggested addition or change to the current design, developed by a designer or design team to resolve a particular design issue. Any number of proposals may be developed to resolve the same design issue.

An *argument* is the designers' rationale for either supporting or opposing particular proposals. Arguments identify the relative merits or demerits of proposals. A design argument is a comparison, which can be either absolute or relative (Ullman, 1991). In an absolute comparison, there is only one proposal being focused on, and it is directly compared to a set of requirements defined by the given design specifications and the results of previously accepted proposals. A decision may evaluate only one proposal based exclusively on absolute type arguments. In a relative comparison, there is a set of proposals being focused on, and the ability of each proposal to satisfy the set of requirements is compared relative to the other proposals.

A design *decision* is a continuous segment of the design process in which the participants evaluate a proposal or proposals to resolve a particular design issue by weighing the arguments supporting or opposing the proposals. The result of a decision (if it is concluded) is either to accept or reject the proposals. The evaluation may be based on an informal consensus or on a formal method such as Pugh's method of concept selection. A decision also may be suspended without accepting or rejecting any proposal, as when the designers that feel additional information is required or that no satisfactory proposal is currently available. In this sense a proposal can exist in the database in one of three states: *accepted*, *rejected*, or *proposed*. When a proposal is accepted, it effects the current state of the design. A rejected proposal has been determined to be unacceptable for the current design (it may have been previously accepted), and does not effect the current state of the design. A proposal that is simply proposed was part of a suspended decision process and also does not effect the current state of the design.

3.2. Example 1

The following is a hypothetical example of design history data of a cantilever beam design problem (see Figure 11.4) organized using the four data-elements of this representation. For this problem Table 11.1 lists the given specifications. This simple example is only meant to clarify how the data-elements are

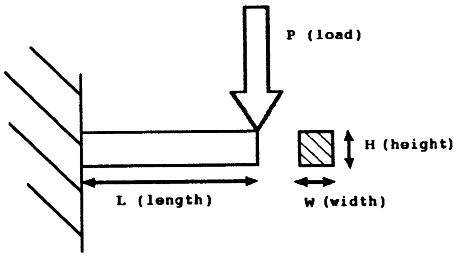


FIGURE 11.4. Cantilever beam.

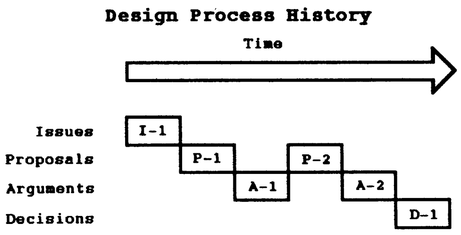


FIGURE 11.5. Design process diagram.

TABLE 11.1. Given specifications.

1.	Beam is made out of aluminum.
2.	Allowable stress is 2000 psi.
3.	Beam length is 12 in.
4.	End load is 10 lb.
5.	Beam cross-section is rectangular.
6.	Minimize production cost.
7.	Minimize beam weight.

used in the representation. The statements recorded in the outline are paraphrased from what the designers articulated during the design process.

Figure 11.5 is a diagram of the data-elements that represent this particular segment of the hypothetical collaborative design process. Each of the data-elements of Figure 11.5 is summarized in Outline-1. In this design process segment the design issue (I-1) was the need to establish the cross-section dimensions for a cantilever beam. Two alternative proposals were developed to resolve this issue by different team members (P-1 and P-2). Argument A-1 represents the designer's reasons for supporting proposal (P-1). Argument (A-2) represents a second designer's rationale for supporting proposal (P-2) and opposing (P-1). Decision (D-1) represents the evaluation process of weighing the arguments and accepting a particular proposal. In this case, the evaluation process was a unanimous group consensus that favored accepting proposal (P-2) and rejecting (P-1). Outline-1 is not a complete example of all the information recorded in the current implementation. Other information such as a time and date stamp and additional inter-object links are not show in this example. The reader is referred to (Nagy, 1990) for a detailed description of the current implementation of the data representation in the DHT.

Outline-1

Issue-1: (*I-1*)
Description: We need to establish the cross-section dimensions for the cantilever beam.
Decisions: Decision-1
Source: Designer-1

Proposal-1: (*P-1*)
Issue: Issue-1
Description: Let the beam width = 0.50 inches, and the beam height = 1.00 inches.
Source: Designer-1

Argument-1: (*A-1*)
Supported Proposals: Proposal-1
Opposed Proposals: none
Rationale: Using the proposed cross-section dimensions, the calculated maximum stress for this cross-section, based on $d_{\max} = Mc/I$, is 1440 psi. This is well below the maximum allowable stress. In addition, as 1/2″ × 1″ is a standard stock size, it will keep production cost low.
Source: Designer-1

Proposal-2: (*P-2*)
Issue: Issue-1
Description: Let the beam width = 0.45 in., and the beam height = 0.90 in.
Source: Designer-2

Argument-2: (*A-2*)
Supported Proposals: Proposal-2
Opposed Proposals: Proposal-1
Rationale: If we keep the height to width ratio 2 : 1, we can reduce beam weight by letting maximum stress equal allowable stress, and then solve for a minimum cross-sectional area. The proposed values are rounded to the nearest 1/20 in. I believe the resulting 19% reduction in weight is more important than the possible additional production cost.
Source: Designer-2

Decision-1: (*D-1*)
Issue: Issue-1
Arguments: Argument-1, Argument-2
Accepted Proposals: Proposal-2
Rejected Proposals: Proposal-1
Evaluation: unanimous consensus
Source: Designer-1, Designer-2, Designer-3

The above outline of a naive collaborative design secession is meant to demonstrate how design process information is recorded using the four data-elements. In order to organize the data within the database the four data-elements are linked to form higher-level data-structures. Four such linked networks are implemented: the *decision-chain*, which is used to record the chronological history of the design process; the *decision-process*, which is used to record the evolution of one issue's resolution; and the *issue-decomposition* and *issue-network*, which are used to represent the interconnectivity of the design-issues. These four network structures are described below.

3.3. Decision-Chain

The method used in this representation to record the temporal history of a design process is to form a chronological sequence of decisions, called a *decision-chain* (see Figure 11.6). As defined above, an individual decision is a continuous segment of the design process where proposals on a particular issue are evaluated. However, individual segments of a design process on a particular issue will not always conclude with accepting or rejecting a proposal. A decision may be suspended without acting on any proposal, or designers may move from one issue to another without evaluating or even identifying any proposals. This is likely to occur during the conceptual design phase where the main focus may be in identifying the design issues. Because the temporal history of the design process is recorded as a sequence of decisions, a decision data-element is used to record each continuous segment of the design process where a different issue is considered. A decision data-element is created even if no proposals or arguments are developed, or an argument evaluation is not carried out. For example, consider the case

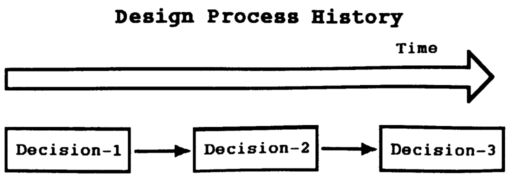


FIGURE 11.6. Decision-chain.

where a designer identifies several design issues, such as the need for a particular part to be both thermally conductive and electrically nonconductive, without proposing any solutions. Two issues have been identified but no additional design work on those issues takes place at that time. This design process segment would be recorded as two issue data-elements each encapsulated in a separate decision data-element. In this case these decision data-elements only serve to record the chronological sequence of the design process and are not "decisions" in the normal sense of the word. The evolution of any particular issue's resolution during the entire design project is recorded as a network of decision data-elements described below.

3.4. Decision-Process

The process of resolving a particular issue may be addressed any number of times during a design project. In collaborative design projects, individual designers or separate design teams may even make independent decisions on the same design issue. At a later point in the design process, the separate groups may get together to resolve any conflicts in the previously accepted proposals. This merging and re-evaluation of previous decisions is modeled by linking the decisions concerning one issue into a *decision-process* network. A decision-process can be considered as a meta-decision on a particular issue, which evolves during the course of the design process. Each re-evaluation process is represented as a new decision with pointers from any previous decision that the designers were aware of when making the re-evaluation. The resulting network of linked decisions forms a tree that grows from the root (see Figure 11.7). At the end of the design process all decisions that focus on a particular design issue would normally be merged at design group meetings into a single composite design-process with one final root decision.

Figure 11.7 is interpreted as follows. The six decisions (Dec-1, Dec-2, Dec-13, Dec-35, Dec-62, Dec-87) represent all those segments of the design project where Issue-1 was considered and, hopefully, finally resolved. When decision Dec-2 took place the participants were aware of the proposals, the arguments, and the evaluation associated with decision Dec-1. This could have been the same group or a different group who had access to a design

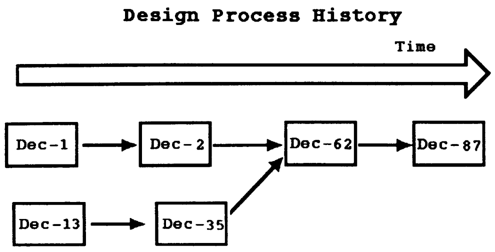


FIGURE 11.7. Decision-process.

history that included decision Dec-1. Decision Dec-13 took place shortly after Dec-1 and without knowledge of what transpired in decision Dec-1. Decision Dec-35 was a continuation of Dec-13, again without knowledge of decisions Dec-1 or Dec-2. The participants in decision Dec-62 were aware of all the previous decisions as were the participants of the final decision, Dec-87.

3.5. *Issue-Decomposition*

A third data-element network implemented in the representation is the *issue-decomposition*. The *issue-decomposition* represents the breakdown of an issue into subissues. It is used in the database to organize the total set of identified issues into hierarchies based on the concept of connected issue resolution. It is therefore used to define whether a particular design issue is currently resolved. The relation between a parent issue and its child subissues is based on an issue's resolution. A parent issue is not defined as "resolved" unless all of its immediate subissues are resolved. As subissues also may be decomposed, the *issue-network* can have any number of levels (see Figure 11.8).

Child issues are connected by "and" links to form a decomposition of the parent issue analogous to a table decomposed into a top and four legs. Just as the table is complete only if each of its parts is complete, a parent issue is resolved only when each subissue, connected by an "and" link, is resolved. Any proposal developed directly to resolve a parent issue must resolve all the immediate "and" linked child subissues.

3.6. *Issue-Network*

The fourth data-element network, called an *issue-network*, is composed of issues and proposals (see Figure 11.9). As described above, an issue is not resolved unless its "and" linked subissues are also resolved. An additional requirement for issue resolution involves the "or" linked child issues. Subissues are connected by "or" links to a parent issue through the proposals developed to resolve that parent issue. Any proposal may introduce new

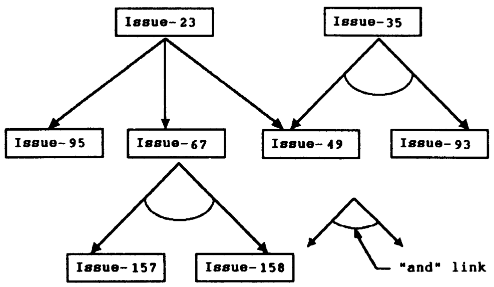


FIGURE 11.8. Issue-decomposition.

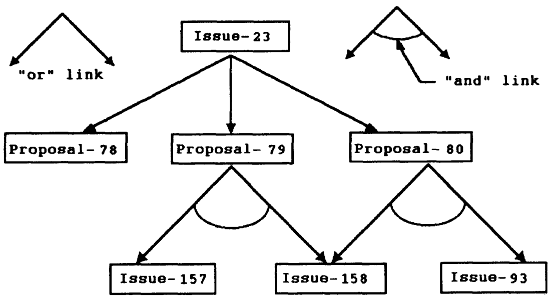


FIGURE 11.9. Issue-network.

"created-issues" that only affect the design if that proposal is accepted. These "stepchild" issues, connected by "or" links, do not decompose the parent issue into parts, so they do not form part of that issue's *issue-decomposition*. Nevertheless a parent issue is not defined as "resolved" unless any subissues, inherited from an "accepted" proposal, are also resolved. For example, a proposal to use steel for the material of a design object might *create* the issue of a need to protect that part from direct exposure to moisture. If the proposal to use steel is accepted, then the *issue* of needing to choose a type of material for the design object would not be resolved until the *created issue* of needing to prevent corrosion also was resolved.

4. Implementation

The purpose of this research is to augment the current DHT so that it can be used to record the history of collaborative design processes. Two of the data-elements used in design process representation presented here were previously developed by other researchers at the authors' institution: *design-objects* and *constraints* (McGinnis, 1990). *Design-objects* are the graphical and semantic representations of the physical artifacts developed in the design process. A design *constraint* is the fundamental data-structure of the representation. Design constraints define all the values and features of the design-objects, and all relationships between design-objects. In the DHT, a constraint represents the finest grain size of information about the design. Figure 11.10 depicts the relationships (arrows) between the six data-elements (boxes) implemented in the design process representation.

The design process representation is implemented in the LISP programming language using HyperClass,[3] an object orientated extension to Com-

[3] HyperClass was developed at Schlumberger Technologies Inc. for building, maintaining, and using database systems.

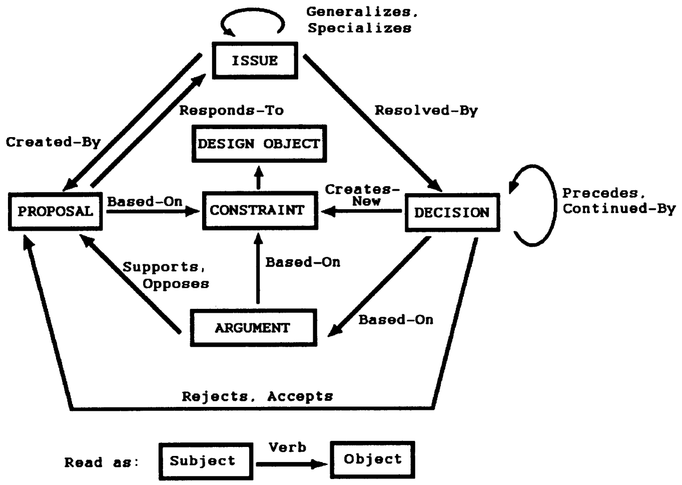


FIGURE 11.10. Data-element network.

mon Lisp. A HyperClass *object* (which should not be confused with a "design-object"), is a type of data record that encapsulates, in a single entity, the object's data with procedures that operate on that data. These encapsulated procedures are called *methods*. The data and methods of an object are stored in *slots* of the object. A complete explanation of HyperClass can be found in (Smith et al., 1988).

Each of the six data-elements depicted in Figure 11.10 is implemented in the DHT as a HyperClass object: *design-object*, *constraint*, *proposal*, *argument*, *decision*, and *issue*. As the *design-object* and *constraint* were previously developed, the reader is referred to McGinnis (1990) and Chen et al. (1990) for complete descriptions of these two objects. The four principal objects developed in this research are the *proposal*, *argument*, *decision*, and *issue* objects. Two additional supplementary objects, the *source*, and *date*, are encapsulated in each instance of the four principal objects to identify the designer(s) originally responsible for developing that particular data-element, and the day and time the development occurred.

The design process representation uses four data-element networks: *decision-chain*, *decision-process*, *issue-decomposition*, and *issue-network* (see Figures 11.6–11.9) to represent a design process history. These data networks are formed by linking the six principal data-objects as shown in Figure 11.10. The data-element networks are implemented in order to organize the design history data stored in the database for later retrieval by users of the database. The links between the six principal data-objects are implemented by the slots in each object.

5. Conclusions

The design history data representation described in this chapter was developed to record information generated in collaborative mechanical design processes. By encapsulating proposals, arguments, and an evaluation within a decision data-element the representation can record the designers' rationale for supporting or opposing alternative proposals, as well as the decision's evaluation process and the effect (if any) of a decision on the design. A history of the design process can be played back by retracing the chronologically ordered *decision-chain.* In addition, the representation includes the idea of a design issue as the tie that binds alternative proposals to a particular decision, and individual decisions into a *decision-process* that maps the history of an issue's resolution. The design issue data-element is also used to record the process of *issue-decomposition* common to the conceptual design phase of the mechanical design process.

The data representation developed in this research borrows from the IBIS deliberation method described in Section 2. This can readily be seen by comparing Figures 11.1 and 11.3. The *position* element in the IBIS system has been renamed *proposal* in this representation to better describe the process that it is used to represent in mechanical design. The data representation described in this chapter adds three principal data-objects to the three used in the IBIS method; *design-object*, *constraint*, and *decision* (see Figure 11.10). Design-objects are the physical artifacts conceptually developed in the design process. The design-objects are defined by the constraints that identify all the values and features of design objects. This relationship is depicted in Figure 11.10 by the *modifies* link between the CONSTRAINT and the DESIGN-OBJECT blocks. The design *decision* is the third data-element not found in the IBIS system. The decision data-element serves three distinct purposes in the DHT that are not represented in the IBIS system. The first is to identify which of all the proposals developed to resolve a particular issue are currently accepted, and which are rejected. The second is to represent the temporal history of the design process. As described in Section 3, the *decision-chain* network is used to represent the temporal history of the design process as a chronological sequence of decisions. The third is to record the chronological history of an issue's resolution as a network of decisions concerning that issue in a *decision-process.*

By going beyond the IBIS method and including three additional principal data-elements, the DHT design process data representation is capable of representing the kinds of design information observed in the design protocol data reviewed for this research. The resulting database is an extension of the kinds of design knowledge currently recorded in design drawings, sketches, designer's notebooks, and meeting notes. It extends current design records by both recording a chronological history the decision process, including the alternatives considered, and by centrally organizing the recorded information in a computer database. By organizing the design data into the four data-element networks described in Section 3, the temporal design history, issue resolution, and issue interaction is represented. By structuring the recorded information in the design history and centrally locating that information in a computer database, the data representation should allow the information to be easily shared among the design participants.

In the introduction of this chapter, six problems associated with collaborative design efforts are cited from current design literature. It is the authors' hypothesis that a design history tool (DHT) employing the data representation described here could help resolve the later five of these six issues. Managing fluctuating design requirements and preventing conflicts should be assisted by design participants readily having access to the most current requirements (recorded as issue data-elements). An additional issue managing aid is provided by maintaining an up-to-date history of an issue's resolution (organized as a decision-process). Communication and coordination between the various design groups as well as among group members should be facilitated by sharing ideas (issues, proposals, arguments) through a centralized database. By recording the design decision process, the data representation directly addresses the problem of preventing the loss of the context of design decisions, and coordinating the decision process between development groups. Having the arguments upon which a decision was based readily accessible to be scrutinized by all members of a design process should help expose critical errors before the final decision evaluation. And, finally, having a centralized design history database should considerably facilitate the sharing of design information from group meetings.

The data representation as implemented in the DHT has currently been tested by the author using videotaped protocol data. These feasibility tests were used simply to verify that the system could be used to record and retrieve the information generated in collaborative mechanical design processes. Research to verify the effectiveness of such a system is proceeding at the authors' institution under the directorship of Dr. Ullman.

References

Chen, A., McGinnis, B., Ullman, D. G., and Dietterich, T. G. (1990). Design history knowledge representation and its basic implementation. Department of Mechanical Engineering, Oregon State University.

Conklin, J., and Begeman, M. (1988). gIBIS: A hypertext tool for exploratory policy discussion. *Proceedings of the Conference on Computer Supported Cooperative Work*, September 1988, ACM, pp. 140–152.

Curtis, B., Krasner, H., and Iscoe, N. (1988). A field study of the software design process for large systems. *MCC Technical Report*, Number STP-233-88.

Ellis, C. A., Gibbs, S. J., and Rein, G. L. (1988). The groupware project: An overview. *MCC Technical Report*, Number STP-033-88.

McCall, R. J. (1989). MIKROPLIS: A hypertext system for design. *Design Studies*, *10*(4), 228–238.

McGinnis, B. D. (1990). An object orientated representation for mechanical design based on constraints. Master of Science Thesis, Department of Mechanical Engineering, Oregon State University.

Morjaria, M., Kelsch, R., and Draugelis, V. (1989). Computer-supported tools for collaborative engineering design. Design Institute, Xerox Corp.

Nagy, R. L. (1990). A knowledge base data representation for collaborative mechanical design. Master of Science Report, Department of Mechanical Engineering, Oregon State University.

Radcliffe, D., and Lee, T. Y. (1989). Design methods used by undergraduate engineering students. *Design Studies*, *10*(4), 199–207.

Rittel, H. W. J., and Webber, M. M. (1973). Dilemmas in a general theory of planning. *Policy Sciences*, *4* 155–169.

Smith R. G., Kleyn, M. F., and Schoen, E. (1988). Impulse reference guide. Research Note SYS-47-41, Schlumberger Technologies Corp.

Tang, J. C., and Leifer, L. J. (1988). A framework for understanding the workspace activity of design teams. System Sciences Laboratory Xerox Palo Alto Research Center, Center for Design Research Mechanical Engineering Dept., Stanford University.

Ullman, D. G. (1991). *Mechanical Design Process*. New York: McGraw Hill.

Ullman, D. G., Dietterich, T. G., and Stauffer, L. A. (1988). A model of the mechanical design process based on empirical data. *Artificial Intelligence in Engineering, Design, and Manufacturing*. *2*(1), 33–52.

Yakemovic, K. B., and Conklin, J. (1989). The capture of design rationale on an industrial development project: preliminary report. *MCC Technical Report*, Number STP-279-89, July 14, 1989.

12 Characterizing Human Analogical Reasoning*

BETH ADELSON

1. Introduction: Motivation and Approach

Skilled problem-solvers often work by analogy as opposed to solving from scratch every new problem they encounter. And this is very much the case in engineering design. For example, a cantilevered beam provides an anology for a cantilevered bridge; as does a suspension bridge for a suspension building. With regard to invention, according to Samuel Morse's diaries,[24] initially, in trying to transmit telegraphic signals across significant distances Morse tried the strategy of building successively stronger generators. He found however, that the signals still degraded with distance. Supposedly the solution to the problem came to him in the following way. While riding on a train, he happened to look out of the window and notice a Pony Express depot, at which horses were being fed and watered. Morse realized that the *relay station* strategy constituted an analogical solution to the telegraph problem as well.[†] In a similar vein Edison's diaries recount that he invented the kinetiscope by setting out to "do for the eye what he had done for the ear" with the phonograph.[4]

The work described here is part of a research program to develop a computational theory that makes use of the central characteristics of human analogical reasoning. We begin by studying the phenomenon in context. Because the purpose of analogical reasoning is to learn and solve problems, we have developed our theory by consistently observing subjects within a problem-solving context. This approach has yielded insights into the nature of the phenomonon being modeled and, as a result, has provided constraints allowing us to specify the theory in a number of ways that increase its power.

*Thanks to Dedre Gentner, Brian Falkenhainer, and Ken Forbus for their loan of SME. This work was funded by grants from NSF's CISE and Engineering Directorates and a Henry Rutgers Research Fellowship.
[†]Although accounts from diaries may not be entirely accurate, as here, they often provide excellent examples of familiar a phenomenon.

As we will discuss in Section 3.1, recent research suggests a class of theory that rests on a process consisting of retrieval, mapping, evaluation, debugging, and generalization.[8,10,11,15,19,28,20,22] Each of these components of the process is defined in detail in Section 3.1, but it is useful to give the reader an immediate feel for the process here.

In this process, a problem that is analogous to the current "target domain" problem, but that previously has been solved, is retrieved from memory. The solution to the old "base domain" problem is then "mapped" or imposed on the new problem, giving the problem-solver a way of looking at the new problem. The old solution is then evaluated and debugged. That is, the problem-solver considers the ways in which the old solution needs to be modified in order to lead to success with respect to the new problem. Once the new problem is solved the method may become a part of the problem-solver's general bag of tricks.

We can use the example from the case study discussed in Section 2 to illustrate the process. In the example, a student is being taught about computer stacks by analogy to cafeteria stacks. The student is reminded that cafeteria stacks exhibit the last-in-first-out property of the data structure being presented to him. This allows him to map or impose the cafeteria domain model onto what he knows about data structures. He then evaluates and debugs the cafeteria model, modifying it in a way that is appropriate for the computer domain.

The work we present here extends existing cognitive theories of analogical reasoning by specifying some possible mechanisms for mapping evaluation and debugging. In our theory these three processes are active and problem-driven. Additionally, they are heavily dependent on the use of knowledge about function, structure, and mechanism. From this point of view we discuss the following processes in detail in Section 3:

1. Purpose-Constrained Mapping: In teaching about a complex domain, tutors focus their students' attention on individual aspects of the domain, allowing the students to map models that are partial but sufficient for their immediate purpose.[8,9] As explained below, this strategy results in incremental learning that makes both mapping and debugging more tractable. In this way, purpose provides a powerful and useful constraining strategy that needs to be included in a specification of the mapping process.
2. Active Evaluation: A system that models human problem-solving needs to be able to identify the bugs inherent in an analogically acquired domain model. Our system actively searches for bugs in newly mapped domain models; the system, rather than a human tutor, initiates the search. The system does so by comparing the nature of the actions and objects in the newly mapped model to the nature of the actions and objects appropriate in the domain being mapped into (see Section 3.3.1 for algorithmic details). This allows the system to identify the aspects of

the model that are inappropriate and therefore unlikely to hold in the domain being mapped into. Our system also has knowledge about the way in which analogical correspondences are meant to be understood across domains. These aspects of our system reflect powerful elements of human reasoning.

3. Active Debugging: Once the buggy portion of a domain model has been identified, it must be replaced by a representation that is accurate in the new domain. Our system constructs and runs target and base domain simulations of the operations embodied in the models. This allows the system to identify mechanisms in the target domain that are *functionally* analogous to portions of the base domain models. The system can, as a result, correct mapped models, maintaining the functional explanation provided by the analogical example while building a representation of a mechanism appropriate to the target domain.

   Here, too, our system's behavior characterizes the constrained way in which analogical examples are understood and used; they are not taken literally, rather they are understood to be only partially applicable. (See the "spring and capacity limitation" example in the next section.) This understanding of the constrained applicability may be what allows problem-solvers to consider analogies to be helpful even though it is known both that they provide imperfect explanations and that the nature of the imperfection is unknown.

We are not alone in the *view* of analogical reasoning implied by the above set of issues. The spirit of Carbonell's[11] work on derivational analogy; Holyoak and Thagard's[19,28] work on multiple constraint satisfaction; Kedar-Cabelli's work on purpose-guided reasoning,[22] Burstein's[6] work on causal reasoning and Falkenhainer, Forbus, and Gentner's work[15] on structure mapping is consonant with our view of analogical reasoning as an active, knowledge-intensive process. Nevertheless, we make a contribution by adding these particular features, representative of human analogical reasoning, to each element in our theory.

2. Case Study: Protocol Data Illustrating the Issues

In developing our theory, we conducted a case study and collected protocol data from it.[1,14,23] Repeatedly we have drawn on the protocol data described below. These data yield insights into the mechanisms that underlie analogical reasoning.

The Protocol

In collecting our data we videotaped a tutor teaching a student about computer stacks as last-in-first-out data structures. The tutor's goal was to have the student be able to specify the procedures for "pushing" and "popping" items onto and off of stacks both in Pascal code and in box and arrow diagrammatic notation. We chose box and arrow notation, as well as code generation, because it is a frequently used exercise designed to have students understand the procedures involved in data structure manipulation. At the beginning of the protocol session, the student had just completed an introductory programming course in which he had learned about some basic programming constructs and about elementary data structures such as arrays and simple linked lists. He had not learned about using a linked list as a stack. The tutor had the intention of building upon the student's existing knowledge of Pascal through the use of analogy.

The relevant events of the protocol can be summarized as follows:

Learning About the Behavior of a Stack

The tutor told the student that in the field of computer science the data structures referred to as stacks are so named because their behavior is analogous to the behavior of the similarly named device that holds plates in a cafeteria. The student then proceeded to assimilate the information about the behavior of a stack. He thought of ways in which he might have previously encountered the use of stacks in programming. He suggested that stacks might be useful in implementing subroutine calls. He then also stated that, in general, when a task had an unmet precondition it would be useful to delay execution of the task by pushing it onto a stack.

Learning About the Mechanism Underlying the Behavior

The tutor told the student that the mechanism of the computer science stack is in some sense analogous to the mechanism of the cafeteria stack. In order to achieve "last-in-first-out" (LIFO) behavior, items are pushed *and* popped at the top of the cafeteria stack. The student drew a diagram of a cafeteria stack and described how *push* causes the stack's spring to compress and *pop* causes it to expand.

Implementing Push and Pop in the Target Domain

The student next drew the box and arrow diagrams for implementing *push* and *pop* using a linked list. He also wrote the Pascal code.

After writing *push*, the student asked if the capacity limitation that results when the spring is fully compressed is relevant in the new domain. The tutor told the student that the *physical* elements of the analogy (springs, movement of plates, etc.) do not apply. The student then asked if the concept of capacity limitation applies even if the spring doesn't. The tutor responded that although capacity limitation is an important concept, the student should disregard it for now.

3. Issues for Specifying a Theory of Analogical Problem-Solving

3.1. *A General Theoretical Framework*

We begin the discussion of our work by describing the class of theory that emerges from the current literature on analogical learning.[1,3,5,8,25,10,11,15,19,28,20,22,31,32]

Converging evidence suggests a model that embodies the following set of processes:

- Retrieval: A relevant conceptual model of a familiar "base" domain is retrieved from memory. This retrieval process is incremental. Base domain models relevant to the emerging "target" domain will be retrieved throughout learning. The retrieval process is affected by factors such as the problem-solver's purpose, his general world knowledge, his knowledge of various base domains, and, importantly, his preexisting knowledge of the target domain.
- Mapping: Correspondences are established between the entities in the base domain model and the currently known entities in the target domain. The base domain model is then mapped to form a model of the target domain.
- Evaluation: Evaluation can be thought of as an experimentation process. The learner attempts to use the newly mapped model to solve problems. The generation of these problems is equivalent to the generation of a set of experiments to test the newly mapped model's accuracy and sufficiency. As we discuss later, part of our goal is to build a system that models the way human learners actively and strategically seek to generate these experiments.
- Debugging: The results of the evaluation process are used to extend and correct the model of the target domain during debugging. Each analogical example is, by definition, only partially correct. As a result, we treat debugging as a central issue in analogical learning.
- Generalization: Here the structures shared by the analogically related domain models may be abstracted away from both domains, to form a more general understanding.

A sufficient account of analogical learning in complex domains must provide detailed specifications of each of the above processes. In this chapter we concentrate on the mapping, evaluation, and debugging components of analogical learning. In Section 4 we touch on retrieval.

Figure 12.1 diagrams the mapping, combined evaluation and debugging, and problem-solving components of our system. The system's mapper takes as input a base domain model and a list of the correspondences between elements in the base model and already known target elements. The mapper produces tentative target domain models, which are then debugged and

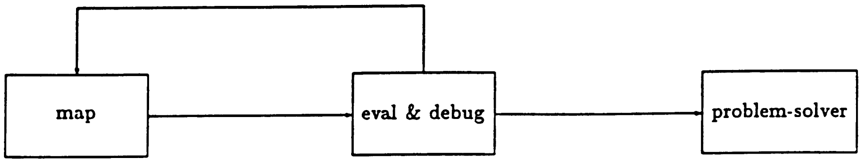


FIGURE 12.1 Components of the analogical reasoner.

evaluated. The debugged models are then passed to a problem-solving component that can generate box and arrow diagrams and Pascal code. Additionally, in our system, the output of the debugger can be used to guide subsequent mappings.

3.2. *Purpose-Constrained Mapping*

Purpose provides an essential constraint in problem-solving, but current implementations of cognitive theories do not make heavy use of this constraint.* The argument for why purpose is necessary in constraining problem-solving runs as follows. Reasoning about a complex domain requires understanding a number of distinct aspects of the domain and the relationships among those aspects.[2,7,12] Given the constraints of the cognitive system, it is not possible to learn all of these various aspects at one time. Rather, to make learning of a complex domain more tractable, students and instructors typically focus on individual purpose-defined aspects of the domain and, one at a time, map partial models from more familiar analog domains.[8,9] These partial models can later be integrated to provide a more full understanding of the target domain.[7]

One section of the protocol data described above provides an example of the way in which purpose can successfully constrain the mapping process by limiting attention to currently relevant aspects of the domain being learned. In teaching the student about stacks, the tutor first explained the LIFO behavior of a stack without regard to the underlying mechanism. The student used this focus of attention suggested by the tutor. He proceeded to think of ways in which he might have unknowingly encountered the use of stacks in programming; he suggested that stacks might be useful in implementing subroutine calls. He then generalized their usefulness, stating that whenever a task had an unmet precondition it would be useful to delay execution of the task by pushing it onto a stack. Note that the student ended up with a sophisticated and correct understanding of *one aspect* of the domain. This enabled him to then sucessfully turn his attention to the mechanism underlying the now well-understood behavior.

*Two notable exceptions are Thagard and Holyoke,[19,28] and Kedar-Cabelli,[22] who also stress the theoretical importance of purpose.

3.2.1. Implementing a Purpose-Guided Mapper

In our computational description, the learning process starts with this selection and mapping of purpose-constrained aspects of the target domain. Our mapping mechanism focuses on partial models of the base domain whose type reflects the problem-solver's purpose, and maps these models, type by type, over to the target domain.

Basically, our mapper uses the focus suggested by the tutor to select a model from the set of models in the base domain. The selected model is then mapped by using whatever cross-domain correspondences may have been supplied by the tutor to replace base domain elements of the model with target domain elements. The mapper also builds, for each element mapped into the target, the "template definitions" described below. Additionally, if our mapper is currently mapping a causal model that provides an underlying mechanistic explanation for a previously mapped "higher-level" behavioral model, the results of having debugged the higher-level model will guide the current mapping (Section 3.3.1). Note that we define a behavioral model as one that describes the results of an action and a causal model as one that describes how that action happens. The behavioral model can then be thought of as "higher-level" than the causal, in that the behavioral model can provide an explanation of an *underlying* mechanism.[2,30]

Our mapping algorithm is similar to Gentner's[17] in that it uses SME's* central principle of mapping structure by mapping objects connected or "structured" by relations. The relations are referred to as "predicates" in that they are aspects predicated to be true of the related objects.†

There are, however, four ways in which our mapping algorithm differs from SME. (1) We have implemented a purpose-guided focusing mechanism that allows our mapper to select a model of a specified type from a base domain containing a variety of models. As discussed below, this allows incremental learning and debugging. (2) The results of debugging early mapped models will guide the mapping of later mapped models that underlie them. As a result, the mapping process becomes more focused as more is known about the relationship between the base and target domains (Section 3.3.1). (3) Descriptive predicates will be mapped by our mapper, but not by SME. We have chosen to map descriptive predicates because we have found that they can have a role in the functioning of the model. For example, as shown in Figure 12.2a, because *full* is a predicate describing the state of its argument, *stack*, it is likely that SME will not map it over; this will make it difficult for SME to produce a target domain model that specifies the current

*Gentner's Structure Mapping Theory[17] is implemented in Brian Falkenhainer's Structure Mapping Engine, SME.[15]

†For example, if Fido the dog is my faithful companion we can predicate a companion relationship between Fido and Beth Adelson, companion (beth, fido). We could then use that relationship via a mapping process to understand the relationship between other pets and humans.

$$\begin{tikzcd}
\mathit{push}(\mathit{plate},\mathit{stack}) \arrow[r] \arrow[dr] \arrow[dd] & \mathit{on\text{-}stack}(\mathit{plate},\mathit{stack}) \\
 & \mathit{on}(\mathit{plate},\mathit{old\text{-}top\text{-}plate}) \\
\mathit{increase}(\mathit{full}(\mathit{stack})) &
\end{tikzcd}$$

(a)

$$\begin{tikzcd}
 & \mathit{push}(\mathit{node},\mathit{stack}) \arrow[d] & \\
 & \mathit{put\text{-}on}(\mathit{node},\mathit{stack}) \arrow[r] \arrow[dl] \arrow[d] \arrow[dr] & \mathit{on}\text{-stack}(\mathit{node},\mathit{stack}) \\
\mathit{increase}(\mathit{number}(\mathit{node\text{-}set})) & \mathit{increase}(\mathit{compression}(\mathit{spring})) \arrow[d] & \mathit{on}(\mathit{node},\mathit{old\text{-}top\text{-}node}) \\
 & \mathit{increase}(\mathit{full}(\mathit{stack})) &
\end{tikzcd}$$

(b)

FIGURE 12.2. Push: behavioral and causal models in the base domain. (a) Behavioral model in the base domain; (b) causal model in the base domain.

fullness of a stack. However, other problem-solving processes may need a model that contains this attribute, so that, for example, a stack that is already full will not be pushed onto. For this reason, we prefer to keep descriptive predicates until after mapping, and then have a debugger get rid of extraneous features that can be found not to participate in problem-solving. (4) Whereas SME tends to drop a base domain entity whose target analog is unknown, our mapper leaves such an entity in the newly mapped model. This allows our system's evaluator to find domain-appropriate replacements. It results in newly mapped models that have the complexity of the models from which they were derived (Section 3.3.2).

Gentner's Structure Mapping Theory[17] is powerful in that it describes a process that serves the purpose of analogical mapping. That is, structure mapping provides the learner with a way to view a barely familiar target domain as a structured and, therefore, functional system. (And hence one that is useful in problem-solving.) Building on Gentner's idea, we have implemented a mapper that preserves structure. Our theory additionally asserts that the mapper should produce a model that maintains the functionality of the source model (points 3 and 4 above) and should make use of acquired knowledge about cross-domain differences (point 2 above).

3.2.2. A Sample Mapping

The following illustrates a mapping produced by our system:

In this example our overall goal is to have the system model the problem-solving in our protocol. That is, we want the system to learn about computer science stacks and stack operations by analogy to cafeteria stacks and to produce box and arrow diagrams and code for the operations *push* and *pop*. To accomplish this, the system begins by taking the tutor's suggestion to focus initially on the behavioral model for *push*. In addition to supplying a focus, the tutor also supplies a list of base and target domain correspondences stating, for example, that *push* in the base corresponds to *push* in the target and plates correspond to nodes. The system selects the suggested model from a base domain containing behavioral and causal models of both *push* and *pop*. The selected model is then mapped into the target domain. The behavioral and causal models for *push* in the base domain can be seen in Figure 12.2a and 12.2b, respectively. How these models are used in the learning process is discussed in the sections that follow.

In Figure 12.3 we see the behavioral model of the target domain produced by our mapping mechanism. What is important to note here is that the nature of the models allows them to be used in the problem-solving that is the system's ultimate goal. That is, tracing along the arrows in, for example, the causal model in Figure 12.2b we see the chain of events that occur when *push* is performed.[26,27] Following the model right to left and depth first: Pushing is equivalent to putting a new node on the stack; this new node is on top of what previously was the top node of the stack. *However*, having put the new node on the stack *causes* the spring to become further compressed; this *causes* the stack to increase in fullness; and the number of nodes in the stack increases.

The format of the models is represented graphically here. In actuality, within the system, the models contain instructions for carrying out the causal chain of state changes described in the diagrams. That is, they are composed of instructions for computing an increased value for the compression of the spring and then based on that new value instructions for computing an increased value for fullness. These instructions are carried out by the system's *simulation machine*.[1,16,13,18] The simulation machine's output is a list of the current values of the variables that describe a state after the operation described in the model has been performed.

$$\begin{tikzcd}
\textit{push(node,stack)} \arrow[r] \arrow[dr] \arrow[dd] & \textit{on-stack(node,stack)} \\
 & \textit{on(node,old-top-node)} \\
\textit{increase(full(stack))} &
\end{tikzcd}$$

FIGURE 12.3. Newly mapped behavioral model for *push* in the target domain.

Additionally, the format of the models allows them to be examined by a debugger, (Sections 3.3.1 and 3.3.2) to be used to generate box and arrow diagrams after debugging.

Although the contents of the models were inferred through protocol analysis, the assertions in the model can be formalized to allow, for example, the deduction that the *old-top-node* is the *last-pushed-node*. Additionally, because the predicates specify computations that result in state changes they have an underlying "procedural" semantics.

The following section describes how our system debugs the newly mapped model of *push* so that problem-solving can be carried out successfully.

3.3. Debugging a Newly Mapped Model

3.3.1. Actively Seeking out Bugs

In our theory, debugging is characterized by an active search for bugs, one that is initiated by the system rather than by the tutor. The base domain model is known, by definition, to provide an imperfect model of the target domain. The base model may contain inappropriate elements that require deletion or transformation; or it may require additional knowledge specific to the target domain.

Our current example illustrates the case in which a newly mapped model contains a concept that is inappropriate in the target domain and needs to be deleted. In subsequent sections we deal with increasingly complicated cases of transforming a newly mapped model. Looking at the behavioral model that our mapper produced for the target domain (Figure 12.3) we see that pushing a node onto a stack that is implemented as a list of nodes, leads to the stack being more full. However, the system contains prior knowledge about the target domain that asserts that lists of nodes are used when a data structure without a prespecified capacity limitation is desired.* Since linked lists have no specified capacity limitation there is an inconsistency between the newly mapped model and prior knowledge of the target domain. The system must have the ability to notice and resolve this inconsistency.

Identifying and Fixing Bugs in a Runnable Model

Here we describe how the system's evaluation and debugging mechanism resolves the "fullness bug" in the course of evaluating the behavioral model of *push*. The system's *evaluator* looks in turn at each element of a newly mapped model in an attempt to determine whether each element it encoun-

* For this example we have supplied the system with the same knowledge of the target domain that our novice programmer had. We have given it models for performing typical operations on variables, arrays, and linked lists. It also has world knowledge about boxes and containers.

ters is appropriate in light of the domain the model has been mapped into. In order to allow the evaluator to carry out the evaluation, the system has been given several kinds of knowledge, labeled K1–3 below:

K1. Any element that occurs in a model has a definition, a *template* consisting of a set of features.[6,29,31,32] Elements in the model are either objects (e.g., *stack*) or predicates. Predicates describe attributes of, or actions on, objects (e.g., *fullness* or *push*, respectively). For objects, one feature in the template specifies the class it belongs to. For predicates, the class of both the predicate and the objects it applies to (its arguments) are listed. For example, the predicate *full* has a template that specifies that *full*'s class is a measurement of capacity; that *full* takes a container as an argument and that container's class is an object with a limited capacity.

K2. The system knows not only which specific objects and predicates are appropriate to each domain, but also which *classes* of objects and predicates are appropriate. For example, the system knows that integer variables in particular, and data structures in general, are appropriate in the computer domain.

K3. The system has general knowledge about how analogical correspondences are meant to be taken. For example, the system contains knowledge that physical contiguity in the base can be appropriately thought of as corresponding to virtual contiguity in the target.

The system uses the knowledge described above in applying rules that allow it to evaluate each element in the newly mapped model. The rules, labeled R1–4, are

R1. Infer that an element currently in the target domain is appropriate in the new model.

R2. If the element is not currently in the target domain but it is of a *class* currently in the target domain, infer that a modified version of the element is appropriate in the new model and use the existing domain definition of the class to modify the element.

R3. If the newly mapped element belongs to a class that has a corresponding class already existing in the target domain (point K3), infer that a modified version of the element is appropriate and use the existing target domain definition to modify the model. (See modfication example below.)

R4. A predicate can only be applied to an argument of an appropriate type; that is, fullness cannot be predicated of a container without capacity limitations (point K1).

We see each of the above rules (and the knowledge they embody) being applied as we follow the evaluator working its way through the model for *push* in Figure 12.3. (The convention we are using for reading through the models is breadth-first, right to left.) Starting at the root of the tree, the evaluator encounters the element *push*. Because the tutor had specified that *push* in the base corresponded to an asserted, but as yet undefined, version of *push* in the target, the system infers that *push* is appropriate to the model and turns to the predicates that follow from it.

Although *on-stack* does not yet exist in the target, its definition states that it is a "membership relation." Applying rule R2, the system finds that other predicates in the target are membership relations [e.g., the Pascal function *in*(*set*)] and, therefore, the system hypothesizes that the predicate holds.

The evaluator next comes to the predicate *on*. The template for *on* states that it is a "physical contiguity relation." The system knows that physical contiguity in the base corresponds to virtual contiguity in the target (rule R3). It therefore makes this change to the predicate's template and then infers that the predicate holds. This is an example of the system's ability to model human problem-solvers in interpreting analogical correspondences in an appropriate, nonliteral manner.

The evaluator turns to the predicate *full*; it finds that *full* is potentially appropriate in that it already exists in the target as knowledge that arrays can be full (rule R1). However, the evaluator finds that fullness can only be predicated of containers having capacity limitations (rule R4). It knows that the stack is being implemented as a list of nodes and that lists do not have capacity limitations. The system suggests that the concept fullness should be removed from the model. It then removes fullness. At this point, if the system is told that the problem arose because no capacity limited containers were being used in this example it will also remove all other predicates whose definitions involve capacity limitations. Figure 12.4 shows the debugged version of push after fullness has been deleted.

But more mileage can be gained from this evaluation. The system has just mapped and debugged the behavioral model. It will now go back and map the causal model using information gained in debugging the behavioral one. When this mapping begins, the system will take note of any elements that have been deleted from the behavioral model (in this example, *full*). Pieces of the causal model that only explain already deleted behavior will not be mapped.

For example, the dashed branch of the causal model in Figure 12.5 will not be mapped, since its only role in the model is to provide a way of computing fullness by specifying that fullness can be determined by considering the

$$\begin{tikzcd}
\textit{push}(\textit{node},\textit{stack}) \arrow[r] \arrow[dr] & \textit{on-stack}(\textit{node},\textit{stack}) \\
 & \textit{on}(\textit{node},\textit{old-top-node})
\end{tikzcd}$$

FIGURE 12.4. Behavioral model for *push* in the target domain: debugged version.

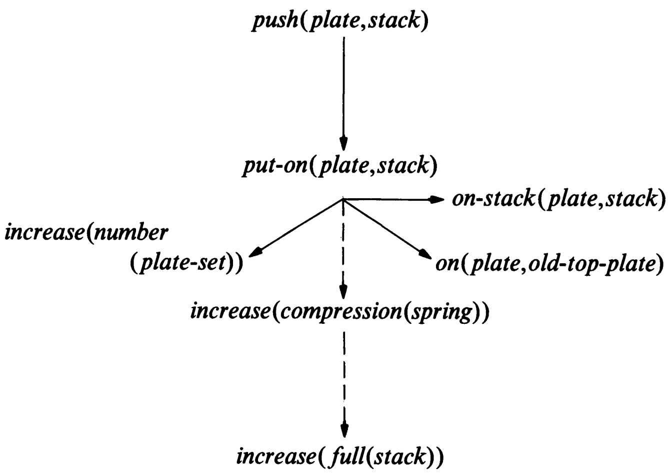


FIGURE 12.5. Causal base model.

current degree of spring compression* in relation to the maximum possible compression.†

This means that the model in Figure 12.7 rather than the more complex one in Figure 12.6 will be mapped into the target domain. As a result, the debugger will have a simpler causal model to deal with; one in which the mechanism supporting the inappropriate concept of full has already been removed. As illustrated here, this strategy of incrementally mapping partial models and using earlier mappings to guide subsequent ones, can simplify the potentially complex process of debugging causal models.

Using the procedure described above, the system now debugs the causal model in Figure 12.7 modifying the definition for *put on* according to its definition in the target domain. This debugging procedure results in the causal model shown in Figure 12.8. Reading from left to right we see that the model now correctly specifies the sequence of actions that make up push (setting the new node's next-pointer to the top node in the stack, setting the stack's head-pointer to the new node). It also specifies the results that follow from these actions (the new node being on the stack, etc.). This model will now be passed to the "box-and-arrow-drawing" portion of the system's problem-solving component. This component of the system can, using the sequence of actions specified in the model, generate box-and-arrow diagrams

*We present this model where the fullness of the stack is calculated using the compression of the spring because it reflects the model used in our protocol. In the next section we present the intuitively appealing example of a base domain model in which fullness is calculated using the number of plates in the stack.

†Recall that the system models contain instructions that specify how to compute fullness based on spring compression.

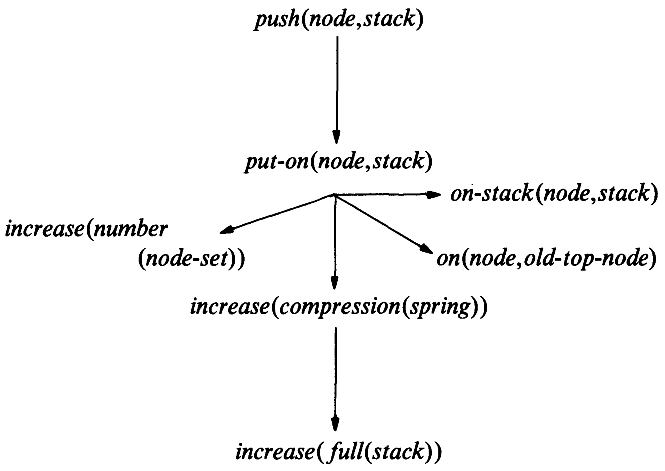

push(*node*,*stack*)
put-on(*node*,*stack*)
on-stack(*node*,*stack*)
increase(*number* (*node-set*))
on(*node*,*old-top-node*)
increase(*compression*(*spring*))
increase(*full*(*stack*))

FIGURE 12.6. Newly mapped causal model with fullness.

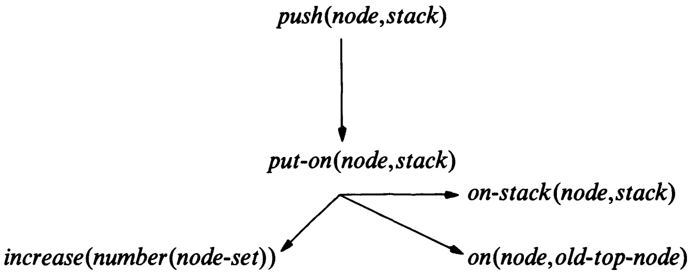

push(*node*,*stack*)
put-on(*node*,*stack*)
on-stack(*node*,*stack*)
increase(*number*(*node-set*))
on(*node*,*old-top-node*)

FIGURE 12.7. Newly mapped causal model with fullness.

of *push* on a computer screen. That is, the "box-and-arrow" portion of the system has knowledge that allows it to carry out the type of actions that occur in the models (such as *set-next-pointer*). These actions are carried out in a way that causes diagrams representing the actions in the models to be drawn on the computer's screen.

3.3.2. Transforming a Mapped Model: Reasoning About Simulations

In the example just described we considered the case where an element needs to be deleted. Our system also handles more complex cases in which a model needs to be *transformed* in various ways by finding correspondences between a target-domain inapropriate element mapped over from the base and existing target domain elements.

The first case in which a model needs this type of transformation is illustrated by the example in which the goal is to implement a stack using an array as opposed to the previous example in which a linked list was used. The

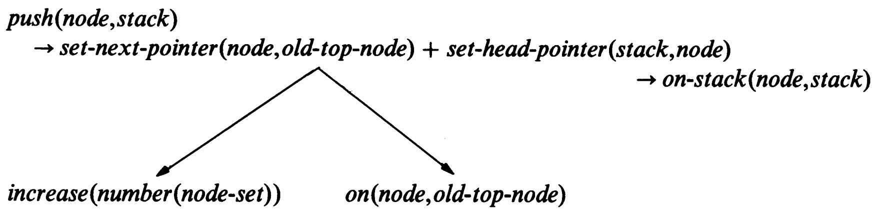

push(*node*,*stack*)
→ *set-next-pointer*(*node*,*old-top-node*) + *set-head-pointer*(*stack*,*node*)
→ *on-stack*(*node*,*stack*)
increase(*number*(*node-set*))
on(*node*,*old-top-node*)

FIGURE 12.8. Debugged causal model in the target.

representation of the base domain is the same as it was for our previous example. Knowledge of the target domain still consists of information about variables and arrays but, this time, not about linked lists. The system again has runnable models for typical array operations such as initialization and search.

The behavioral model of push is again mapped into the target domain. This time no changes are made in the behavioral model; the fullness of the stack is found to be consistent with the system's knowledge of the capacity limitation of an array. After mapping the behavioral model, the system maps the causal model of the stack into the target domain and then begins to evaluate and debug it. During this process the system questions the tutor on the appropriateness of the spring in the causal model (Figure 12.9). (The system's knowledge specifies that physical objects like springs do not belong in models of computer memory.)

The tutor tells the system that the domain-appropriate *functional analog* of the spring needs to be found. In finding the functional analog of the spring the system will draw on several types of *relational knowledge* (RK1–3). This relational knowledge is *learned* by the system; the system notices and stores

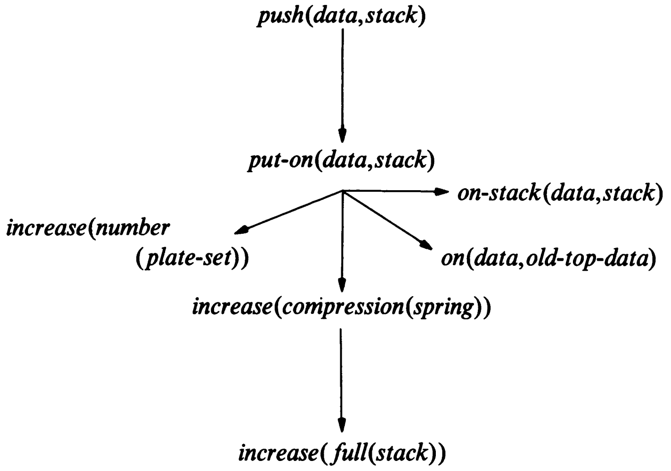

push(*data*,*stack*)
put-on(*data*,*stack*)
on-stack(*data*,*stack*)
increase(*number* (*plate-set*))
on(*data*,*old-top-data*)
increase(*compression*(*spring*))
increase(*full*(*stack*))

FIGURE 12.9. Newly mapped causal model of pushing onto an array.

these relations whenever it acquires a new model. They do not have to be hand-coded by the programmer.

RK1. The system contains functional to structural mappings; knowledge relating state changes and the mechanisms causing them.[2,13,16,21] For example, it knows that "changes in fullness are supported by changes in the mechanism comprised of the spring, the set of plates, etc."

However, we want to stress that the system's knowledge does not contain any explicit statement concerning *how* changes in fullness are related to changes in the spring. This is what needs to be determined.

RK2. The system has knowledge relating actions and the state changes they produce. It knows, therefore, that "pushing leads to changes in fullness."

RK3. The system also has knowledge relating actions and the mechanisms involved. It knows that "pushing involves a change in the spring."

In order to find the piece of target domain mechanism with the same function as the spring, the system will find what sort of state change in the base is associated with a particular change in the spring. It will then turn to the target, look at the parallel state change, and determine what piece of mechanism is effected in the way that the spring was. To do this the system first needs to focus on the base and find what state changes the spring is involved in. It examines its knowledge of functional to structural mappings (RK1) and finds that the spring is involved in changes in fullness. Now, in order to find out the *nature* of the relationship between changes in fullness and changes in the spring, the system runs a simulation of *push* and obtains values for the fullness of the stack and the compression of the spring before and after the simulation is run. The system then compares the direction of change in both fullness and spring compression and finds that there is a positive relationship between the two. Currently, the system can recognize positive and negative correlations, as well as the lack of relationship between two state variables. It is possible to expand this part of the system to include the recognition of more complex, but regular relationships.

The system now needs to find what piece of mechanism in the *target* domain changes for the same reason and in the same way as the spring (i.e., increases with fullness). The system begins by retrieving an operation in which fullness increases. It will then run this operation and look for pieces of mechanism that register increases in fullness. Target domain knowledge about the relation between actions and state changes (RK2) asserts that initializing an array causes fullness to increase; the system simulates the process and finds that in the target, it is the array-index that increases with fullness.

As a result of this process, in which corresponding simulations are sought, run, and evaluated for the purpose of finding functionally analogous mechanisms, the system correctly hypothesizes that the array index is the analog of

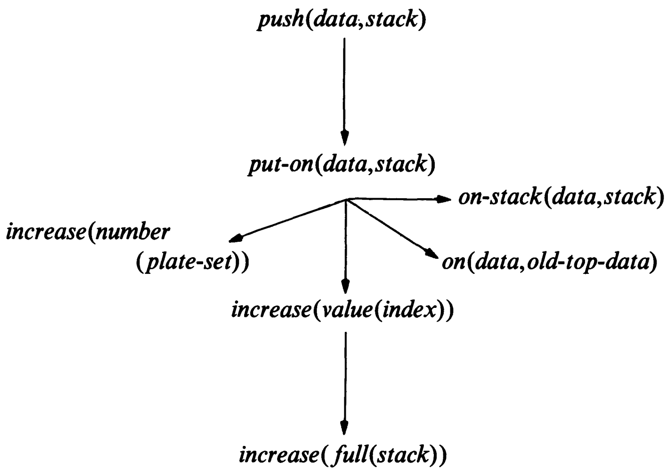

push(*data*,*stack*)
put-on(*data*,*stack*)
on-stack(*data*,*stack*)
increase(*number* (*plate-set*))
on(*data*,*old-top-data*)
increase(*value*(*index*))
increase(*full*(*stack*))

FIGURE 12.10. Partially debugged causal model of pushing onto an array.

the spring. The system can then substitute the array-index for the spring (Figure 12.10). This method of finding functional analogs for base domain mechanisms allows the system to choose and maintain the part of a causal analogy that is appropriate across the base and target domains. It reflects the way in which human problem-solvers understand both the limitations and the utility of analogical examples.

3.3.3. Breaking Ties

The system is also able to deal with a variation on the previous example of a transformation. That is, it is able to break ties when more than one piece of target domain mechanism is found to be analogous to a piece of the base. In these cases the ties are broken by considering the role that the base domain object played in the base model and comparing that to the role played by the tied target objects. A situation in which the system needs to choose among competing potential analogs is illustrated by considering an example in which the base domain model and the preexisting target domain knowledge are different than they were in the previous example. For this example, in which the stack is again being implemented using an array: (1) A base domain model is used in which fullness is calculated using the number of plates in the stack. As mentioned above, this is an intuitively appealing representation, although it differs from the one used in our protocol. This situation produces the target model shown in Figure 12.11. (2) The target domain includes the concept of a *data-set*, which comprises the contents of an array.

When the evaluator reaches the branch that contains the assertion that the number of plates in the *plate-set* increases as a result of *push*, it questions

$$\begin{tikzcd}
\textit{push(data,stack)} \arrow[d] & \\
\textit{put-on(data,stack)} \arrow[r] \arrow[dr] \arrow[dd] & \textit{on-stack(data,stack)} \\
 & \textit{on(data,old-top-data)} \\
\textit{increase(number(plate-set))} \arrow[d] & \\
\textit{increase(full(stack))} &
\end{tikzcd}$$

FIGURE 12.11. Causal model using plates to calculate fullness.

the appropriateness of the *plate-set*. Once again, the tutor tells it to find the functional analog. Using the method described just above in Section 3.3.2 the system finds that the number of elements in the array's data-set *and* the value of the array's index both increase as an array is initialized. In order to determine which is the better analog for the plate-set the system now looks at the *role* played by the plate-set in the act of pushing. The system looks at the template definition for *push* and finds that for this action, the plate-set is the *contents* of the container. Turning to the target domain, the system finds that during array initialization, the data-set also serves as the *contents* of the container, whereas the index serves to locate the cell currently being initialized. As a result of comparing the roles of the index and the data-set to the role of the plate-set, the system breaks the tie; it decides that the data-set provides a better analog to the plate-set than does the index. The system then replaces *plate-set* with *data-set* in the model.

4. Future Work

There is an aspect of the debugging process that still needs to be addressed. Now that the models mapped from the base domain have had changes made to them, they must be checked to see that they are still sufficient. We are in the process of implementing a mechanism to do this through a series of simulations designed to test that the models still exhibit aspects of LIFO behavior that the system knows are important. For example, pushing and then popping a set of elements must result in reversing their ordering.

Additionally, we have not discussed the retrieval process. Currently, we are working on expanding our code generator to reflect the repeated retrievals we have observed. These retrievals produced at least two *qualtitatively different* kinds of examples, used in two different kinds of processes, both of which are analogical in nature. One is a process in which the surface features of past examples are put into correspondence with the surface features of the current problem. This is strongly reminiscent of the process described by Anderson and Pirolli.[5,25] The second is a process of reasoning from the constraints made explicit in diagrams that depict the actions involved in the concept being learned. The models that have been presented in this chapter are meant to serve as underlying representations for these drawings. We need to address the issue of which kind of process occurs at different stages in problem-solving in order to predict which kind of analogy will be retrieved at a given time.

Last, we have already collected protocols on the task of turning a stack into a queue. This will allow us to address several issues. (1) The problem-solving involved will put more pressure on the processes and the representational format specified in our theory. (2) Because turning a stack into a queue requires reasoning about the relationship between structure and function, we will be able to look at that important issue. (3) We will also be able to make statements about the important, but less frequently studied, process of within domain analogy, which is highly important or engineering design.

5. Summary and Conclusions

We have presented a discussion of three of our system's mechanisms: one for mapping, one for evaluating mapped models, and one for debugging inconsistencies. We have implemented a purpose-constrained mapper that reflects the way students increasingly limit their focus of attention as more is known about the relationship between a base and a target domain. We have also implemented an evaluation mechanism that identifies inconsistencies as elements of newly mapped models are checked to see if they are the sort of elements that are known to exist in the target domain. In doing so the evaluation mechanism uses knowledge about the nature of the base and target domains and the way in which relations apply across analogous domains. The evaluation mechanism reflects the nonliteral way in which analogies are understood. Finally, we have presented a debugging mechanism that maintains functional aspects of base models while adding target-appropriate causal explanations. This allows the system to model the way human problem-solvers select and use the level of explanation that is appropriate in an analogical example.

The development of the theory has been possible because we have worked within a problem-solving context, reflecting the purpose of analogical reasoning. This approach has yielded insights into the nature of the phenomenon being modeled and, as a result, has allowed us to develop active, reasoning, and knowledge-intensive mechanisms that are characteristic of the nature of the analogical reasoning process.

References

[1] Adelson, B. (1989). Cognitive modeling: Uncovering how designers design. *The Journal of Engineering Design. 1*, 1.

[2] Adelson, B. (1984). When novices surpass experts: How the difficulty of a task may increase with expertise. *Journal of Experimental Psychology: Learning, Memory and Cognition*, July.

[3] Adelson, B., Gentner, D., Thagard, P., Holyoak, K., Burstein, M., and Hammond, K. (1988). The role of analogy in a theory of problem-solving. *Proceedings of the Eleventh Annual Meeting of the Cognitive Science Society.*

[4] Brian, G. (1926). *Edison: The man and his work.* Garden City, NY: Knopf.

[5] Anderson, J., Farrell, R., and Sauers. (1985). *Cognitive Science.*

[6] Burstein, M. H. (1983). Causal analogical reasoning. Michalski, R. S., Carbonell, J. G. and Mitchell, T. M. (Eds.), *Machine Learning: Volume I.* Los Altos, CA: Morgan Kaufman Publishers, Inc.

[7] Burstein, M. H. (1986). Concept formation by incremental analogical reasoning and debugging. In Michalski, R. S., Carbonell, J. G. and Mitchell, T. M. (Eds.) *Machine Learning: Volume II.* Los Altos, CA: Morgan Kaufmann Publishers, Inc.

[8] Burstein, M., and Adelson, B. (1987). Mapping and integrating partial mental models. *Proceedings of the Tenth Annual Meeting of the Cognitive Science Society.*

[9] Burstein, M., and Adelson, B. (1992). Analogical reasoning for learning. In R. Freedle (Ed.) *Applications of Artificial Intelligence to Educational Testing.* Hillsdale, NJ: Erlbaum.

[10] Carbonell, J. G. (1983). Transformational analogy. Problem solving and expertise acquisition. In Michalski, R. S., Carbonell, J. G. and Mitchell, T. M. (Ed.), *Machine Learning: Volume I.* Los Altos, CA: Morgan Kaufman Publishers, Inc.

[11] Carbonell, J. G. (1986). Derivational analogy: A theory of reconstructive problem solving and expertise acquisition. In Michalski, R. S., Carbonell, J. G. and Mitchell, T. M. (Ed.), *Machine Learning: Volume II.* Los Altos, CA: Morgan Kaufman Publishers, Inc.

[12] Collins, A., and Gentner, D. (1983). Multiple models of evaporation processes. In *Proceedings of the Fifth Annual Conference of the Cognitive Science Society.* Rochester, NY: Cognitive Science Society.

[13] de Kleer, J., and Brown, J. S. (1985). A qualitative physics based on confluences. In D. Bobrow (Ed.), *Qualitative Reasoning about Physical Systems.* Cambridge, MA: MIT Press.

[14] Ericsson, A., and Simon, H. (1983). Verbal reports as data. *Psychological Review*, *87*(3), 214–51.

[15] Falkenhainer, B., Forbus, K., and Gentner, D. (1986). The structure-mapping engine. In *Proceedings of AAAI-86.* Los Altos, CA: Morgan Kaufman, Inc.

[16] Forbus, K. (1985). Qualitative process theory. In D. Bobrow (Ed.), *Qualitative Reasoning about Physical Systems.* Cambridge, MA: MIT Press.

[17] Gentner, D. (1983). Structure-mapping: A theoretical framework for analogy. *Cognitive Science*, *7*(2), 155–70.

[18] Hammond, K. (1988). *A Theory of Planning.* New York: Academic Press.

[19] Holyoak, K., and Thagard, P. (1987). Analogical Mapping by Constraint Satisfaction. *Cognitive Psychology.*

[20] Kolodner, J., and Simpson, R. (1984). Experience and problem solving: A framework. In *Proceedings of the Sixth Annual Conference of the Cognitive Science Society*. Boulder, CO: Cognitive Science Society, pp. 239–243.

[21] Kuipers, B. (1985). Commonsense reasoning about causality. In D. Bobrow (Ed.), *Qualitative Reasoning about Physical Systems*. Cambridge, MA: MIT Press.

[22] Kedar-Cabelli, S. (1984). Analogy with purpose in legal reasoning from precedents. Technical Report 17. Laboratory for Computer Science, Rutgers, NJ.

[23] Newell, A., and Simon, H. A. (1972). *Human Problem Solving*. Englewood Cliffs, NJ: Prentice-Hall.

[24] Okagaki, L., and Koslowski, B. (1987). Another look at anologies and problem-solving. *Journal of Creative Behavior*. *21*(1).

[25] Pirolli, P. and Anderson, J. (1985). *Canadian Journal of Experimental Psychology*.

[26] Schank, R., and Abelson, B. (1977). *Scripts, Plans, Goals and Understanding*. Hillsdale, NJ: Erlbaum.

[27] Schank, R., and Riesbeck, C. (1981). *Inside Computer Understanding*. Erlbaum: Hillsdale, NJ.

[28] Thagard, P., and Holyoak, K. (1985). Discovering the wave theory of sound: Inductive inference in the context of problem solving. In *Proceedings of the Ninth IJCAI*, Los Altos, CA: Morgan Kaufmann Publishers, Inc., pp. 610–612.

[29] Waltz, D. (1982). Event shape diagrams. In *Proceedings of the National Conference on AI.*

[30] Winograd, T. (1974). *Natural Language Understanding*. New York: Academic Press.

[31] Winston, P. (1977). Learning by creating and justifying transfer frames. TR 414a. MIT AI Lab. 1977.

[32] Winston, P. (1982). Learing new principles from precedents and exercises. *AI*.

13 Entropy Measures in Engineering Design

Ronald S. LaFleur

Abstract. The development of a design science requires that progress made through research and technology be accountable. The difficulty in measuring progress lies in the different points of view of researchers, teachers, managers, and practitioners. This is compounded by design issues such as specification fuzziness, individual/team decision making, multifunctional design, and concurrency in the product development. A common, universal measure is needed. A commonality between problems is that mass, energy, and information are stored and transferred in the product or technical system. The entropy function has the power to integrate the mass, energy, and information measures of multifunctional problems into one measure. This provides a way to measure, in an unbiased way, the efficacy of design solutions, design methods, technical systems, and the advancement of design science.

1. Introduction—Diversity in Engineering Design

Diversity is a beneficial character of design evolution of products, engineering methods, or technical systems. The more choices available, the higher the chance of satisfactory progress in engineering. The diversity in engineering methods, design research, multifunctional problems, and engineering environments creates a problem when measuring alternatives in order to make comparisons and insure progress.

Diversity is a necessary feature of engineering design methods. On a very basic level, there is a lack of information in design problems. The designer must provide information as design specifications, decisions, and constraints. The method and substance of these input conditions are different between countries, regions, companies, groups, and design problems. The selection of one method over time reduces the number of potentially successful methods. The chances for evolving the "best" method are highest if method diversity allows choices.

Engineering design research augments industrial practice and the teaching of design methods. Progress is the identification of important design-related

TABLE 13.1. Nomenclature

System		*Fields*	
Φ	General fitness functional	t	Time
V	Volume	Δt	Time differential
N	Mole number of species i	s	Mass specific entropy
E	Energy	e	Mass specific energy
I	Information	v	Mass specific volume
X	General variable vector	n	Mass specific mole numbers of species i
S	Entropy		
T	Temperature	ρ	Material density
p	Pressure	J	Flux rate
μ	Electrochemical potential	x	Spatial coordinate
M	Number of subunits	u	Velocity
Ω	Number of configurations	w	Molecular weight
K	Boltzmann's constant	F	Body force
P	Configuration probability	c	Concentration
ω	Frequency of occurrence	σ	Entropy production
D	Degrees of freedom	A	Reaction affinity
vl	Variation level	L	Phenomenological coefficient
		k	Thermal conductivity
Superscripts		*Subscripts*	
DES	Design environment	i	Electrochemical species
VER	Verification environment	k, l, p, q, m	Vector counters
CON	Construction environment	in	Flow inward
APP	Application environment	out	Flow outward
$\bar{}$	Overbar—averaged	E	Energy degree of freedom
E	Heat flux	0	Zero energy state
T	Temperature gradient	env	Environment value
μ	Electrochemical potential gradients	$(\ )_n$	Natural process
S	Entropy flux	$(\ )_v$	Virtual process
u	Velocity gradients		
N	Species flux		
V	Visco-plastic stress		
R	Electrochemical reaction		

variables and understanding relationships between them. In any one research endeavor, a diversity of conceptual and detailed ideas are considered from which candidate hypotheses evolve. The potential for the research to yield new understanding is higher when diversity is higher; diversity is good for design science.

The complexity of the design problem is highest when considering a number of different functions. For example, the external shape of a modern turbine blade is governed primarily by its aerodynamics and strength. The aerodynamic and fatigue physics, units, and methods of measure are different enough to constitute discrete disciplines. But disciplines are for engineering convenience. In the true situation, different physics interact. Measurement of these interactions is along the lines of established disciplines; one being the input to the other.

The various situations in design evolution can be distinguished by "engineering environments" of differing conditions, activities, and goals. For example, a test engineer works with prototypes while a manufacturing engineer works with the subunits and their assembly. Each environment has its own goals and, consequently, differing measures and units of "goodness" or fitness. Concurrent engineering attempts to meet the diverse concerns of application, manufacturing, and testing environments at the time of design decision making. This complicates the designer's job. The complexity can only be managed using design tools for the diverse concerns plus their interactions.

This chapter proposes a single measure that is universal and not arbitrary: the entropy function. The need for a single measure of fitness in engineering design and metadesign is explored in Section 2. This is followed by a description of a viable single measure, the thermodynamic entropy in Section 3. In successive sections, the single measure is discussed in terms of its ability to represent different scales, variable types, and levels of variation.

2. The Need for a Single Measure—Single Measure Criteria

The required features of a universal measure are formed as a set of criteria addressing design effectiveness, mutual existence of natural and human processes, scale, residence in engineering environments, and uncertainty.

2.1. Representing Different Scales of Engineering Problems

The engineering of useful devices and the development of design methods are processes that occur on a variety of scales. For example, the design of a cleanroom and the design of VLSI microcircuits are on opposite ends of scale. This example can be stretched further by considering engineered materials and industry design. Scale is a measure of amount of material or smallest resolved length that is configured in the design.

Engineering problems are usually solved by "decomposing" the system into smaller, more manageable, design problems. Then the scale of a problem is determined by the scale of its subunits. Combination is the reverse of decomposition. However, the best system is not necessarily a combination of the best subunits due to boundary interactions. The boundaries are defined by the surface geometry and the types of fluxes they permit such as mass and energy.

Scale is a relative observation based on the form of a system and its subunits. The system decomposition process leads to a larger number of simpler design problems. The simplification is a byproduct of subunit specialization. A system often has complex multifunctional purposes. The subunit functions must contribute to the function of the system as

$$\Phi_{single} = \Phi(\Phi_{subunit\,1}, \Phi_{subunit\,2}, \ldots) \tag{13.1}$$

This interaction between form and function transcends scale.

The form-function of the different scale structures is related to physical behavior. Due to various boundary conditions on a structure, the collection of matter will respond or "behave." The physical laws that govern behavior are scale independent; e.g., conservation of mass, chemical species mass, mechanical and thermal energy. However, the detail of behavior and function description is only as good as the scale of the system and its subunits; i.e., the smallest resolved scale. To simplify through detailing, the engineer is motivated to decompose the system into subunits already designed and well characterized. This is the case when the subunits are known materials or off-the-shelf items. Therefore, a single measure of performance must be applicable over many scales occurring in design and engineering systems.

2.2. Representing Different Points of View

The engineering process involves a diversity of issues such as performance, manufacturability, and environmental impacts. Consequently, the process is affected by people with differing points of view. In a typical company, a variety of people all wearing different hats are involved with product development. Small businesses require individuals to wear many hats. A product or design method is initiated by a need stated by marketing personnel or managers. Project design engineers plan, configure, or analyze candidate components or systems. Manufacturing engineers interactively input requirements of fabrication and assembly. Technicians enact the manufacturing plan, turning raw materials into the product and assembling components. Test engineers verify product safety, performance, and tolerances. The primary product evolution ends when sales personnel enact packaging and shipping of the approved products. A secondary product cycle arises when servicing, failure, recycle, or disposal are considered.

Different personnel have different concerns in the product evolution. These concerns are measured differently. For example, the design performance of a heat exchanger is measured by effectiveness (actual heat exchange divided by the maximum possible exchange) while material cost is measured in parts such as tubing, 180 degree bends, thin fins, and solder. The overall fitness of a design or method must reflect diverse issues and accommodate various measures.

The evolution of products or engineering methods can be detailed by a sequence of single decisions. A single decision is a yes or no to either accept or reject choices. A single decision is quantified because yes and no are discrete values of a single variable. The single decision must weigh the different issues and integrate them into one measure. The functional relationship between the many special measures and the single decision variable has been arbitrary. A special example is a weighted sum of presumed independent measures such as

$$\Phi_{single} = 1.8\Phi_{aerodynamics} + 0.9\Phi_{manufacturing} + 2.5\Phi_{fatigue}. \tag{13.2}$$

The coefficients indicate the relative weight of the contributions while ensuring dimensional consistency of engineering or economic units. More generally, the presumably independent measures of special domains represent coordinate axes such that the single measure depends on them as

$$\Phi_{single} = \Phi(\Phi_{domain\,1}, \Phi_{domain\,2}, \Phi_{domain\,3}, \ldots). \tag{13.3}$$

The dependency of one measure upon another reduces the dimensionality but may increase the nonlinearity of the single decision variable.

Alternatively, the independent performance measures can be defined along the special types of fundamental fluxes in engineering systems. The fundamental physics of mass, chemical species, and energy fluxes suggest that

$$\Phi_{single} = \Phi(\Phi_{mass}, \Phi_{chemical\,species}, \Phi_{mechanical\,energy}, \Phi_{thermal\,energy}). \tag{13.4}$$

A single measure of performance must allow for a variety of physical effects, integrate the different measures of physical effects, and accommodate different domains of engineering and technology.

2.3. Information Flow Between Engineering Environments

On the technical system scale, engineering environments are defined in terms of the different activities that occur in the engineering method. Different tools and personnel, with their different points of view, act within four primary environments: application, design, verification, and construction (LaFleur, 1992). Information can flow between environments as shown in Figure 13.1 below.

The application environment is the origin of the product idea or need and raw materials. The product will encounter actual conditions and perform an

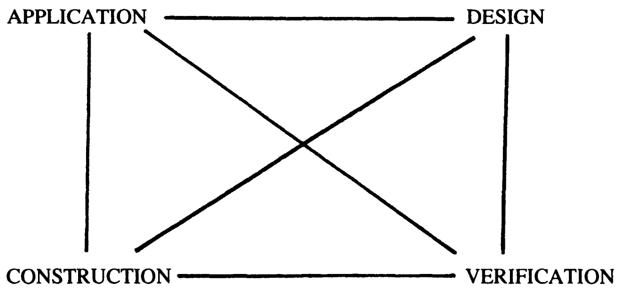


FIGURE 13.1. Information flows between engineering environments.

actual task in the application environment. In the design environment, the product, conditions, and functions are abstract. The abstract variables are determined by information gathering, input decisions, physical modeling, and solution techniques. In the verification environment, the candidate design is prototyped and instrumented to test for task satisfaction, tolerances, and safety. Computer simulations that test the design under artificial conditions are a part of verification. The abstract design is brought into reality in the construction environment. Raw material or subunits are shaped, treated, and assembled. The constructed product may then be verified for quality and tolerances.

The routing of the product through the environments establishes the engineering method. The location and routing is associated with time in the form of schedules and logistics. Branching of the path, such that information from different environments operates on the design simultaneously, is the cornerstone of concurrent engineering. The product is in different forms in the application, design, verification, and construction environments. Consequently, fitness is measured differently in each environment. The overall fitness of the engineering method should account for all special fitness measures as

$$\Phi_{single} = \Phi(\Phi^{DES}, \Phi^{VER}, \Phi^{CON}, \Phi^{APP}), \tag{13.5}$$

where the three-letter abbreviations are the engineering environments. The information flowing from one environment to another requires the use of common units of fitness measuring. A single measure of design and method fitness must be evaluated in each engineering environment.

2.4. Evaluation Uncertainty

Evaluation of the special fitness measures is accompanied by a degree of uncertainty. The special measures of fitness are quantitative and qualitative variables. Fitness evaluation is completed through experiments or modeling. Interpretation of operational conditions, experimentation, modeling of design physics, and the quantification of design effectiveness are imprecise processes.

$$\Phi_{single} = \Phi(\Phi_{qual-quant}, \Phi_{data}, \Phi_{oper.\,env}, \Phi_{interact}, \Phi_{behave}). \tag{13.6}$$

Qualitative variables are fuzzy due to language imprecision. For example, an automobile seat may be comfortable or uncomfortable. Although comfort or discomfort is very important to the consumer, the two conditions do little to help the engineer change the design. The test engineer could devise a verification experiment using a rating system. This process of quantification is accompanied by a degree of uncertainty due to population statistics and rating uncertainty.

Quantitative measures are not immune from fuzziness and uncertainty. The use of instruments and a single prototype is accompanied by scatter and averaging. The test engineer seeks precise and accurate instruments but still must live with a limited population of prototypes or test cases. Statistics plays an important role in evaluating the measures of fitness.

When evaluating the fitness measures, a prototype is modeled using computer simulations or experiments. Either situation requires adoption of a typical operating environment. In the application environment, the conditions are real. Therefore a degree of uncertainty is introduced by modeling the actual operational environment. This type of uncertainty can only be diminished by performing tests over a range of expected operational variations. This yields statistics and a measure of product robustness.

Computer and experimental models of the candidate design represent a typical configuration. Experimental verification attempts to produce actual physical behavior and task functioning as a model to the application environment. The system may need to interact with other devices, in which case, the verification model approximates the interactions.

In computer models, simulations are produced using idealized laws of physics, which neglect many real effects such as friction, nonlinear interactions, and inhomogeneities. Only the primary physical processes are captured. This is not a problem because, usually, the engineer is concerned only with the primary physical behavior that leads to task functioning. Nonlinear interactions could amplify neglected secondary terms such that the primary physical behavior is altered. This leads to an unpredicted discrepancy or another type of uncertainty.

Although it is usually ignored, a level of uncertainty is present in every engineering problem. A single measure of design or method effectiveness should account for the built-in uncertainties.

2.5. Summary: Features of a Single Measure

There is a need for a single measure of fitness for a variety of reasons. These reasons are formalized in the single measure criteria. Then the product effectiveness and the method by which it is produced could be assessed at the same time. The effectiveness criterion is stated as follows: *The single measure must assess the effectiveness of designed products, design methods, and engineering technical systems.*

The design of products and the design of engineering methods can be considered on equal footing if both natural physical processes and human decision processes are represented. Since the engineering process is a combination of human and natural produced effects, this leads to the mutual existence criterion as *the single measure must permit both natural and human effects to be assessed at the same time.*

Decomposition is a common tool in the design and analysis of devices. The single measure must transcend all scales from microscopic systems up to large macroscopic technical systems. The scale criterion is stated as *the single measure must be quantifiable at any scale and remain applicable over decomposition or combination.*

Different variable types are due to different concerns and the engineering environments that must be represented. The diverse ways of assessing fitness may be transferred to a single measure. The residence criterion is *the single measure must quantify design fitness in the different engineering environments with consistent units.*

The information flow between environments is accompanied by uncertainty. The level of variation of variables must be represented. This will provide an expression of uncertainty that can be tracked and assessed in the evaluation. Therefore, the uncertainty criterion is stated as *the single measure must represent different uncertainty levels of conditions and variables in the engineering method.*

This section outlined the need for a single measure and some of the fundamental requirements the single measure must satisfy. In the next section, entropy is proposed as the single measure that can satisfy these criteria.

3. The Concept of Entropy: Basis and Balance

Entropy is proposed as the single functional measure. This section reviews the foundation of the entropy function and its role in system description. The geometric foundation of equilibrium thermodynamics was pioneered by Josiah Willard Gibbs (Gibbs, 1961). Gibbs saw the system state as a surface with geometric properties such as slope and curvature. Later work led formulations of nonequilibrium that rectify and incorporate accepted dynamical laws.

3.1. Macroscopic Properties

Thermodynamic theory describes the interaction of hidden microscopic modes of action in matter with macroscopically observable states and processes. Classically one begins with a series of postulates (Callen, 1985). The first postulate addresses the macroscopic equilibrium of a system as characterized by energy, chemical species mole numbers, and volume. Extensive variables are divisible during decomposition and additive during combination of subunits k.

$$X \equiv X_{system} = \sum_k X_k, \quad \text{where } X \text{ is } V, N_i, \text{ or } E. \tag{13.7}$$

Equilibrium is a macroscopic property. On the subunit scale, the energy of any one subunit may fluctuate due to configuration changes with its neighbors. Thus the subunit is not at equilibrium. However in an isolated system, the macroscopic scale averages the subunit fluctuations:

$$X = \frac{1}{\Delta t}\int_0^{\Delta t} \sum_k X_k(t)\,dt = \sum_k \frac{1}{\Delta t}\int_0^{\Delta t} X_k(t)\,dt = \sum_k \bar{X}_k. \tag{13.8}$$

The state of equilibrium is observed when the system's scale is sufficiently large to produce constant and uniform characteristics. A macroscopic system that is not isolated is not at equilibrium. Since X is conserved, the X fluctuations about equilibrium must flow to and from neighbors; the subunit has amounts of fluxing sources/sinks of X on j boundaries with its neighbors:

$$X_k = \bar{X}_k + \sum_j X_{kj,in}(\Delta t) - \sum_j X_{kj,out}(\Delta t) = X_k(\Delta t). \qquad (13.9)$$

A macroscopic system that is in communication with other systems cannot be at an equilibrium state. The system's state is then a question of constraints on the system. Consequently, decomposition and combination are important.

3.2. Required Amount of Information and Entropy

Equilibrium is one configuration out of many possible configurations consistent with the boundary constraints. Alteration of the constraints allows the system to access different, nonequilibrium configurations. A certain amount of information is required to characterize each configuration (Shannon and Weaver, 1949). Less information is needed when constraints affect the designed system; constraints are input information to the system configuration problem.

Something is zero at equilibrium. Equilibrium is characterized by a balancing of "forces" or uniform and constant energy, mole numbers, and volume. Equilibrium is also indicative of noncommunication with neighboring systems, i.e., isolation by zero boundary fluxes. More information is needed to describe a system configuration at a new equilibrium state when a constraint (information) is removed; information is transferred. Therefore, given constant constraints, the equilibrium configuration will require the maximum amount of information for description (Raisbeck, 1963). The amount of information needed to characterize the state of a system is a function of the set of configuration variables. For example, a special, thermally open system of constant volume and chemical species

$$I = I(\bar{E}_k, E_{kj,in}, E_{kj,out}, V, N_i). \qquad (13.10)$$

The maximum information needed to describe equilibrium when fluxes are zero occurs when the "slope" of the information hypersurface is zero,

$$\frac{\partial I}{\partial X} = 0 \quad \text{at equilibrium, where } X \in \bar{E}_k,\ V,\ N_i. \qquad (13.11)$$

Equilibrium is represented by the maximum amount of information needed for characterization. This sheds light on the decomposition method. Smaller systems have less possible configurations and require less information to describe. Subunits are formed by internal partitions or constraints.

Alternatively, the traditional thermodynamic view originates from the second postulate: the existence of entropy and its maximization at equilibrium (Callen, 1985). The entropy depends on the same variables as the information measure:

$$S = S(\bar{E}_k, E_{kj,in}, E_{kj,out}, V, N_i). \tag{13.12}$$

Information is a human-descriptive variable and entropy a natural variable. Entropy is postulated to be a maximum at equilibrium.

$$\frac{\partial S}{\partial X} = 0 \quad \text{at equilibrium, where } X \in \bar{E}_k,\ V,\ N_i. \tag{13.13}$$

A third postulate states that the entropy is extensive and therefore is divisible by decomposition and additive by combination.

$$S(E, V, N_i) = \sum_k S_k(E_k, V_k, N_{i,k}). \tag{13.14}$$

Similarly, the information required to describe a system is equal to the sum of the information required to describe its subunits:

$$I(E, V, N_i) = \sum_k I_k(E_k, V_k, N_{i,k}). \tag{13.15}$$

Thus, decomposition does not create or destroy information or entropy, but breaks the system into smaller, more manageable subunits.

3.3. Intensive System Variables—Temperature and Entropy Units

The use of entropy allows the classical definition of slope functions. The hypersurface of the system's state has slopes that are zero at equilibrium and nonzero elsewhere. The entropy relation is partially differentiated to yield definitions of the temperature, pressure and electrochemical potentials such as

$$\frac{1}{T} \equiv \frac{\partial S}{\partial E} \rightarrow T(E, V, N_i). \tag{13.16}$$

The slope functions are known as the equations of state. The slopes are intensive and are not additive when combining a decomposed system. Each subunit has its own set of intensive variables. Consider an isolated system decomposed into subunits. Since the energy, mole numbers, volume, and entropy are extensive, the equilibrium condition for the system yields that the temperature, pressure, and electrochemical potentials be the same between adjacent subunits that have a mutual, fully communicating boundary. Since intensive variables drive fluxes, the fluxes are zero at equilibrium.

The temperature is qualitatively consistent with the human observations of "hot" and "cold." The concept of temperature led historically to temperature instruments and scales. Units of measure were defined to give quantitative measure to the qualitative sensations. Units placed on temperature require that entropy have the units of energy divided by the temperature units. For entropy to maintain its nondimensional status, temperature must have units of energy.

3.4. Nonequilibrium Entropy Balance for Continuous Systems

Real systems are rarely in true equilibrium. Most functions of engineered devices utilize continuous energy, mass, and volume transfer and are not isolated. The continual transfer requires gradients of intensive variables between the subunits or through the system in communication with its surroundings. Behavior, task functioning, and effectiveness of a system are related to the transfer of energy, mass, and volume. The fundamental entropy relation dictates that such transfers create a transfer of entropy dictated by a balance of entropy equation.

The nonequilibrium change of a system's entropy is tracked in terms of entropy flux to and from the system and the entropy produced or destroyed within the system. The entropy-changing processes are produced by the transfers of energy, mole numbers, and volume. The balance of entropy is the variation of the fundamental entropy relation:

$$\delta S = \frac{1}{T}\delta U + \frac{p}{T}\delta V - \frac{\mu_i}{T}\delta N_i. \tag{13.17}$$

In terms of material time derivatives of mass-specific variables,

$$T\rho\frac{Ds}{Dt} = \rho\frac{De}{Dt} + p\rho\frac{Dv}{Dt} - \mu_i\rho\frac{Dn_i}{Dt}, \tag{13.18}$$

where density is mass averaged. Each term on the right-hand side is evaluated by incorporating the dynamical physical laws of conservation of thermal energy (total–mechanical), mass, and chemical species.

Recognizing the common form of conservation equations as a rate of change term being equal to the net flux plus a net source term yields

$$\rho\frac{Ds}{Dt} = -\frac{\partial J_m^S}{\partial x_m} + \sigma, \tag{13.19}$$

where the entropy flux, in terms of heat and chemical species fluxes, is

$$J_m^s = \frac{J_m^E}{T} - \frac{\mu_i}{w_i}\frac{J_{mi}^N}{T}, \tag{13.20}$$

and the entropy production (source), in terms of viscous dissipation, chemical reactions, heat dissipation, gravity work on mass species, and mass species dissipation is the scaler (de Groot and Mazur, 1984):

$$\sigma = \frac{1}{T}J_{mi}^V\frac{\partial u_i}{\partial x_m} - \frac{1}{T}A_iJ_i^R - \frac{1}{T^2}J_m^E\frac{\partial T}{\partial x_m} + \frac{J_{mi}^N}{w_i}\left[F_{mi} - \frac{\partial}{\partial x_m}\frac{\mu_i}{T}\right]. \tag{13.21}$$

The terms have a common form of a spatial gradient multiplied by a flux. Thus, the entropy production is related to mass, chemical species, thermal and mechanical energy behaviors, and functions within the system.

3.5. Phenomenological Laws

Physical effects, produced by gradients within the system or subunits, are responsible for the function of designs or engineering methods. These gradients are supported by boundary conditions on the system or decomposed interfaces between the subunits. The entropy production and entropy flux terms are related to these boundary induced gradients.

Phenomenological laws are the relationships between fluxes and gradients. A special example is Fourier's law of heat conduction for a homogeneous isotropic system where heat flux is the thermal conductivity multiplied by the temperature gradient. Other examples include Fick's law, Hooke's law, Ohm's law, and Stoke's law. These laws were developed for specific physical systems in the absence of other physical effects. The laws were also developed in terms of energy-type measurements.

Alternatively, nonequilibrium thermodynamics utilizes general flux-gradient relationships stated in terms of entropy. This results in kinetic coefficients as an alternative to the traditional material properties of thermal conductivity, binary diffusion coefficient, or resistance (de Groot and Mazur, 1984; Haase, 1969). For example, the general phenomenological relation for heat flux is expressed in terms of temperature, electrochemical potential, and velocity gradients as

$$J_m^E = -L_{mj}^{ET}\frac{1}{T^2}\frac{\partial T}{\partial x_j} - L_{mik}^{E\mu}\frac{1}{T}\frac{\partial \mu_i}{\partial x_k} - L_{mpq}^{Eu}\frac{1}{T}\frac{\partial u_p}{\partial x_q}. \tag{13.22}$$

However, some of the kinetic coefficients can be stated in terms of accepted material properties and temperature. For example, the kinetic coefficient for temperature driven heat flux is related to the thermal conductivity tensor as

$$L_{mk}^{ET} = T^2 k_{mk}. \tag{13.23}$$

Anisotropic and cross-effect kinetic coefficients are not readily available but may be symmetric due to Onsager's reciprocal relations. For example, the Dufour effect is the flux of heat driven by species concentration gradients and the Soret effect is the mass flux of species driven by temperature gradients. Material properties for the Soret and Dufour effects are not as widely available as scaler thermal conductivities, viscosities, and mass diffusion coefficients.

The fluxes occur on the boundaries and within the system or subunits. Since scale is selected by decomposition, there is an opportunity for approximation of the transfer laws. The different effects of system functioning can be modeled in terms of phenomenological flux equations with gradients and kinetic coefficients. The interior fluxes of a system can be approximated using the fluxes on the boundaries. The boundary fluxes are relevant in task functioning while interior fluxes contribute to entropy production.

The entropy function can be calculated and tracked in terms of a variety of underlying physical processes. The equilibrium fundamental relation links entropy to mechanical (volume), electrochemical (mole numbers), and energy processes. The processes between decomposed units are driven by differences of temperature, pressure, and electrochemical potentials. At the heart of all processes in a design's operation or in the engineering method are the fundamental mass and thermal, mechanical, and electrochemical processes. Therefore, entropy is proposed as the single measure for effectiveness of designs and engineering methods. It will be shown in successive sections that the entropy quantity has the ability to represent many aspects of the design process and technical system development. Some of the criteria outlined in Section 2 will be addressed in a general fashion.

4. Application of Entropy and Scale Issues

The properties of designed systems or subunits change when the boundaries are open or closed to fluxes. The effects of internal and external partitions that define the scale of systems and subunits are important. This section addresses satisfaction of the scale criterion.

4.1. *Special Decomposed Subunits*

The constraints on the system's boundaries produce the gradients that support physical behavior and task functioning. Chemically closed systems refer to systems with no chemical species fluxes. Thermally closed systems do not have thermal energy (heat) flux and mechanically closed systems do not have work fluxes. These special situations are of interest to designers of special components.

Each type of closed system yields a special form of the entropy balance equation and method of calculating the subunit entropy. For example, the rate of entropy change in the fully closed system is equal only to the entropy production inside the system.

$$\rho \frac{Ds}{Dt} = \sigma. \tag{13.24}$$

Using phenomenological flux-force relations, the entropy production is a positive definite quantity. For example, for a homogeneous, isotropic, incompressible material with linear phenomenological relations and no cross terms, the entropy production is

$$\sigma = \frac{1}{T^4} L^{ET} \left(\frac{\partial T}{\partial x_j}\right)^2 + \frac{1}{T^2} L^{N\mu_i} \left(\frac{\partial \mu_i}{\partial x_m}\right)^2 + \frac{1}{T} L^{Vu} \left(\frac{\partial u_j}{\partial x_m} + \frac{\partial u_m}{\partial x_j}\right)^2. \tag{13.25}$$

This can be used for many systems as material requirements dictate. All electrochemical, thermal, and mechanical systems have positive definite entropy production inside their boundaries.

In open systems, the increase or decrease in entropy rests with the net entropy fluxes compared to the positive definite entropy production. The only way to decrease the entropy and required information of a system is to have openings to fluxes. Fluxes are the realm of nonequilibrium thermodynamics and are dictated by the special functions of the decomposed system and its subunits.

The entropy balance applies to macroscopic scales of the systems or subunits. In macroscopic cases, the continuum assumption leads to a wealth of special discipline knowledge for the determination of gradients, kinetic coefficients, and boundary fluxes (Bejan, 1982; Haase, 1969; and LaFleur, 1990a). The decomposed scale also leads to approximations of interior fluxes for the calculation of entropy production. The macroscopic entropy is calculated in terms of near-equilibrium physical behaviors and functions. The alternative is to calculate the system's entropy in terms of "in the limit" decomposition to a large number of microscopic subunits followed by combination. This idea leads to quantum mechanics and thermostatistics.

Extensive properties, coupled with conservation laws, indicate that system decomposition or subunit combination do not create or destroy energy, mole numbers, volume, entropy, or information. The fundamental relation of entropy links the observable macroscopic configuration variables and the hidden, microscopic configurations within all matter. The scale independence of the first law of thermodynamics yields a balance that is not affected by decomposition. The fundamental entropy relation is independent of scale in macroscopic systems.

The jump to microscopic description is not kind to the fundamental entropy relation. As the scale of a system approaches the macroscopic to microscopic boundary, the number of subunits that affect the system increases dramatically. The number of configurations is related to the number of atoms using combinatorics. A large number of configurations are possible and the required information is large. Microscopic systems design is the realm of materials scientists and engineers.

4.2. Combining Finite State Subunits—Thermostatistics

In the limit, the microscopic subunits have a large number of configurations and the fluctuations allow the system to visit different configurations frequently. This idea may also be applicable to design or combinations of macroscopic subunits with finite numbers of states.

The finite configurations of the subunits are described by quantum theory where each configuration is a certain energy state. Combination of the subunits creates a combinatorics problem where the number of combined system configurations is factorially related to the number of subunit configurations. Fluctuations from one system configuration to another is assumed to be random. Therefore each configuration is equally probable. Unequal probabilities are discussed later.

For example, consider a system decomposed into M similar subunits of two states each (of energy 0 and energy E_i). The number of subunits at the E_i state is the total energy divided by the subunit energy or

$$M_j = \frac{E}{E_j} = M_j(E, E_j). \tag{13.26}$$

The number of subunits at the 0 state is the balance or

$$M_0 = M - M_j. \tag{13.27}$$

The number of configurations is equal to the number of ways to distribute M_j energy packets among the M subunit compartments. Combinatorics yields the number of possible configurations as

$$\Omega = \frac{M!}{M_j!\,M_0!} = \frac{M!}{M_j!(M - M_j)!} = \Omega(M_j, M) = \Omega(E, E_j, M). \tag{13.28}$$

A similar approach can be used for dividing a system into discrete subunits with individual configurations. The key is to model each subunit with a limited number of finite energy states and probabilities for the energy states.

The number of possible configurations is related to the information required for description and the system entropy. The number of configurations is multiplicative between the subunits while information and entropy are additive among the subunits. The function that connects the entropy to the number of configurations is the natural log function. Thus the fundamental relation for entropy stated in terms of configurations is

$$S(E, V, N_i, E_j, V_j, N_{ij}, M) = K \ln \Omega(E, V, N_i, E_j, V_j, N_{ij}, M), \tag{13.29}$$

where K may be chosen as Boltzmann's constant and fix the scale of S to match the Kelvin temperature scale. From this entropy fundamental relation, versions of temperature, pressure, and electrochemical coefficient can be found (the quantities that drive fluxes and produce the design's function). Equation (13.29) can be used as an explicit basis to track entropy changes in terms of finite configuration changes.

Natural fluctuations between the different configurations are described by thermostatistics. Consequently, the state of the system is not known but is described by probabilities and statistical moments. Thermostatistics provides a method of calculating entropy for finite state systems when each state has a level of probability. Similar mechanisms occur during the design process when different configurations are tried and probabilities of design outcomes can be formulated. The probabilities depend on the number of possible design solutions and the amount of extensive quantities associated with the choices.

4.3. Material Partitions and System Environment

The change of the system's configuration leads to observable, macroscopic changes in energy, mass species, and volume. Often this is seen as matter segregation or the formation of material partitions. A system with discernable material partitions is not at true equilibrium. The condition for zero fluxes requires uniform character of the matter. This can not be accommodated in systems with open material partitions. Organization of systems into uniform property subunits creates a singularity of entropy at the subunit interfaces. The material partitions can be defined as configured subunits when decomposing.

At nonequilibrium, the conditions external to the system have the opportunity to influence the internal entropy and information required for description. The internal information is associated with the concept of organization or order/disorder by Shannon and Weaver (1949). It is the action of external effects that creates organization such that less information is required for characterization. The external conditions and constraints are defined as the system's environment (LaFleur, 1991). The exterior of a subunit is the subunit's environment. In most cases, subunits are the environment of other systems or subunits. Therefore, the environment itself is a system that has its own configuration variables and entropy and information.

Defining a large closed system (supersystem) as a system plus its environment yields the traditional view that entropy, information, and disorder are maximized. Since entropy production is positive definite, the supersystem entropy will monotonically increase with time. Therefore entropy is maximized in the supersystem. This does not mean that the system's entropy is maximized; the system's entropy may be minimized at the expense of its environment, i.e., environmental impact.

5. Representing Different Variable Types

One required feature of a single measure is the representation of different variable types. Entropy is an appropriate measure if it is applicable to multifunctionals, qualitative or quantitative variables, uncertainty, and continuous/discrete systems.

5.1. Multifunction Design

High functioning systems, such as products and technical systems can be decomposed into subunits with lower, special functioning. Subunit measures of fitness must be related to performances such as task functioning, safety, environmental impact, and required technical system resources. Functioning that is reduced by decomposition is stated in terms of fundamental fluxes.

The technical system can be examined in a similar way, i.e., decomposition to special subunits and functioning in terms of thermal and mechanical energy and chemical species fluxes. Technical system measures of fitness such as economics, personnel time, scheduling, quality, and market share can all be fundamentally related to the physics in the entropy balance equation.

For example, considering personnel as a subunit in the technical system, the cost of workers is related to the amount of work done. Work done transforms inanimate products as changes in thermal energy, mechanical energy, and electrochemical species. Apparently diverse measures can be stated in terms of the entropy balance when measured by extensive quantity changes.

5.2. Quantification of Qualities: Fuzziness and Probabilities

In engineering design, qualitative variables are not accountable unless converted to quantitative measures. The conversion is not exact and may allow fuzziness using statistics of population data. The conversion yields averages and statistical moments, a quantified statistical representation of the qualitative variables. Statistics plays a role in calculating entropy and required information.

For example, if a system and its environment are subunits of a closed supersystem of fixed volume and mole numbers, the super-system energy is

$$E_{super} = E + E_{env} \quad \text{and} \quad E = \sum_{m=1}^{\Omega_E} P_{Em} E_m, \tag{13.30}$$

where the configuration probability equals the number of available environment configurations divided by the number of possible supersystem configurations

$$P_{Em} = \frac{\Omega_{E,env}(E_{env,m})}{\Omega_{E,super}(E_{super})} = \frac{\Omega_{E,env}(E_{super} - E_m)}{\Omega_{E,super}(E_{super})} = P_{Em}(E_m, E_{super}), \tag{13.31}$$

where $\Omega(E)$ is the number of configurations of energy E (Callen, 1985). For example, if many configurations of the environment are possible with a particular system configuration, then the system configuration is probable.

Information theory (Shannon and Weaver, 1949) yields that the entropy of the qualitative configuration is related to the probability of the configuration

$$S_{Em}(E_m) = -K \ln P_{Em}. \tag{13.32}$$

The total number of configurations is a multiple of the number of configurations, Ω_j, over each degree of freedom, D:

$$\Omega = \prod_{j=1}^{D} \Omega_j = \Omega_1 \Omega_2 \ldots . \tag{13.33}$$

For every degree of freedom, configuration frequency of occurrence based on sorting in qualitative bins, is related to outcome probability as

$$P_{jm} \equiv \frac{\omega_{jm}}{\omega_j}, \quad \text{where } \omega_j = \sum_{m=1}^{\Omega_j} \omega_{jm}. \tag{13.34}$$

For example, a questionnaire on learning environment was used to poll an undergraduate and graduate class. The questionnaire had numerous questions from four categories unknown to the student. The results for the two degrees of freedom are shown in Table 13.2 below. The frequency data can be reduced to probabilities using equation (13.34) for each of the two degrees of freedom.

The entropy in each degree of freedom (LaFleur, 1990a) is calculated by adding the entropy of the configuration, weighted by the probability of occurrence or

$$S = \sum_{j=1}^{D} S_j = \sum_{j=1}^{D} \sum_{m=1}^{\Omega_j} P_{jm} S_{jm} = -K \sum_{j=1}^{D} \sum_{m=1}^{\Omega_j} P_{jm} \ln P_{jm}. \tag{13.35}$$

Therefore, the entropy can represent fuzzy variables, population-frequency data and uncertainty in the engineering method. Based on the example in Table 13.2, the entropy of the undergraduate and graduate classes were 1.328 K and 1.276 K respectively. Adding these two degrees of freedom yields a total entropy of 2.604 K. From the entropy measure, it is clear that the graduate class had less entropy; the experience and knowledge of graduates are constraints and graduate classes tend to be more specialized. In a similar way the entropy can measure the advancement of knowledge through design science research.

The probabilistic calculation of entropy is especially applicable to the conversion of qualitative variables to accountable quantitative variables. In the example given above, each qualitative question could form a finite state subunit and could be analyzed in detail to give an entropy for each outcome. Then the total entropy would be calculated in terms of the middle of equation (13.35), the sum of the product between an outcome's entropy and the probability of that outcome.

TABLE 13.2. Learning environment questionnaire responses.

Class	ω_{j1}	ω_{j2}	ω_{j3}	ω_{j4}	ω_j
Undergraduate $j = 1$	34	91	53	62	240
Graduate $j = 2$	8	35	30	37	110

5.3. Continuous and Discrete Systems

Entropy can be used to assess the fitness of both continuous and discrete systems. Continuous systems contain a continuum of matter and allow fields to form. Fields are mathematically described by dependent variables such as temperature, pressure, electrochemical potential, concentration, etc., and independent variables (space and time). System characteristics are described by field distributions with partial differential equations for balance equations.

Discontinuous systems have partitions usually formed along interfaces between different matter. The discontinuous system can be viewed as a sum of continuous subunits with interfaces between them. Continuous subunits can be treated in terms of fields. The interface between the discrete subunits is a surface that has its own balance equation.

The entropy is discontinuous between continuous subunits and the interface is a subunit that contains the entropy discontinuity. Boundary surfaces are of lower spatial dimension than a system volume; a lower number of configurations are possible. Variationally, a quantity is constant on the surface. Geometrically, at any instant, the surface is a special function of three spatial coordinates. This can be expressed as a zero variation of the function between the coordinates.

A system and environment may be treated as both continuous and discontinuous. The field equations for continuous systems or subunits are used in the entropy balance to find entropy fluxes and production. The interior entropy fluxes can be approximated in terms of boundary fluxes. For example, LaFleur (1991) gives the solution of a pipe insulation problem in terms of thermal and fluid entropy production. The entropy associated with the configuration of discontinuous units into a system is measured using information-thermostatistical theory discussed above. Configuration probability is stated in terms of the frequency of occurrence or the number of configurations along the degrees of freedom of energy, volume, and electrochemical mole numbers.

6. Representing Different Levels of Variation

In engineering methods and environments, there are a variety of variables such as constraints, conditions, specifications, physical behaviors, material properties, and geometries. A single measure is required to represent quantities with different levels of variability. The entropy measure tracks engineering method progress and represents the different variables using a level of variation hierarchy (LaFleur, 1989).

6.1. Variable Types

Some features of engineering problems are constant. A distinction can be made between different types of constants. One type is a universal constant.

For example, numbers, conversion constants, pi, Boltzmann's constant, etc. are accepted as universal constants. Another constant type is derived from the material tables referenced in engineering. Tabulated material properties are accepted as constant although the materials may be unknowns in the design problem. Another type of constant is the constraints on the design problem or method. Constraints define the design problem and limit the system configurations. These are adjustable but do not vary during the solution process of one problem. Another type of constant is the physical boundary conditions on the system. These limit the physical behavior and produce the gradients needed for functioning. Condition adjustment is usually performed between solutions in search of control or application-matching operating conditions.

The problem contains unknowns that characterize the effectiveness of the solution. Behavior variables represent the internal configuration of the system including temperature, pressure, energy, chemical species concentration, velocity, and strain. Spatial coordinates and time locate field distributions and internal and boundary fluxes. Task performance variables, such as safety and efficiency, characterize the system or subunit fitness. Unknown variables may have known upper and lower bounds due to conditions and constraints. Qualitative measures, imprecise specifications, or nonlinear behavior require the use of probabilistic variables. Statistical measures are derived from histograms that indicate probabilities under the action of random effects or imprecision.

6.2. The Variable Hierarchy

The constant, characterization, and statistical variable types have different levels of variation. Entropy must be influenced by the problem variables to reflect the overall effectiveness of the technical system or engineering method. The problem's variables can be organized into a hierarchy based on the level of variation. This sorts and organizes knowns and unknowns to be expressed in the entropy measure. Lower entropy means more order or more constraint on possible configurations. Knowledge lowers the variability of the solution set.

The problem variables' levels of variation lead to a hierarchy of variation. Variables can be arbitrarily sorted into compartments of different variation levels. For example, a nine-level hierarchy is defined in Table 13.3 below (LaFleur, 1989).

Incomprehensibles are unobserved variables and are usually assumed to be ineffective on the system operation. Level 3 and lower variables are known variables and are available after problem set-up. Research is needed if the problem statement does not explicitly state the knowns. Level 4 represents the independent coordinates that arise in the governing physical equations and are used to track field, behavior, and performance variables. Unknowns are identified by levels 5 and higher and are evaluated in the

TABLE 13.3. Example variable hierarchy of nine levels.

$X(vl) \equiv$ variable X of	Variational level vl
vl	Description
0	Universal constants
1	Material properties
2	Geometric constraints
3	Physical conditions
4	Independent variables
5	Bounded dependent variables
6	Unbounded dependent variables
7	Random variables
8	Incomprehensible variables

engineering method. Behavior variables are governed by physical laws. Performance measures are defined in terms of behavior variables. Design or configuration are determined by a decision-making process; input information provides closure.

For example, the design of a heat exchanger can be treated using the level of variation hierarchy to yield:

$vl = 0$	pi
$vl = 1$	for the two fluids: thermal conductivity, specific heat, viscosity
$vl = 2$	inlet and exit fitting sizes, maximum length and shell, diameter
$vl = 3$	for the two fluids: inlet flow temperature and flow rate
$vl = 4$	radius, axial coordinate, time
$vl = 5$	for the two fluids: temperature distribution, flow velocity distribution, pressure drop, net heat exchange and materials of the shell, tubes and fins, tube diameter, number of tubes, tube pattern, tube wall thickness, fin spacing, fin thickness, shell diameter, shell thickness, head thickness

Level 5 variables may be categorized in terms of behavior variables (governed by physical principles), performance variables (governed by function definitions), and design variables (degrees of freedom). Using level 5 assumes that the bounds of variables are known (bounds are level 2 or 3 variables).

6.3. Implicit Variations

The entropy, as the single measure of system or method efficacy, must depend on the variables sorted in the variation hierarchy. Thermodynamics requires that a fundamental relation be derived from which all properties and design sensitivities can be found such as the implicit formula of

$$S = S(X(0), X(1), X(2), X(3), X(4), X(5), X(6), X(7), X(8)). \quad (13.36)$$

Nonequilibrium is stated in terms of an expansion using sensitivities as

$$\delta S = \frac{\partial S}{\partial X(vl)} \delta X(vl), \quad \text{where } vl = 0, 1, 2, 3, 4, 5, 6, 7, 8. \tag{13.37}$$

The variation of entropy is due to processes in the system or method. Level 0 variables do not vary, variation of independent variables are zero, and incomprehensibles are assumed to be ineffective. The nonequilibrium entropy variation can be split into natural and human-controlled processes (LaFleur, 1990a). Natural processes governed by conservation laws are the variation of level 5, 6, and 7 variables. Human-controlled or virtual processes are the variation of level 1, 2, and 3 variables

$$\delta S = (\delta S)_n + (\delta S)_v, \tag{13.38}$$

where

$$(\delta S)_n = \delta S(5,6,7) \quad \text{and} \quad (\delta S)_v = \delta S(1,2,3). \tag{13.39}$$

Spatial or time dependence is not tracked in the variational process.

The unknowns sorted as levels 5, 6, and 7 must be solved to characterize the system or engineering method in terms of the entropy measure. These variables are solved in the engineering method using analysis, design decisions, and statistics. Unknowns depend on the knowns, independent variables, and each other. Levels 7, 6, and 5 variations can be solved successively or simultaneously in terms of lower level variables as

$$\delta X_k(5, 6, \text{or } 7) = f_k(X(4), X(3), X(2), X(1), X(0)). \tag{13.40}$$

The processes of solution can occur if the number of unknowns is matched by an equal number of governing equations or input decisions, i.e., an informational balance. Input decisions derived from engineering environments are of level 3 or lower, i.e., they are constant but adjustable. The input of information constrains the engineering problem degrees of freedom. The entropy of the problem, as a function of the levels of variations, should decrease with solution knowledge. This tracks the advancement of the design process or method improvement.

If the system is characterized by integral variables (space and time averages), then the unknowns are reduced to level 3 variability. The solution process reduces the level of variability. Therefore, the determined system or engineering method has entropy of level 3 variation. System or method improvement is attained through the engineer-controlled virtual process of level 1, 2, and 3 variations. Therefore, the problem solution effectiveness of the engineering system is measured by how the levels of variation of a problem decrease. The entropy tracks the problem's highest level of variation.

Nonlinear systems are not well behaved and may produce instability and multiple solutions. For example, LaFleur (1991) gives the design of pipe insulation using four methods. The genetic algorithm results in a nonlinear design evolution that has a "strange attractor" as a fuzzy region of solutions. In this case, the solution process does not fully reduce the problem level of variation completely and leaves degrees of freedom. The remaining degrees of freedom require extra information inputs leading to one system design or one engineering method. In the case where nonlinearity produces multiple solutions, one must be selected. Human control is a virtual process and constitutes artificial selection (LaFleur, 1990b). The natural selection is a real process produced through fluctuations that act on unstable solutions that separate regions of stable solutions.

Using the variable hierarchy, the entropy can be stated in terms of variables with different levels of variations, from constants up to random variables. The engineering method's effectiveness is accountable in terms of how unique (linear) problem solutions are found or how nonunique (nonlinear) solutions are selected naturally or artificially.

7. Summary

Entropy was hypothesized to meet the single measure criteria as a common basis for measuring the effectiveness of designed products, engineering methods, and technical systems. The entropy function thermodynamics were reviewed and entropy was found to be widely applicable over macroscopic to microscopic scales, equilibrium or nonequilibrium processes, and continuous or discrete systems. Entropy was found to apply to the decomposition of a system into smaller subunits and to the combination of components into a system. The general system was treated as a sum of subunits that themselves may be systems. The treatment of a system's environment as a system identifies environmental entropy.

Multifunctional system performance can be measured using the entropy calculated from the special actions of mass species, thermal energy, and mechanical energy on the systems's boundary. Internal fluxes that cause entropy production could be modeled in terms of boundary fluxes for specific engineering domains. Qualitative variables can be converted to quantitative measures of entropy by tracking the number of degrees of freedom, number of configurations, and the probabilities of each configuration occurring.

The entropy function is related to the engineering problem's level of variation. Important variables are sorted into a variation hierarchy where implied relationships govern the overall problem level of variation. Both natural and human controlled processes in engineering design are tractable through the natural and virtual processes of the entropy function. The increase of understanding is the statement of variables and relationships between them. Knowledge reduces the problem's level of variation. Therefore, research on applied design, new methodologies, and testing of hypotheses could be tracked using the single measure of entropy. Design science advances are reflected by lowered entropy.

References

Bejan, A. (1982). *Entropy Generation through Heat and Fluid Flow*. New York: Wiley.

Callen, H. B. (1985). *Thermodynamics and an Introduction to Thermostatistics*. New York: Wiley.

de Groot, S. R., and Mazur, P. (1984). *Non-Equilibrium Thermodynamics*. New York: Dover.

Gibbs, J. W. (republished 1961). *The Scientific Papers of J. Willard Gibbs, Ph.D., LL.D., Volume I*, New York: Dover.

Haase, R. (1969). *Thermodynamics of Irreversible Processes*. Addison Wesley.

LaFleur, R. S. (1989). A hierarchy for organizing multivariable design and analysis problems. *Proceedings of the Annual ASEE Meeting of the St. Lawrence Section*, Session 14A3, Engineering Education Methods, pp. 1–10.

LaFleur, R. S. (1990a). *Lecture Notes of Advanced Thermodynamics*. Clarkson University.

LaFleur, R. S. (1990b). The role of evolution in design. *ASEE DEED Bulletin*, *14*(3), Spring 1990.

LaFleur, R. S. (1991). Evolutionary design theory using dynamic variation and thermodynamic selection. *Research in Engineering Design*, *3*, 39–55.

LaFleur, R. S. (1992). Principle engineering design questions. *Research in Engineering Design*, *4*, 89–100.

Raisbeck, G. (1963). *Information Theory*, Cambridge, MA: MIT Press.

Shannon, C. E., and Weaver, W. (1949). *A Mathematical Theory of Communication*, Urbana, IL: University of Illinois Press.

14
Design Education

KENNETH J. WALDRON AND MANJULA B. WALDRON

Abstract. The instruction of design in engineering curricula has long been controversial. New influences such as changes in industry practices, changes in the approach of the profession, as reflected in accreditation criteria, and changes in the legal and regulatory environment within which we must operate, mandate some rethinking of design instruction. Some traditional approaches continue to be useful, but new technical materials and methodologies must be incorporated into already crowded curricula. Some suggestions are provided in this chapter.

1. Introduction

The essential difference between an engineer and a scientist is that the engineer creates new artifacts and technologies, whereas the scientist studies the world as it currently exists. The parts of engineering that relate directly to this creative or synthetic activity are design and manufacture (Waldron, 1992). In some engineering fields we are not accustomed to think in terms of "manufacture," but there are other, equivalent words such as construction, which is really the same concept applied to artifacts of larger scale. In other cases the engineer's product may be a process chart, or a set of software. Nevertheless, some sort of artifact is produced, and there is a sophisticated planning process involved in the production of that artifact.

Design is planning for manufacture. An excellent command of manufacturing processes will avail nothing without good design (Hoover and Jones, 1991). This point has tended to be lost in the current concern over improving manufacturing industry.

Is there enough emphasis on design in engineering education? This is an old question that we keep revisiting. Nevertheless it is an appropriate question at this time for several reasons.

One reason is the new Accreditation Board for Engineering and Technology (ABET) criteria. These specify one year of mathematics and basic sciences, one-half year of humanities, and one- and one-half years of engineering topics. There is no longer any set number of hours for design as opposed to engineering science topics. There is a requirement for a meaningful design experience toward the end of the student's educational program. There is also a requirement for the integration of open-ended problems throughout the curriculum.

Another reason is that industry has now absorbed the concept of concurrent engineering into their culture, and they are demanding that students have experience in working in teams and, in particular, in cross-disciplinary teams.

Yet another reason is the current trend among university administrations to insist that curricula extend over no more than 120 semester credit hours. Since almost all engineering curricula currently require substantially more credits than this it is predictable that there will be pressure to cut material. Time in the curriculum devoted to design activities will be vulnerable to reduction under these pressures.

The new ABET criteria (Engineering, 1994) throw the responsibility for ensuring that our curricula have appropriate design content onto us, the engineering academic community, and onto the professional community of which we are part. It is no longer possible to hide behind the ABET requirements. Equally, it is no longer possible to use them as a reason for a lack of innovation. It is up to us whether we treat this as a disaster, or as an opportunity.

As far as providing experiences in working in multidisciplinary teams is concerned, we are at the same point that industry was at 5 or 10 years ago: we are not set up for that kind of activity. We will not get much sympathy from our colleagues in industry, since, in many cases, companies had to go through very painful restructuring exercises to get to a point that they could fully utilize the interdisciplinary teaming ideas of concurrent engineering. We need to get to work and remove whatever barriers we may have to this mode of operation.

Students learn to design by doing it (Koen, 1994). Meaningful project experiences must form the core of any educational program in engineering design. This is because it is in part a creative activity akin to that of a creative artist. The medium is different and is, in fact, very much more complex, but the nature of the process is very similar.

This central fact has made design instruction the bane of generations of academic administrators. Design resists being taught "efficiently" by packing students into lecture classes that can be taught with minimal faculty effort. Design instruction demands individual interaction between the student and the instructor. Consequently, design instruction is manpower intensive and does not fit when universities make policies about course section sizes and instructor workload.

Another consequence is that design resists attempts to turn it into a "science" (Dixon, 1988). It is, fundamentally, a synthetic activity. Sciences are fundamentally investigative activities. Once again, design does not fit the pattern. Consequently, faculty who make design their primary focus face problems with the promotion and tenure process when that process is based on a narrow, science-based definition of scholarship. This does not have to be. Most universities also use different definitions of scholarship as criteria against which synthetic activities are judged. There is no reason these definitions cannot be adapted to evaluation of design activities. However, the use of design activities in promotion and tenure is perceived to be a problem by engineering faculty, which negatively impacts their willingness to commit effort to design activities and design instruction.

For all these reasons, there is a continuing tension in engineering curricula between the need to give instruction in design and the tendency to reduce, or even eliminate that instruction since it doesn't fit the primary instructional paradigm of the university for science-based subjects.

2. Project Work

As was pointed out above, students learn design by doing it. The provision of meaningful project experiences is therefore crucial in any design curriculum. Of course, good feedback on the project work is essential to the learning experience. The most effective feedback comes from having the artifact designed actually manufactured as a prototype. Many manufacturing problems will not be identified until a part is put in the hands of a technician for manufacture. It is important for students to learn, early, the importance of communicating with those who will be making the artifact. Functional shortcomings may not be easily envisaged when the design only exists on paper, but will be immediately apparent when it is prototyped. Finally, the experience becomes more vivid to the student and is consequently better remembered.

It is easy to shy away from carrying projects through to manufacture of a prototype. Large and expensive artifacts, such as civil engineering structures, may be economically impossible to prototype. Even smaller items may be expensive to prototype, particularly since most prototypes must be predominantly handmade. Nevertheless, carrying projects through to the prototype stage gives the students the best possible design experience.

Paper projects, in which only drawings and specifications are produced, may be the only viable option given the fiscal and temporal constraints of the program. However, to be effective, a paper design project must be exhaustively critiqued by one, or preferably several, experienced designers. Doing this in the format of a design review in which the students must present their work for criticism by the rest of the class, the instructors, and experienced invited experts, is effective in focusing the students' attention by requiring them to defend their work. The feedback obtained may not be as effective as having a prototype manufactured and tested, but it will certainly pick up major shortcomings. It should be emphasized that a paper design project without adequate feedback to the students is of little instructional value.

All of this requires instructors with significant design experience. Unfortunately, for reasons discussed in the introductory section, many engineering faculty are not equipped to be effective design project instructors, and many are not willing to put in the necessary effort to work with the students effectively. Some schools solve this problem by hiring experienced designers who have retired from industry, or who work on a part-time basis as instructors for design projects.

Project work is also essential to provide students with project management and team-working experience. Ideally, each project should be conducted by a team with members from a variety of relevant academic backgrounds. This might include students from outside the engineering college, such as students from business, communication, or industrial design, as well as students from several engineering disciplines if the project is broad enough. After many years of being discouraged from collaborative work, students often have difficulty adapting to the team situation.

Use of project management tools, such as a statement of work, a Gantt chart, or other time scheduling tools, and, of course, a budget should be required (Ulrich and Eppinger, 1995). Students should learn about professional conduct and division of work in team-working situations.

Students need to be aware that a design can be viewed as a hierarchy of decisions. When making a design decision, the designer sets a dimension, or selects a material or component. There is a second set of decisions, which are decisions which direct the design process. These are decisions about what to do next, or about how far to pursue analysis or experiment. They are management decisions.

Students also need to understand that an important feature of design is that the available time, and resources, become constraints on the design process. Often, complete analysis cannot be performed within these constraints. The designer must then make design decisions with only partially quantitative information. There is an important set of decisions in any design project which trade-off the quality of information that can be obtained by further analysis against the time and resources required to obtain that information. A finite element analysis of a part requires considerable expense in personnel time to generate the model, and computer time to analyze it and present the results. It may also take several days to yield results. If computations using simple strength of materials models indicate that the factor of safety is large there is no point wasting those resources on the finite element analysis. Even if the strength of the part appears to be marginal, if time and money is short, cruder but less expensive alternatives to further analysis will be used. For example, the cross-section might be simply increased to a size judged to be safe, or a higher strength material might be selected.

A consequence is that designers must often make decisions on the basis of incomplete information or, more accurately, on the basis of incomplete information from analysis combined with the knowledge base generated by their integrated experience. Such decisions can be very difficult, particularly when the success or failure of the project hangs on them. Nevertheless, they must be made. Making these judgment decisions is a new and important experience for students who are accustomed to "black-and-white" situations in which there is only one right answer, and analysis is pursued until that answer is manifest.

An increasing number of schools are using projects from industry (Bailey, 1995), and are expecting the sponsoring company to contribute to the costs of prototyping the design and operating the course (Beach, 1993; Roeder, 1994). This strategy eliminates the objection to manufacturing prototypes on grounds of cost. It can also make proper levels of course staffing more palatable to administrators. Most important, it ensures that the problems used are real problems, and are perceived as such by the students. This practice also provides an important interaction with the industry sponsor, and provides a real customer to which the group must present their work.

3. A Design Methodology

Although the primary means of learning design is by practice, students and experienced designers need structure to enable them to find their way through the process. This is the function of design methodology. There has been a lot of attention paid to design methodology in recent years. Such concepts as Quality Function Deployment (QFD) (Clausing, 1994) and Concurrent Engineering are design methodologies (Ullman 1992; Wesner et al., 1994). Unfortunately, the presentations in many traditional textbooks are grossly oversimplified and misleading. This is, in fact, an extensive and complex subject.

Although the prescriptive presentation found in most older books (Pahl & Beitz, 1984) is incomplete and therefore misleading, it is a useful starting point for presentation of design methodology. Figure 14.1 has been used by the first author for some years when introducing the subject. This figure shows a generic flow-chart of the design process. The portion of the chart to the left of the heavy dotted line deals with the activities of the human design team. The items to the right of that line are computer-based tools that can be used in the appropriate portions of the process.

The three blocks in the top left-hand corner deal with the identification of the problem by "the customer," and the interactions between the design team and the customer, or sponsor. The process begins with a perceived need. It is useful for the students to be able to negotiate the specifications upon which the design is to be based with a real sponsor, as occurs in professional design work. The original specification from the sponsor will often contain data in forms that are not directly useful for design purposes. Students need experience in identifying the data they need to drive the design

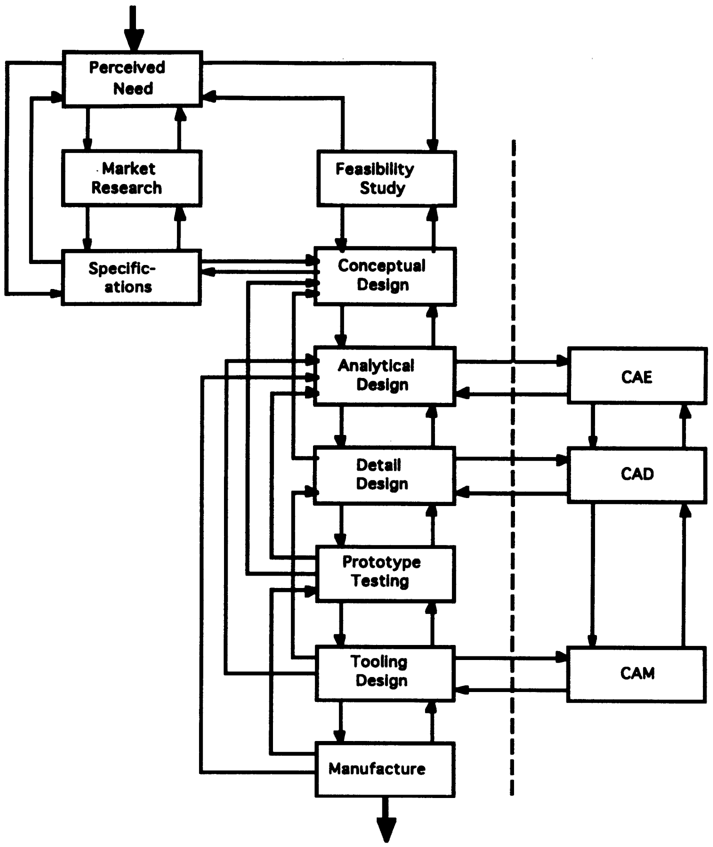


FIGURE 14.1. The design process. The central stem represents the major activities of the design team. The three blocks in the top left-hand corner represent the interactions with the sponsor of the project that result in the specifications which drive the design process. The blocks to the right of the heavy dotted line represent computer-based design tools which might be used in the design process.

process, and extracting that information from the sponsor's requests. QFD techniques are useful for this transformation (Fleischmann, 1994) and computer tools such as ITI's Quality Capture^{TM} are now available for students to create their house of quality (Hale, 1995).

The central stem of Figure 14.1 indicates activities which are primarily carried out by the design team. In particular, the three blocks indicated by the heavier lines might be regarded as forming the "core" of the design activity.

The conceptual design block represents the part of the process that encompasses the formulation of the concept upon which the design will be based and the approximate determination of major dimensions, and selection of major components. The analytical design block represents the phase in which the design concept has been identified, and the major features have been sized, at least approximately. This is the process of refining the sizes of major elements and setting all important dimensions. This is done by means of intensive analysis of the system being designed and its component parts. The detail design block represents the part of the process in which fully dimensioned working drawings are produced of all parts to be manufactured. Minor components, such as bolts and other fasteners, may also be specified, and most tolerances will be specified during this stage.

Attention needs to be drawn to the characteristic of the design process that design decisions made earlier in the process provide constraints on later decisions. For this reason, the further the design progresses, the more tightly constrained it becomes. The conceptual designer operates with very few constraints. This can be a problem for students and junior engineers, since at this stage there is no "reference frame" in which to operate. Once the concept has been set and the major features designed, all later design decisions must be compatible with those made earlier. The nature of the design decisions which must be made therefore becomes increasingly specific.

This presentation is misleading because the design process is presented here as being sequential, with clear boundaries between the different stages. This is far from being the case in practice. It will be seen, on Figure 14.1, that there are arrows in both directions along the design stem. This is meant to indicate the frequent situation in which as the design of a component, or subsystem, is pursued in greater detail, it becomes apparent that it cannot be built in the form postulated at earlier stages of the process. It is then necessary to go back to those stages of the process and change the design, requiring, in turn, rework of other parts of the system design. Students should learn that this kind of backtracking can greatly increase the duration and cost of the design process, and is to be avoided whenever possible. The point should be made that early correction of a problem is almost always less expensive in the long run. The real solution to this problem is to develop a number of alternative concepts in parallel. When problems arise it is then usually possible to solve them by switching to one of the alternative concepts, reducing rework. One of the important points of working in a team is that different team members can develop alternative concepts in parallel.

The most grievous problem with the presentation of Figure 14.1 is the presentation of the process as being a serial process with each stage completed before the next is initiated. This is counter to the most important feature of concurrent engineering, which is the extensive use of parallelism in the design process (Ward 1994). It is tempting to operate in a one-task-at-a-time mode, but the pressure from the marketplace for shorter and shorter product cycle times requires that each stage of the process be initiated long before its predecessor is complete. Again, different members of the team can be pursuing different stages of the design process in parallel. For example, the design of manufacturing tooling may be proceeding at the same time as analysis of the design itself. The key to making this work is excellent communication among all members of the team. Students need to be taught the importance of communication both within the team, and with other concerned parties such as sponsors and management. The point needs to be made that this cannot be left to chance, and that formal protocols and formal reviews are important communication tools.

It is worthwhile to also introduce some of the more advanced design methodologies, such as Quality Function Deployment. There is a great deal of terminology from Design for Value, Quality Circles, and other such concepts that is freely used in industry. An overview that provides some introduction to these concepts will be helpful to new graduates in making the transition to the world of industry.

It is also very important that students are required to estimate the cost per unit of producing their design, and that this be done on a realistic and comprehensive basis. The relationships between price and volume for different manufacturing processes, and the costs of specifying unnecessarily tight tolerances are fundamental to effective functioning as a design engineer in industry. Students must also appreciate the real costs of the design and development process itself in order to develop the discipline to be able to make good project management decisions. There is never enough time and money to pursue analysis to completion. An essential part of the design process is deciding how far it is necessary to analyze each function before making the necessary design decisions.

4. Social Issues

Design projects provide a convenient place in the curriculum to introduce instruction on social issues. These range from regulation and liability matters, through intellectual property issues to professional ethics.

Regulations and standards are, of course, an integral part of the design process. It is not possible to provide even an introduction to all of the bewildering array of regulatory agencies that govern different industries, and to the volumes of regulations that they have generated, without specialized courses. However, students must be made aware of the need for designers to be knowledgeable about regulations and standards applicable to their industry. They should also be made aware of the importance of maintaining proper documentation of their work for the purpose of satisfying the certification requirements of the regulating agency or agencies. A good tool for this purpose is the traditional designer's notebook. The discipline of putting all work into a suitable notebook, of signing and dating each page as it is completed, and indexing the work, carries a message on the importance of all of the designer's work, and it's potential value to the employer. It also brings home the importance of being able to establish the time at which each part of the work was done.

Of course, this type of documentation is equally important for establishing precedence and rights in patent proceedings. Students do need to understand what rights a patent establishes, and the general features of the procedure for obtaining a patent. The same is true for copyrights.

There are many misconceptions in the engineering community regarding product liability. Students should receive instruction on their responsibilities as professionals, and on those of the companies that may employ them. They should understand that a good faith commitment to designing quality and safety into a product is usually the best defense against product liability actions. It is certainly true that there are horror stories in which even companies that have fully complied with applicable regulations at the time of manufacture, and have used the best available technology, are still vulnerable to product liability judgments. See, for example, Yodice (1992) and Wolk (1993). The apparent enormity of these rather rare cases obscures the fact that they are aberrations and that companies that use good practices do have fewer claims and judgments made against them. Students should be made aware that a constructive approach to avoiding liability by early and frequent communication between the design team and the legal department is also important.

Finally, ethics and professionalism covers the areas outside of formal regulations that are essential to productive interactions with sponsors, clients, and other professionals. In our highly competitive industrial environment, the use of another's ideas without attribution has become a rather common practice. The maintenance of the respect and trust that is essential for all professional interactions is seriously damaged by this and other practices that are commonly justified by competition and proprietary interests. Students need to be aware of the negative effects of unethical or deceptive conduct, even when that conduct is perfectly legal. They also need to be aware of the responsibilities they assume as professionals, and the special status attained by certification as a professional engineer. They should be made aware that they are members of a professional community that has common interests, and that that community may be of assistance to them in advancing their careers. They also have a responsibility to that community to uphold its professional standards.

5. Innovative Curricula

Recently many design education programs across the country have incorporated some of the above features in their capstone design courses (Innovations, 1993). There are courses developed using concurrent engineering practices. Communication, group interaction, and group decision making become important. Developing trust, respect, and understanding of knowledge and methods used by people from different disciplines becomes crucial. Support systems and methods that can facilitate these interactions become important. In Chapter 8 Thurston discusses group decision making and a failure modes and effects analysis scheme for information flow. In Chapter 16 Ramanathan presents a knowledge-based software system that can facilitate access to timely information and knowledge by different group members of the concurrent design team so that better and informed decisions can be made. In Chapter 15 Ishii discusses structured methods for life-cycle design and design reviews. In this section we present a few selected case studies that present different ways in which educators have addressed the issues mentioned above.

Gabrielle (1994) reports using reverse engineering to teach design. That is, conducting an in-depth analysis to try to recreate the original design process. This is then followed by a re-engineering phase where students create new designs. For their projects they selected commonly used household products with which the students were familiar such as a hair dryer. They formed a team of five to six students to explore the different disciplines involved. At the end of the reverse engineering phase they found that the students had a real appreciation for the engineering involved in the design of even these simple products.

Fleischmann (1994) reports teaching environmentally responsible design to undergraduate students by integrating this theme into their curriculum. They used an integrated design approach in response to the industry demands that all engineering graduates understand how the engineering profession affects the environment. In order to cope with the amount of information students need in their curriculum, they integrated QFD and LCA tools into the environmentally motivated designs that students were carrying out. They developed educational materials that their faculty and students could use. These include a project manual that contains information on value and ethics, materials and recycling, and case studies, project management, planning, QFD, LCA, etc. The key to the success of their curriculum was to ensure that there was a balance between the complexity which inclusion of legal, ethical and regulatory framework provide and the students' ability to integrate this material into their projects.

We used product dissection in teaching life-cycle design issues to create world-class products. The premise was that most designs are evolutionary rather than revolutionary. As Brown shows in Chapter 9, there is creativity even in routine designs. Re-engineering certainly needs more creativity than routine design. The students were assigned to teams so that the team members were from as many different backgrounds as possible. This included industrial design and mechanical, electrical, and biomedical engineering. Each team had four members and had a product and a process coach who guided them through the process and product design issues, based on the information from the weekly homework exercises and internet communication on an as-needed basis.

In the course students learnt both the process issues and the structured product methods. Through a series of appropriately designed homework assignments the students applied these methods to their own products and teams. They carried out structured brainstorming for functional and value analysis of their product by creating a why and a how diagram (Ishii et al., 1994).

The students also did team-building exercises focusing on trust and respect by identifying what special knowledge and talent each team member possessed. Students were introduced to quality principles. Wesner, Hiatt, and Trimble's book, *Winning with Quality* (1994), was a required text for the students. Students used the Internet, group meetings, and class discussions for communication amongst themselves and with the product and the process coaches.

Students first took apart the product they selected and identified the functional and structural relationships. They learnt to transform customer requirements through QFD methods by creating a house of quality. They carried out value analysis, design for assembly, serviceability, design for recyclability, failure modes and effects analysis, and learnt about design trade-offs. They benchmarked their products and re-engineered their product along one of the life-cycle issues most suitable for their product. They learnt group decision making and their decision was arrived at through team consensus based on data from their DFX analyses.

The students reported that they developed an appreciation for obtaining and integrating information, learning to work with others, and applying the knowledge they had gained in their engineering training. One group chose to redesign along the user issues identified by QFD analysis. Another group chose to pursue safety issues through examination of regulations and user requirements, and the third focused on the recyclability issues after carrying out a serviceability analysis.

Bailey (1995) reports the results of a three-quarter capstone design course that takes industry-defined "real-life" design projects from product definition to prototype construction and testing. The customers of the student teams were the industry personnel who provided them with the projects. The student groups developed the design specifications using the QFD method, scheduling via critical path techniques, systematic generation of alternative designs, selection of criteria for optimal designs, and the construction and testing of the prototype. The students used the Internet for communication and collaboration and learnt about the relationship of the design process to technology transfer. Students worked in groups of three to four. They visited their industrial clients and developed their designs with a complete knowledge of the industrial setting. The results were impressive. One of the groups redesigned the tools and operation of molded packaging in such a manner that the net productivity doubled for the client. Both industrial engineering and mechanical engineering knowledge was used in the solution.

Beach (1993) describes his two-quarter integrated design, manufacturing, and marketability (IDMM) course. Cross-functional teams of four students from design, manufacturing, and marketing worked on projects provided by industry clients. The design was driven by the customer and constrained by finances. Team evaluation was based on the firm's profitability. Students obtained significant manufacturing hands-on experience and they were coached by practicing engineers. The students enjoyed learning different cultures and vocabulary and walking through the product realization process.

All students gained substantial experience in materials and manufacturing tools and made manufacturing decisions in the context of market-driven information. The students learnt shop practices and, through this experience, acquired respect for each others' abilities and learnt a common vocabulary. Each team had a volunteer coach from industry who was a practicing designer. They learnt about guarding proprietary information. Through the Process of Change laboratory the students learnt to combine the "soft" issues with the "hard" engineering design issues. By the end of the first quarter the students had learnt about conjoint analysis and marketing issues, shop practices and had their design mapped out and marketing analysis completed. In the second quarter they learnt topics such as QFD, process design, and product realization. They worked on the prototype development and completed the market simulation. The students made their presentations and received feedback from their industrial clients.

6. Summary

In this chapter we discuss a design education philosophy based on over 25 years of personal industrial and academic design experience and on the design theory and methodology research results of the last decade. The importance of project work is established and a design methodology based model for design education is presented. The importance of making students socially responsible for their design is emphasized. Several design education courses taught in different institutions are presented as case studies as models to provide guidance.

References

Bailey, R. E. (1995). Cooperative Industry/University Design Projects and the Education of Mechanical Engineering Seniors at The Ohio State University. Working Paper.

Beach, D. (1993). Integrated design, manufacturing and marketability. In *Innovations in Engineering Design Education, Resource Guide*. New York, NY: American Society of Mechanical Engineers, pp. 263–266.

Clausing , D. (1994). *Total Quality Development*. New York: NY. American Society of Mechanical Engineers Press.

Dixon, J. R. (1988). On research methodology toward a scientific theory of engineering design. *Artificial Intelligence for Engineering Design, Analysis and Manufacturing (AIEDAM)*, *1*(3).

Engineering Accreditation Commission Accreditation Board for Engineering and Technology, Inc. (1994). *Criteria for Accrediting Programs in Engineering in the United States*, New York.

Fleischmann, S. T. (1994). Design for recycling—Teaching environmentally responsible design. In *Innovations in ME Curricula for the 1990's*. New York, NY: American Society of Mechanical Engineers, pp. 15–18.

Gabrielle, G. A. (1994). The use of reverse engineering projects in a mechanical engineering senior design course. In *Innovations in ME Curricula for the 1990's*. New York, NY: American Society of Mechanical Engineers, pp. 1–6.

Hale, R. (1995). Quality function deployment and quality capture software. Presentation in the seminar on design and development in global market series, at the Ohio State University April 1995.

Hoover, C. W., and Jones, J. B. (1991). *Improving Engineering Design: Designing for Competitive Advantage*. Washington, DC: National Academy Press.

Innovations in Engineering Design Education (1993). *Resource Guide*. New York, NY: American Society of Mechanical Engineers.

Issii, K., Eubanks, F., and Beiter, K. (1994). ME 883 Life-Cycle Design Course notes. Mechanical Engineering Department, The Ohio State University. Columbus, Ohio.

Koen, B. V. (1994). Toward a strategy for teaching engineering design. *J. Engineering Education*, *83*(3), 193–202.

Roeder, C. L. (1994). Teaching engineering and business in parallel. In *Innovations in ME Curricula for the 1990's*. New York, NY: American Society of Mechanical Engineers, pp. 11–13.

Ullman, D. G. (1992). *The Mechanical Design Process*. New York: McGraw-Hill.

Ulrich, K. T., and Eppinger, S. D. (1995). *Product Design and Development*. New York: McGraw-Hill.

Waldron, K. J. (1992). Secret confessions of a designer. *Mechanical Engineering*, *114*(11), 60–62.

Ward, A. C. (1994). The second Toyota paradox. Proceedings of ASME Design Theory and Methodology Conference, DE-68 American Society of Mechanical Engineers, New York, pp. 79–90.

Wesner, J. W., Hiatt, J. M., and Trimble, D. C. (1994). *Winning with Quality*, Reading, MA: Addison Wesley.

Wilson, C. C., and Speckhart, F. H. (1994). Superior engineering design program. In *Innovations in ME Curricula for the 1990's*. New York, NY: American Society of Mechanical Engineers, pp. 27–29.

Wolk, A. A. (1993). Product liability: A plaintiffs' lawyer responds. *AOPA Pilot*, June, pp. 117–119.

Yodice, J. S. (1992). Product liability: A case study. *AOPA Pilot*, December, pp. 127–128.

15
Life-cycle Design

K. Ishii

Abstract. This chapter addresses life-cycle design and describes a model of simultaneous design reviews using *design compatibility analysis* (DCA). Early stages of design affect various issues that comprise the life-cycle cost of products as well as reliability and serviceability. Our research uses the concept of DCA to model the life-cycle cost and customer value of mechanical systems. DCA focuses on good and bad examples of designs and gives an overall evaluation of designs in a normalized scale. This chapter summarizes the methodologies and computer tools developed based on this model. Specifically, the chapter describes computer programs that evaluate layout designs and give suggestions for improvement.

1. Introduction

Life-cycle design is a practice of incorporating various values of a product in the early stages of design. These values include not only functionality but also manufacturability, serviceability, recyclability, etc. Figure 15.1 shows these values in the life-cycle of a product. It is essential that these issues be addressed at this time since life-cycle values and costs are "locked-in" once preliminary design is complete.

Life-cycle design is largely an organizational and managerial challenge. However, the rapidly advancing field of computer-aided design provides an opportunity to use computers to promote life-cycle engineering more effectively. Design for assembly (DFA) is perhaps the most mature of these disciplines. Boothroyd and Dewhurst (1983) and many others have proven that DFA using computers can provide significant cost savings. There are other computer programs that assist other aspects of life-cycle designs (Poli et al., 1988; Desa et al., 1989; Cutkosky et al., 1988; Duffy and Dixon, 1988; Turner and Anderson, 1988; Simmons and Dixon, 1985).

In order to promote life-cycle design, we need to model the practice to identify the essential elements: parties involved and the necessary information flow. This understanding will lead to not only organizational innovation

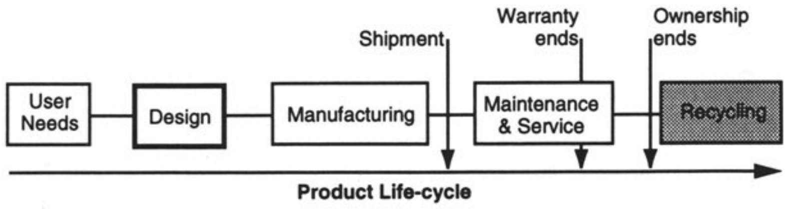


FIGURE 15.1. Life-cycle of products.

but also effective computer aids for life-cycle design. The main goal of our research is to develop a framework for computer programs that help designers to evaluate a candidate design with respect to various life-cycle values. Design compatibility analysis (DCA), has shown its utility in design for injection molding (DFIM), design for forging, design for serviceability (DFS), and process selection.

DCA is a model of design reviews in which experts with different responsibilities judge the candidate design from various angles. The concept has led to many computer programs. The framework is effective for designer training as well as a preliminary screen for manufacturability and trade analysis of several candidate designs. The object-oriented nature of this approach accelerates the modeling of the product values and allows us to implement "expert's" knowledge at critical stages of design.

2. Compatibility Methodology

2.1. *Model of Design Reviews*

In the past several years, we have been developing a flexible methodology that supports life-cycle evaluation of designs. While we do not claim that our framework covers every aspect of life-cycle design, it has proven to be both versatile and adaptable. We have applied our compatibility approach to design for assembly (Adler and Ishii, 1989), design for injection molding (Beiter and Ishii, 1990; Ishii et al., 1989b), forging process design (Maloney et al., 1989), design for serviceability (Gershenson and Ishii, 1991) and process selection (Ishii et al., 1990). This section gives a brief description of our general approach.

The central idea of our model is to evaluate simultaneously a candidate design from multiple viewpoints (Ishii et al., 1988, 1989a). That is, we seek to model "round table" design reviews in which the various experts evaluate the proposed design and suggest improvements (Figure 15.2). These suggestions primarily focus on modification of the candidate design, but may also be directed to respecification of the process (use of an alternative molding machine, etc.) or even renegotiating the user requirements.

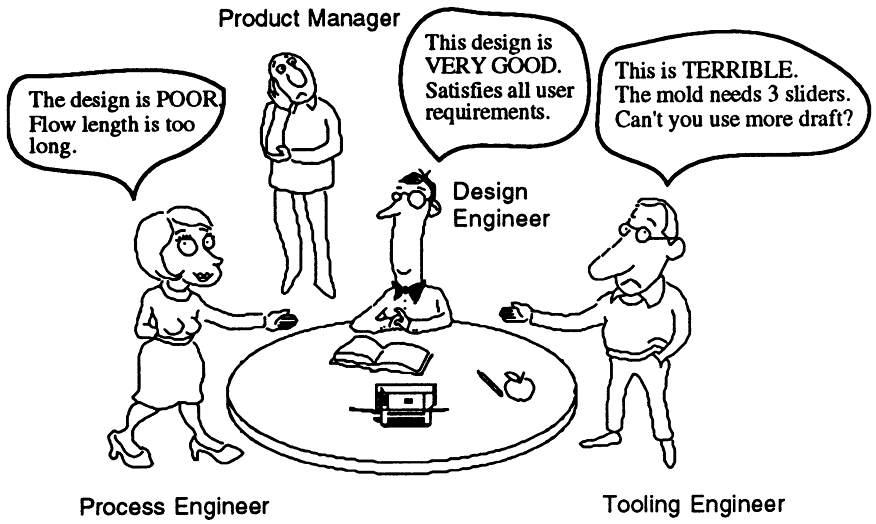

Product Manager
The design is POOR. Flow length is too long.
This design is VERY GOOD. Satisfies all user requirements.
This is TERRIBLE. The mold needs 3 sliders. Can't you use more draft?
Design Engineer
Process Engineer
Tooling Engineer

FIGURE 15.2. The model of design review. The compatibility approach models a design review in which multiple reviewers with various expertise study a candidate design. Each gives his/her own comment about the compatibility between the design and the life-cycle value for which he/she is responsible. Note that the compatibility comment includes "in between" cases such as "poor."

Note that in Figure 15.2, each expert describes his/her view of the compatibility between the candidate design and his/her field of expertise using an adjective qualifier: excellent, very good, good, poor, bad, and very bad. They do not have to be adjectives since the qualifiers are eventually mapped into a [0,1] measure. The key here is that some compatibility issues are absolute design rules, i.e., definitely not permitted, or absolutely good, while some others are not so extreme. The qualifier "poor" indicates that the compatibility is undesirable, but if other constraints dominate the final decision, then the expert will accept the design.

Our approach views the experts' design knowledge as compatibility comments and compiles them as C-data. A C-data contains an ID number, the associated design components/features, compatibility descriptor such as "very good" and "poor," reasons and suggestions, and, most importantly, the conditions for the data to be true. Some C-data also looks at compatibility problems within the candidate design and inconsistency in the specifications.

C-data:	ID	= mbase1
	elements	= plate, ribs
	descriptor	= poor

reason = with wall thickness of Th and edge gating, the maximum flow length for the melt is too long and may require very high packing pressure.

suggestion = 1. use a center sprue gate
2. consider multiple gating
3. increase thickness

conditions = gate location = edge of the plate.
plate-thickeness (Th),
attribute (maximum flow-path) = FP,
attribute (suggested flow-path) = MFP
FP > MFP (15.1)

A collection of C-data comprises the compatibility knowledge-base (CKB).

$$\text{CKB} = \{\text{c-data} \mid \text{c-data} \subset G \times X \times P \times [0,1]\} \tag{15.2}$$

where:

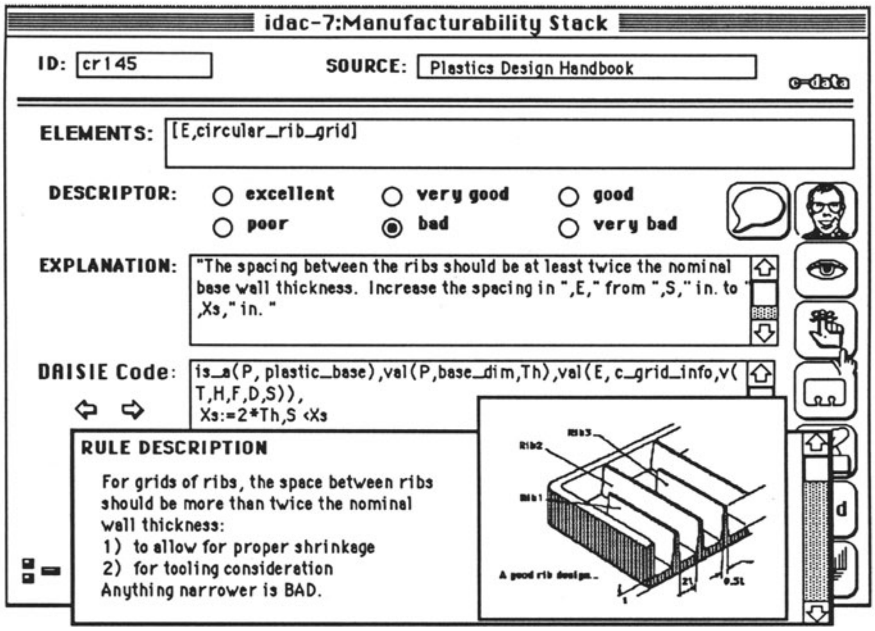
G:	Universe of discourse of the user requirements
X:	Universe of discourse of the candidate design
P:	Universe of discourse of the decisions related manufacturing and other life-cycle process
[0, 1]:	Normalized measure between 0 and 1.

That is, CKB is a set of relations between the specification space G, design solution space X, the life-cycle process space P, and a rating between 0 and 1. Figure 15.3 is an example of a poor compatibility for the design of injection molded parts.

Our model further looks at how the design team combines everybody's comments, makes compromises, and arrives at a total "consensus" evaluation. Figure 15.3 shows the concept of DCA, which models the evaluation process. DCA is a knowledge-based technique for calculating the total normalized measure of compatibility. At any time in the development process, designers can check their candidate design by comparing the proposed design with CKB. The compatibility model also utilizes the attribute rules (qualitative or quantitative) which reason about the characteristics of the candidate design and the detailed implications of the specifications. An attribute rule may involve the use of design formulae (e.g., stress equations) frequently used by designers and process engineers.

2.2. *Decomposing Candidate Designs to Elements*

In order to develop a computer program that utilizes the compatibility knowledge, we must describe the data in a form the program can recognize. More specifically, the preconditions of C-data must be represented with a standardized set of parameters. This section introduces the notion of "design



idac-7:Manufacturability Stack

ID: cr145

SOURCE: Plastics Design Handbook

ELEMENTS: [E,circular_rib_grid]

DESCRIPTOR: ○ excellent ○ very good ○ good ○ poor [✓ bad] ○ very bad

EXPLANATION: "The spacing between the ribs should be at least twice the nominal base wall thickness. Increase the spacing in ",E," from ",S," in. to ",Xs," in. "

DAISIE Code:
```
is_a(P, plastic_base),val(P,base_dim,Th),val(E, c_grid_info,v(
T,H,F,D,S)),
Xs:=2*Th,S <Xs
```

RULE DESCRIPTION

For grids of ribs, the space between ribs should be more than twice the nominal wall thickness:
1) to allow for proper shrinkage
2) for tooling consideration
Anything narrower is BAD.

FIGURE 15.3. Representative C-data. Each compatibility information element is contained within an individual data card. This card contains the type of element in question, the conditions for it to be true, and a rating if this condition is true.

elements" to allow uniform representation of C-data. Such representation is essential if computers are to relate a candidate design to the compatibility knowledge base.

Each expert "breaks up" the part design into different building blocks or design elements. Since these design elements are associated with design rules, it is important to identify the decomposition process of each expert. These elements serve as one of the data organization keys in our compatibility knowledge base. Using the injection molding example, we define the elements of a proposed plastic design so that we can organize the compatibility information by discrete elements:

A design element (s_i) is the smallest physical unit in a design that is of any interest to an expert who is evaluating the design.

Naturally, the element decomposition of a candidate design will be different among groups of experts. We have adopted the designer's language in collecting and organizing design rules and guidelines. Hence, the compatibility data will also be represented in terms of the design elements derived from the designer's perspective. In injection molding, an example of such a design element is Boss 1 shown in Figure 15.4.

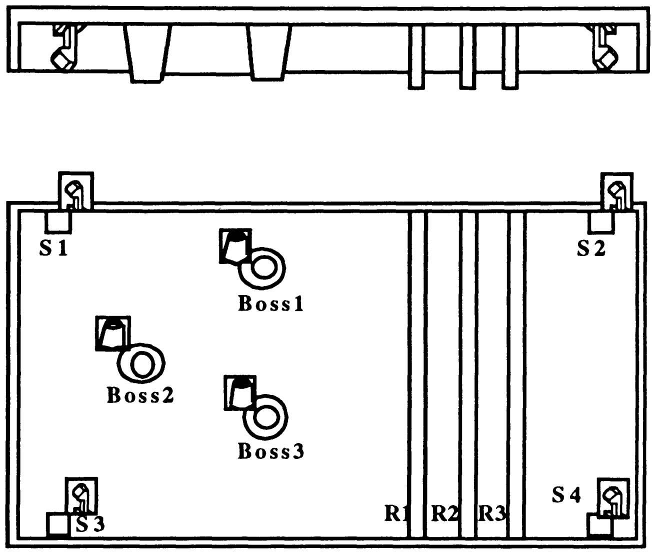


FIGURE 15.4. Model for a computer plastic cover. This sketch environment allows the user to define individual elements of the proposed design as well as interactions between elements. Each element (boss, rib, snap, etc.) is further defined by answering questions regarding their size, thickness, and overall dimensions.

The tooling engineer or process engineer views the candidate design from different perspectives, which typically involves several design elements. For example, a tooling engineer may identify a design rule associated with how closely a rib can be incorporated in a mold. This rule will involve not just one rib but two, e.g., R1 and R2 in Figure 15.4. Hence, two rib elements will fill the ⟨elements⟩ slot in the corresponding C-data.

In short, the ⟨element⟩ slot in the C-data is a mapping for cross-referencing the interest of the designers, tooling engineers, and molders. As the next section explains, DCA checks the candidate design for compatibility of each element with respect to other elements.

2.3. Computation of the Compatibility Index

For any individual expert or combinations of experts, the model computes a match index (MI), a measure of compatibility between the expert's rules and the design. Figure 15.5 shows the flow of DCA.

Given a description of the proposed design and the design specifications, DCA analyzes the individual elements relative to the specifications. Then, for each element that makes up the design, DCA selects from the compatibility knowledge base the data that applies to it as matching compatibility data (MCD). The applicability criteria of the C-data is twofold: (1) the element

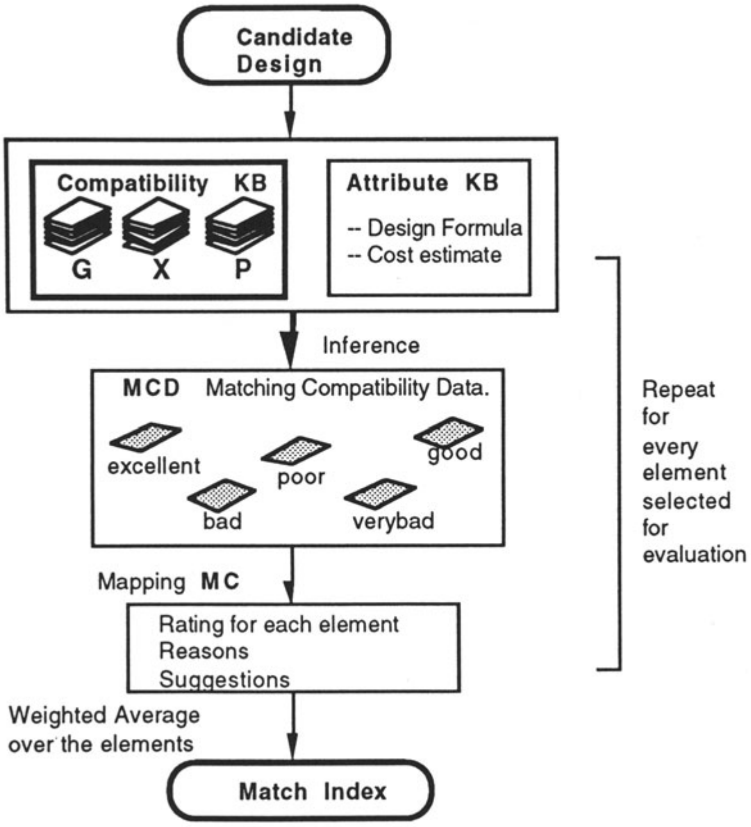


FIGURE 15.5. Flow of DCA.

must be referred to by the C-data, and (2) the design must satisfy the conditions in the C-data.

Then, the program rates the compatibility of each design element with respect to applicable C-data and computes the match coefficient index. This computation utilizes the function MC which maps MCD into a number between 0 and 1. The function MC operates as follows. Each C-data in MCD includes an adjective descriptor that takes the value {excellent, very good, good, fair, bad, very bad}. The program maps these descriptors into a numerical code {1.0, 0.8, 0.6, 0.4, 0.2, 0.0}, respectively. Hence, the n adjectives are mapped to n set of numbers between 0 and 1. We then use the function *bestinfo* to total this set of numbers into a single match coefficient, $M(s) \in [0, 1]$. In our application, the function *bestinfo* takes the rating of the worst C-data if there is at least one "negative" comment (ratings less than 0.5) about the design; otherwise, *bestinfo* takes the best C-data. If MCD is empty, i.e., there is no reason to believe the design is bad nor good, the match coefficient is assigned 0.5. Note that the function MC is user definable, i.e., one may adopt other mapping such as taking the mean.

The total evaluation for the entire set of elements is the **match index**:

$$\mathrm{MI} = \Sigma_K u(s) \cdot M(s) \tag{15.3}$$

where:

MI = the match index

K = the set of design elements

$u(s)$ = the weight of evaluation of element s $[\Sigma_K u(s) = 1.0]$

$M(s)$ = value of individual C-data

Hence, design compatibility analysis gives a normalized rating for a candidate design.

$$\mathrm{DCA}: G \times X \times P \times \mathrm{CKB} \rightarrow [0,1] \tag{15.4}$$

Note that the match index is only an averaged measure of compatibility and does not reflect situations where most design elements are compatible but some minor elements are not. This element may seriously jeopardize the entire design. Hence, a good design has a high match index and a narrow range of the match coefficients over the set of design elements.

Note that each design element has an associated weight of evaluation, $u(s)$. As described previously, a design element is related to the functional decomposition of the design. Hence, we can interpret $u(s)$ as the weight of importance of the associated function of the element. A designer can assign the weights according to his/her needs. As a default, each element will have equal weight. The match index, then, is the measure of how well the design satisfies the requirement and how compatible this function is with other elements.

3. Applications of DCA

DAISIE (Designers AId for SImultaneous Engineering: Adler and Ishii, 1989) utilizes an object-oriented programming environment devoted to aid life-cycle design using DCA. The underlying languages are Prolog and HyperCard. DAISIE has served as a platform for several life-cycle design aids:

1. Design for serviceability (Gershenson et al., 1990)
2. Material and process selection (Ishii et al., 1990)
3. Design for injection molding (Ishii et al., 1989b)

In addition, a research group at Stanford University applied DAISIE to design for assembly (Ishii, Adler, and Barkan, 1988). DCA has also been successfully applied to the tribological design of machine elements (Ishii, Klinger, and Hamrock, 1990).

Various design interfaces are available depending on the application. For the injection molding system DAISIE/DFIM, designers can describe their proposed design in a "shorthand sketch" (MacDraw-like sketching environment, Figure 15.6). DAISIE asks the user questions regarding the requirements and process constraints through HyperCard. The system uses DCA to evaluate the compatibility and suggests remedies/improvements through textual and pictorial information.

3.1. Design for Serviceability and Reliability

A current trend in industry is to produce designs that are as simple as possible to assemble (design for assembly [DFA]). Very often, but not always, DFA leads to more reliable designs due primarily to the reduced number of parts. Unfortunately, DFA may sometimes lead to designs that are very difficult to service.

Some systems designed for assembly may be impossible or very difficult to replace in the field. The lack of tunability or adjustability of some DFA designs may degrade the performance of the device after servicing. The possible enhanced reliability due to DFA and modular designs, i.e., reduced service frequency, could be offset by an increased cost of each repair. Hence, manufacturable designs without thorough consideration for serviceability and reliability could lead to unexpected increases in servicing and warranty costs. In addition, the intangible effects on customer satisfaction could be quite significant.

Many companies have compiled comprehensive guidelines for serviceability design. The guidelines address, for various service modes, (1) provisions to detect servicing needs, (2) design features to enhance the ease of servicing, and (3) estimated life-cycle service cost. However, the strong push for manufacturability (assembly, modularity) sometimes compromises serviceability and reliability considerations beyond a justified level. We are developing applications for some major automotive industries to help designers to access their proposed designs with respect to serviceability.

Recently, we have developed a computer aid to analyze the life-cycle service cost of automotive systems and provide suggestions that improves the serviceability of candidate designs (Gershenson and Ishii, 1991). The system, based on DCA, allows the user to describe the layout/configuration of preliminary designs using icons and links (Figure 15.6). The system further performs what we call "phenomena-based serviceability analysis." This method identifies cost driving service modes, analyzes the life-cycle service cost based on these phenomena (service modes), and maps the costs to the

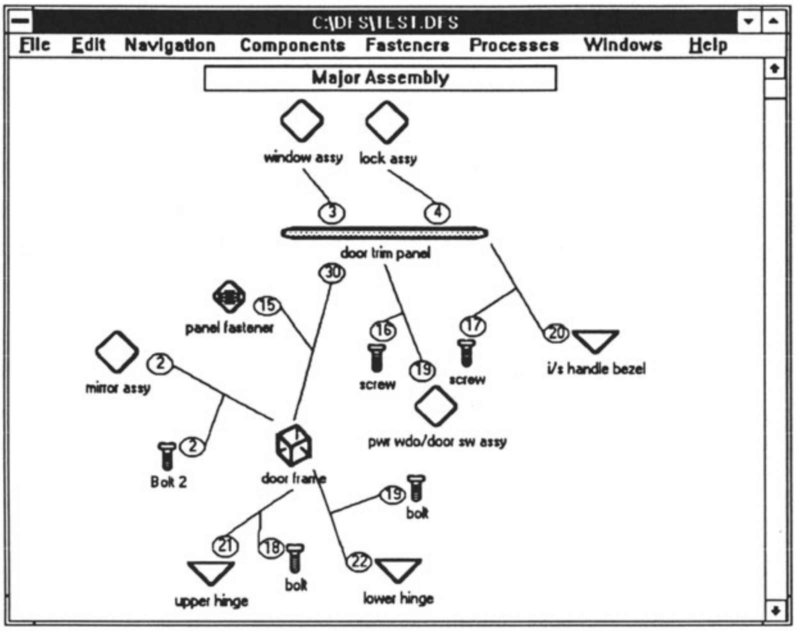


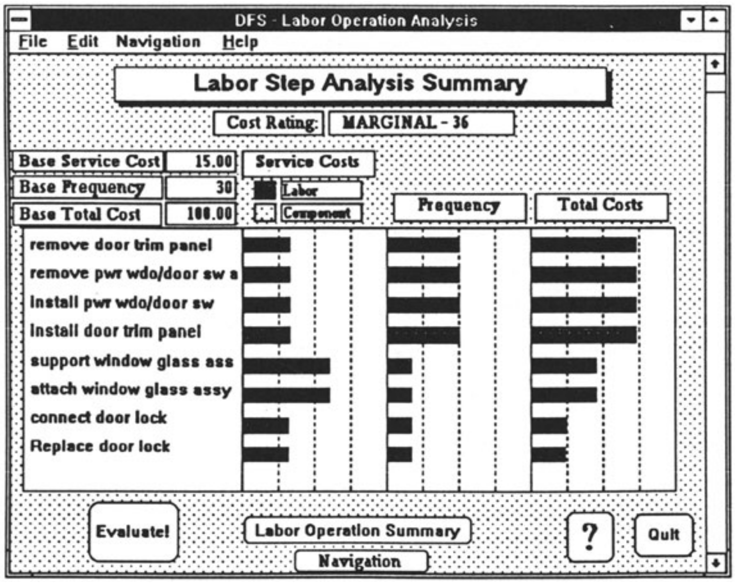
FIGURE 15.6. Design description using icons. This pallet allows the user to define the method of assembly/disassembly of the individual components or modules within a car door.

actual construction of the candidate design (Figure 15.7). Figure 15.8 shows the design suggestions.

This method, service mode analysis (SMA) has been applied to door hardware systems at General Motors, to power train systems at Ford, and to appliances at General Electric. We are currently extending the methodology to accommodate advanced planning for product retirement and recycling of recovered materials.

3.2. Material and Process Selection

Most people agree that the cost and quality of a product are "locked" into the layout design. Many companies are actively pursuing means to integrate the life-cycle values of the product early in its development. In particular, design for manufacturability (DFM) has provided engineers a systematic methodology to reduce development time, cut production cost, and reduce defects. DFM typically focuses on the particular manufacturing process, e.g., machining, stamping, injection molding, assembly, etc., and seeks to incorporate into the early product design features that can prevent manufacturing problems and significantly simplify the production process.



Labor Step Analysis Summary

Cost Rating: MARGINAL - 36

Base Service Cost	15.00
Base Frequency	30
Base Total Cost	100.00

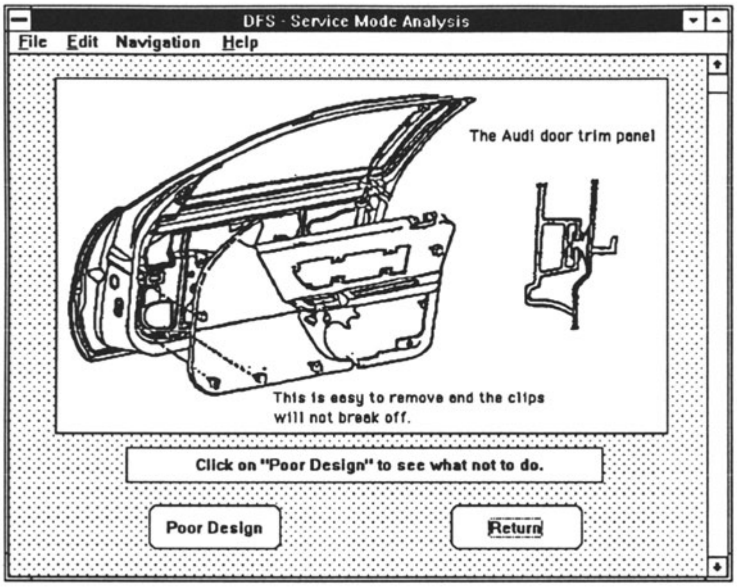
Service Costs: ■ Labor ▢ Component

Frequency

Total Costs

- remove door trim panel
- remove pwr wdo/door sw a
- install pwr wdo/door sw
- install door trim panel
- support window glass ass
- attach window glass assy
- connect door lock
- Replace door lock

FIGURE 15.7. Labor operations. From this output it can be determined graphically how different "phenomena" or conditions will affect serviceability cost.



This is easy to remove and the clips will not break off.

Click on "Poor Design" to see what not to do.

FIGURE 15.8. Design suggestions. Suggestions are provided with visual examples of both good and bad design.

While this type of activity certainly enhances product competitiveness, it usually applies to a specific process. What precedes DFM is a very important decision; selection of the material and manufacturing process.

Frequently encountered process selection targets include (1) electronics housing: sheet metal forming or injection molding; and (2) automotive parts: machining or die casting or investment casting. These decisions not only affect the DFM methods that follow, but also the product's overall market competitiveness.

A variety of factors influence this decision, many of which cannot be estimated accurately, e.g., volume of sales. While there are many handbooks for qualitative guidance in selecting a process, they do not provide a quantitative means to compare the suitability of each process to a given part. Today, most engineers select a process based on their experience and intuition in addition to "guesstimation" (estimation based on educated guesses) of many of the influencing factors. Engineers can greatly benefit from a design tool that allows them to compare different processes in a more rational, systematic manner, utilizing as much quantitative information as possible.

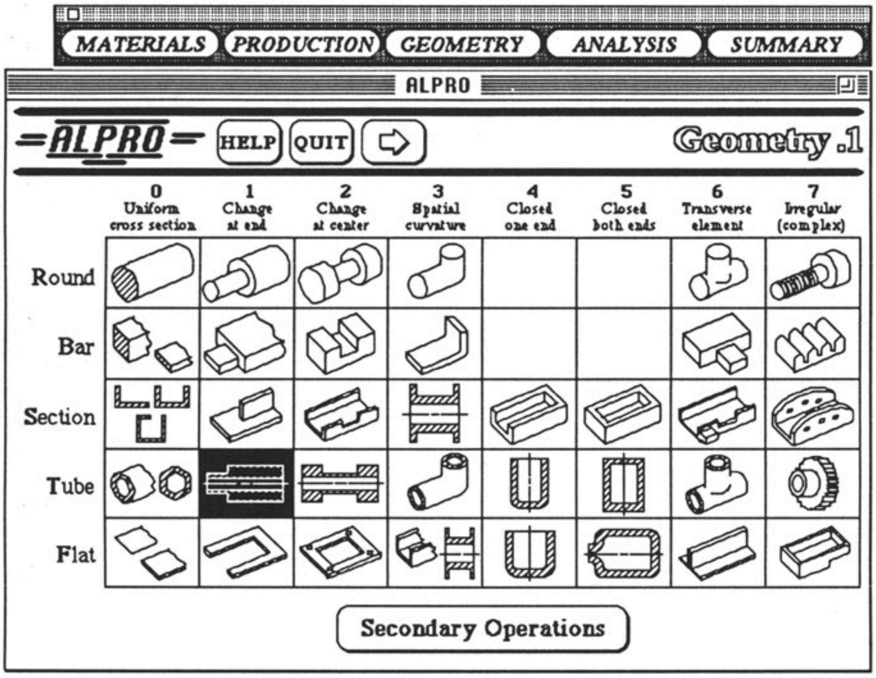
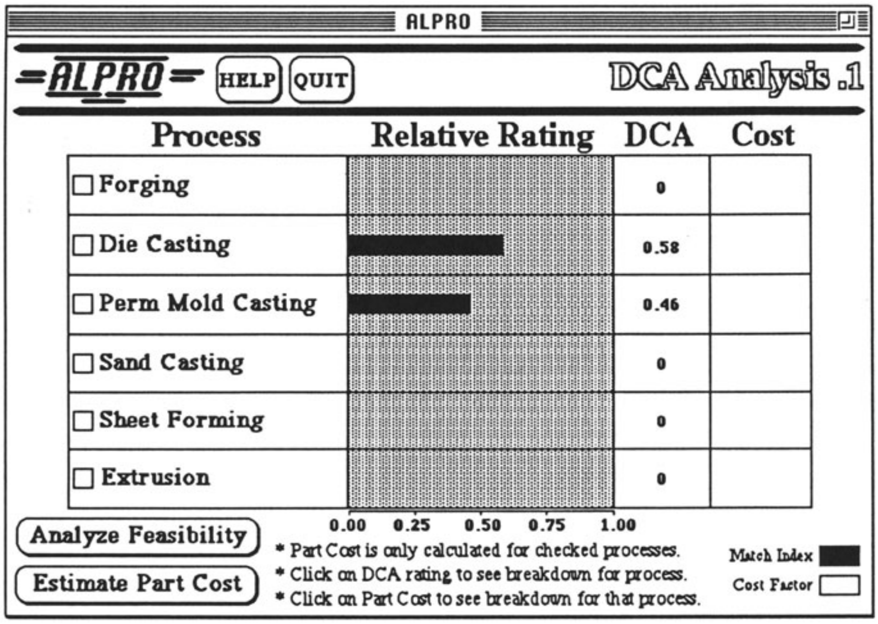


FIGURE 15.9. Geometric specifications. This card is asking the user for information regarding general shape. The total enclosed volume of the product is also specified.



Process	Relative Rating	DCA	Cost
☐ Forging		0	
☐ Die Casting		0.58	
☐ Perm Mold Casting		0.46	
☐ Sand Casting		0	
☐ Sheet Forming		0	
☐ Extrusion		0	

0.00 0.25 0.50 0.75 1.00

* Part Cost is only calculated for checked processes.
* Click on DCA rating to see breakdown for process.
* Click on Part Cost to see breakdown for that process.

Match Index ■
Cost Factor □

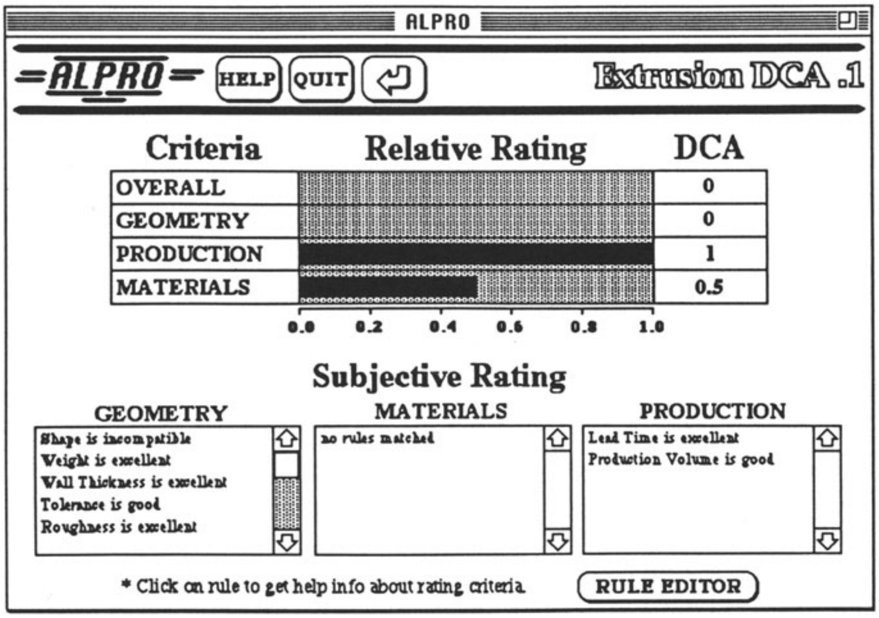
FIGURE 15.10. Ranking of candidate processes. The program uses DCA to rank the compatibility of candidate processes to the product specifications.

The application HyperQ/Process is used to determine the most appropriate and cost-effective manufacturing process for a proposed design. Again, it uses DCA to evaluate the compatibility of a given specification with various candidate manufacturing processes. The program receives the product specifications in three modules: (1) geometry, (2) production, and (3) material and mechanical strength (Figure 15.9). Then DCA ranks the compatibility of commonly used manufacturing processes (Figure 15.10). If the user requires further analysis, DCA illustrates the details of its compatibility studies (Figure 15.11). The program is undergoing field testing in several companies.

3.3. *Design for Injection Molding*

This application, which we now call **HyperDesign/Plastics**, evaluates proposed product designs for their compatibility with the injection molding process.

The user sketches a design then selects icons from a floating pallet and places them upon the drawing (Figure 15.12). The application then asks questions regarding each feature, such as, rib height, wall thickness etc. Once all the feature data and interaction information is entered the application evaluates the design with respect to its built in C-data. A rating from 0 to 100



Criteria	Relative Rating	DCA
OVERALL		0
GEOMETRY		0
PRODUCTION		1
MATERIALS		0.5

Subjective Rating

GEOMETRY
Shape is incompatible
Weight is excellent
Wall Thickness is excellent
Tolerance is good
Roughness is excellent

MATERIALS
no rules matched

PRODUCTION
Lead Time is excellent
Production Volume is good

* Click on rule to get help info about rating criteria

FIGURE 15.11. Process-specific output. This card shows a specified process rated in the areas of material, production, and geometry. The card also shows which C-data applied to this example, and leads the user to improvement ideas.

is returned along with reasons for the rating and suggestions to improve the design (Figure 15.13).

4. Conclusion and Future Work

This chapter addressed the methodologies and computer programs that help designers incorporate various life-cycle values into early designs of a product with appropriate balance. The proposed method of design compatibility analysis captures the design guidelines and cost models in a compatibility format. DCA uses the object-oriented compatibility data to (1) compute an overall "goodness" of designs, (2) give reasons, and (3) provide suggestions for improvement.

The previous applications have proven useful as training tools for inexperienced designers. Our current effort addresses the on-line use of the previous applications as well as the integration of multiple life-cycle values. Through these efforts, we hope to enhance the concept of design compatibility analysis to encompass not only qualitative guidelines but also quantitative cost models. More specifically, our future research will address:

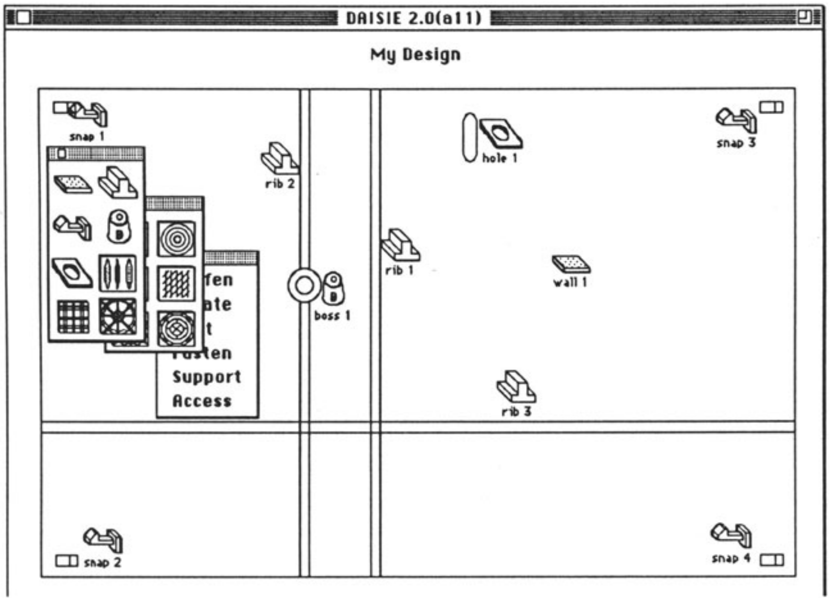
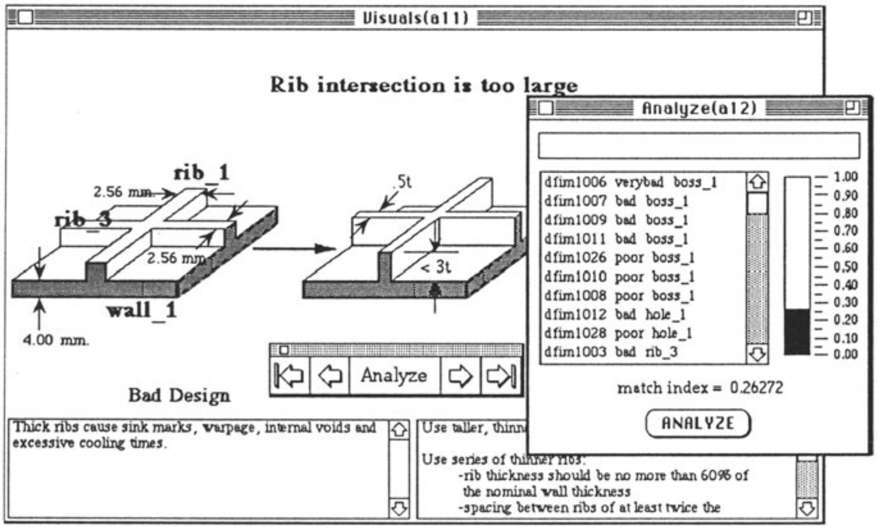


FIGURE 15.12. HyperDesign/Plastics: Sketcher. Users input their design by sketching it in a MacDraw like environment and then identifying design elements by placing appropriate icons on the sketch. Future versions will import feature information directly from a CAD application.



Rib intersection is too large

```
dfim1006 verybad boss_1
dfim1007 bad boss_1
dfim1009 bad boss_1
dfim1011 bad boss_1
dfim1026 poor boss_1
dfim1010 poor boss_1
dfim1008 poor boss_1
dfim1012 bad hole_1
dfim1028 poor hole_1
dfim1003 bad rib_3
```

match index = 0.26272

Thick ribs cause sink marks, warpage, internal voids and excessive cooling times.

Use taller, thinn

Use series of thinner ribs:
-rib thickness should be no more than 60% of the nominal wall thickness
-spacing between ribs of at least twice the

FIGURE 15.13. Evaluation and suggestion card. After the application has evaluated the proposed design, the program displays suggestions for improvement (visual representations of the C-data).

1. Systematic identification of user's life-cycle requirements
2. Methods to represent and store design alternatives
3. Comprehensive measure of "goodness" of design

In addition to these fundamental goals, our group is now addressing **recyclability**. The natural extension of manufacturability and serviceability is the impact of product designs on the utilization of the components and materials after the product's useful life. The question of recyclability is critical as we deplete the earth's limited resources and quickly fill our environment with hazardous waste. Our research efforts focus on design constructs that are compatible with easy disassembly, separation, and identification of source materials, and their reprocessing.

Acknowledgments

This paper is a result of several projects on life-cycle engineering. Sponsors of these projects include NSF/DDM, NSF/ERC at Ohio State, US Army Material Command, Alcoa, Apple Computer, BFGoodrich, Hewlett Packard, NCR, General Electric, General Motors, Ford, Nissan, and Rockwell. We deeply appreciate their support. The author expresses his sincere appreciation for the industrial liaisons and students who have contributed to this work. Special thanks to John Gershenson and Kurt Beiter who contributed the serviceability and injection molding examples, and to Steven Krizan and C. Fritz Eubanks who helped in the preparation of the manuscript.

References

Adler, R. E., and Ishii, K. (1989). DAISIE: Designer's aid for simultaneous engineering. *ASME Computers in Engineering 1989*, Anaheim, California, July, 1989.

Barkan, P. (1988). Simultaneous Engineering: AI helps balance production and bottom lines. *Design News*, March 1988.

Beiter, K., Ishii, K., and Hornberger, L. (1991). Geometry-based index for predicting sink mark in plastic parts. *Proc. of the ASME Design Theory and Methodology Conference*, September, 1991, Miami FL.

Boothroyd, G., and Dewhurst, P. (1983). Design for assembly: A designer's handbook. Wakerfield, RI: Boothroyd Dewhurst Inc.

Cutkosky, M. R., Tanenbaum, J. M., and Muller, D. (1988). Features in process-based design. *ASME Computers in Engineering 1988*, *1*, 557–562.

Desa, S., et. al. (1989). The application of a design for producibility methodology to complex stamped products. *Proc. of the 1989 ASME Winter Annual Meeting*: *Concurrent Product and Process Design*. Dec. 1989, San Francisco, CA.

Duffey, M. R., and Dixon, J. R. (1988). Automating the design of extrusions: a case study in geometric and topological reasoning for mechanical design. *ASME Computers in Engineering 1988*, *1*, 505–511.

Gershenson, J., and Ishii, K. (1991). Life-cycle serviceability design. *Proc. of the ASME Design Theory and Methodology Conference*, September, 1991, Miami FL.

Ishii, K., and Barkan, P. (1987). Design compatibility analysis: a framework for expert systems in mechanical system design. *ASME Computers in Engineering 1987. 1*, 95–102.

Ishii, K., Adler, R., and Barkan, P. (1988). Application of Design Compatibility Analysis to Simultaneous Engineering. *Artificial Intelligence for Engineering Design, Analysis and Manufacturing* (*AI EDAM*), *2*(1), 53–65.

Ishii, K., and Goel, A. (1989a). A model of simultaneous engineering. In J. Gero (Ed.), *Artificial Intelligence in Engineering*: *Design*. Computational Mechanics Institute. Transaction of the 4th International Conference on AI Applications in Engineering, July 1989, Cambridge UK. (AI ENG 89), pp. 484–501.

Ishii, K., Hornberger, L., and Liou, M. (1989b). Compatibility-based design for injection molding. *Proc. of the 1989 ASME Winter Annual Meeting*: *Concurrent Product and Process Design*. Dec. 1989, San Francisco, CA.

Ishii, K., Klinger, J., and Hamrock, B. (1990). Compatibility-based design for contact stress. *Proc. of the ASME Design Automation Conference*, September, 1990, Chicago, IL. Vol. DE-Vol. 23-1. pp. 323–330.

Ishii, K., Lee, C. H., and Miller, R. A. (1990). Methods for process selection in design. *Proc. of the ASME Design Theory and Methodology Conference*, September, 1990, Chicago, IL. Vol. DE-27. pp. 105–112.

Ishii, K., Krizan, S., Miller, R. A., and Lee, C. (1991). HyperQ/Process: An expert system for manufacturing process selection. In G. Rzevski, (Ed.), *Applications of Artificial Intelligence in Engineering VI*. Oxford, UK: Computational Mechanics Publications, pp. 405–422.

Makino, A., Barkan, P., Reynolds, L., and Pfaff, E. (1989). Design for serviceability expert system. *Proc. of the 1989 ASME Winter Annual Meeting*: *Concurrent Product and Process Design*. Dec. 1989, San Francisco, CA.

Maloney, L., Ishii, K., and Miller, R. A. (1989). Compatibility-based selection of forging machines and processes. *Proc. of the 1989 ASME Winter Annual Meeting*: *Concurrent Product and Process Design*. Dec. 1989, San Francisco, CA.

Pinilla, J. M., Finger, S., and Prinz, F. B. (1989). Shape feature description and recognition using an augmented topology graph grammar. Proceedings of the 1989 NSF Engineering Design Research Conference.

Poli, C., and Fernandez, R. (1988). How part design affects injection molding tool costs. *Machine Design*, November, 24. pp. 101–104.

Simmons, M. K., and Dixon, J. R. (1985). Expert systems in a CAD environment: injection molding part design as an example. *ASME Computers in Engineering 1985*.

Spies, K. (1957). Categorizing Closed-Die Forgings. (in German), Werkstuttstechnik u. Masch.-Bau, Vol. 47, pp. 201–205.

Turner, G. P., and Anderson, D. C. (1988). An object-oriented approach to interactive, feature-based design for quick turnaround manufacturing. *ASME Computers in Engineering 1988*, *1*, 551–555.

16
Support for Workflow Process Collaboration

JAY RAMANATHAN

Abstract. Global competition has created a tremendous need to streamline the total collection of activities (or the workflow process) by which high-quality products are designed, maintained, and serviced. To meet this need, companies are embarking on practices like integrated product-process design and team-oriented management. These practices attempt to identify and address different types of constraints early during design to reduce problems and iterations in "downstream" activities. Numerous individuals must then apply these practices when developing each product component. Given that various disciplines and departments are also involved, the problem of managing the workflow process is quite complex. Furthermore, existing applications developed over the years must somehow be leveraged in any solution. A fundamental challenge addressed here is to develop process-driven information systems to actively assist the way in which each worker, within each department or job category, performs each one of these activities *correctly*. By ensuring correctness and timeliness within the context of the overall workflow process, dramatic cost and cycle-time reductions are achievable while producing quality products.

1. Introduction

Collaboration required to support enterprise-wide practices often involves numerous disciplines and must be structured to ensure value is, in fact, incrementally added to the product. While structure is often desirable, some processes are more ad-hoc. The term *workflow process* is used here to refer to the collection of activities (structured or ad hoc) that must be performed by different types of workers who must collaborate for any professional endeavor. (The adjective *workflow* distinguishes the human work activities from processes performed by tools that manipulate materials, and from business processes that focus on functional decomposition as opposed to the flow of control between activities, as we shall see later.)

Focusing on workflow process support for concurrent engineering, it is important to note that workers from *different departments or job categories* must *collaborate* to both identify and address two types of interacting constraints. One type is *process-related* and arises out of enterprise-specific policies and manufacturing issues (such as properties of existing tools or quality criteria). These process constraints, in turn, provide the context for the other type, *product-related* constraints. Based on process constraints, product requirements are refined and addressed during conceptual and detailed design. Terms like *concurrent engineering*, *integrated product design*, and *life-cycle engineering* are often used to describe the identification and refinement of the interacting product/process constraints and to differentiate this design methodology from detailed product design performed in isolation.

Networked hardware and software systems are required for the cost-effective support for collaborative work. But, "integration" must go beyond the mechanics of transmitting, exchanging, and sharing data in a networked environment. The hardware/software system must actively "know and assist" the collaborative workflow process. An enterprise may follow a specific discipline or workflow methodology in its use of existing software. This knowledge must be represented in a machine-readable form and used to actively guide engineers through a collaborative workflow process that exploits investment in software. Here, we will refer to an integrated software system that supports a workflow as an *assistant*.

Considerable work has been done in groupware (ACM, 1991; Ishii, 1991) and a variety of products (like mail and calendar systems) are now available commercially. The focus of these efforts has been in facilitating *unstructured* collaborative interaction between users at different workstations. An example is the collaborative editing of documents or drawings. In this case, one user performs the editing and other users at other workstations can see the results and make contributions by editing the same drawing. In addition, artificial intelligence research has concentrated on the problem-solving nature of detailed design conducted by individuals. More recent research in design emphasizes the process-oriented nature of large-scale design (Waldron, 1988; Waldron and Waldron, 1988). Issues of making the project plan and team decisions visible to the concurrent engineering team have been studied by efforts reported in Kyung (1991) and Klein (1993). Process modeling and project management within software engineering have been addressed by Krasner (1992). The design of knowledge-based information systems has been reported in Ashok (1987); Chandrasekaran and Johnsonson (1992); Fisksel (1989); Gupta and Madnick (1987); Kannapan (1993); Ramanathan and Sarkar (1988); Sarkar (1989); and Williams (1990).

The cost-effective development of information systems for design, manufacturing, and logistics has also been the concern of a spectrum of national and international efforts, like the United States Air Force's IISS (Integrated Information Support System) (Althoff, 1990; WRDC, 1990), EIF (Enterprise Integration Framework) (EIF, 1990), IICE (Information Integration for Concurrent Engineering) (Mayer, 1993), DARPA's DICE (Defense Advanced Research Projects Agency Initiative in Concurrent Engineering)

(Ramana, 1993), CALS Industry Steering Group (CALS, 1991), KIDS (Knowledge Integrated Design System) (KIDS, 1990), and European Commonwealth's ESPIRIT (AMICE, 1989). These efforts are being conducted to define standards-based information architectures to support the requirements of concurrent engineering, manufacturing, and logistics. While all these efforts emphasize that building cost-effective information systems is an interdisciplinary effort involving organizational and behavioral theories, networking, database, artificial intelligence, and software engineering techniques, they each also emphasize the need for *process management*. Some efforts go beyond stating the need and outline a strategy. A good example is the draft standard (U.S. Department of Commerce, 1993) for Functional Process Improvement within the Department of Defense.

The *disciplined*, *process-model-based* approach to collaboration presented here has evolved out of the OBID (Object-Based Integrated Design) project —an Air Force SBIR grant to commercialize technology relevant to concurrent engineering—and related research programs and conducted over the past 12 years (Almy, 1991; Blattner, 1979; Ramanathan, 1993; Ramanathan, 1987). Driven by the process management requirements of organizations, the focus is on modeling different *roles* that must *enact* the discipline represented in a workflow process model. When the model is negotiated, created, and agreed upon by the collaborators, it is used by a process model-driven software assistant to govern the collaborative enactment of workers on the actual projects. Thus, the assistant supports design as a collaborative problem-solving activity involving steps (and decisions) that transform product information from one form to another. Many different types of problem-solving agents, such as workers (engineers), analysis software, expert systems, and a variety of database applications, are also managed by the assistant in order to transform and develop the product information. Further, support is for *long-term* collaboration requiring the global state of the problem-solving and the manipulated information entities to be saved in a database. Management visibility is provided by metrics such as queue times of activities, delays, and other attributes of the global state. Process management as a technology is also positioned here in the context of the information system architectures.

Thus, this chapter reflects an interdisciplinary perspective in developing assistants for workflow process support and management. It also discusses and assesses their tremendous commercial impact and viability. The chapter also includes future process management technology issues that need to be addressed both from an engineering and an information systems perspective.

2. Problems Due to Lack of Process Assistance

In this chapter, problem scenarios arising from lack of support for different types of workflow processes are examined. Though the workflow problems examined here are related to design and manufacturing, similar problems exist due to lack of process support in many professional endeavors. Assistants, which have been developed and utilized in commercial pilots by aerospace industry vendors to support the users and address the problems in each of these scenarios, will be presented later along with measured and significant productivity improvements.

Existing Problems in Interdisciplinary Design and Manufacturing

The first user scenario considered here presents the difficulties encountered when engineers attempt to design blades for multistage jet engine compressors. Currently, design is done collectively by aerodynamics, stress, dynamics, and mechanical engineering specialists using mainframe application programs that have been developed over the years to analyze different design parameters. Typically, each *type of engineer* must make key design decisions (e.g., blade thickness, T/C ratio, attachment design, tilt and lean, etc.) that determine how other engineers use those decisions to set up the parameters and invoke analysis applications. Based on each application function invoked, other key decisions are made or earlier decisions are rejected and design must iterate between the disciplines. Today the design process is hindered by many factors:

- Manually allocating work to engineers and keeping track of the status of each blade design rapidly approaches exponential complexity because of the need to keep track of the decisions associated with each department, each application, each blade number, each compressor stage, and *each* design iteration.
- Difficult for engineers of one specialization to anticipate downstream problems that their design decisions might create for engineers of another specialization. Selecting design alternatives that reconcile downstream problems is especially difficult for a novice engineer.
- Considerable effort is expended in setting up the correct application invocation calls, based on earlier project-specific decisions, and getting into and out of applications.
- Difficult to maintain design histories to gain insight that would help refine the design process. Because of this, key process decisions leading to process improvement are typically lost.
- Lack of knowledge regarding the precise status of design, despite the use of project management tools. While project management tools are used, they often reflect an inaccurate status because they are updated off-line based on information that does not reflect all the exceptions, problems, and the variations from the planned project. These inaccuracies soon lead to major perturbations of the plan.

Due to the problems presented above, the design of a compressor traditionally takes many months.

The next example has to do with assembling a product and occurs further downstream. Most companies find themselves with critical product information missing, misplaced or in some engineer's "private" database. Because a way to improve quality is to assemble a complete set of product information for manufacturing, it is necessary to assemble a complete product information packet (which may, in turn, be stored in a product data management system). This packet can then be provided to manufacturing. The problems in assembling the packet are numerous:

- Time consuming to interact with multiple applications and databases—manufacturing database for open orders, bill-of-materials database for the drawings and process plan, and other databases where specific product information may reside (if it does not reside in the product data management system yet).
- Time consuming to track the exact status for each open order (e.g., which drawings and process plans have been completed for this order). The production planner often needs to make numerous phone calls to determine which order can, in fact, be released based on a complete information packet.
- Time consuming to contact appropriate engineering departments to develop pieces of the information (i.e., CAD drawings, process plans, etc.) and have the information appropriately reviewed.
- Lack of precise status makes it impossible to provide accurate feedback to customer on planned availability of the product.

Even after information is provided to manufacturing, further problems remain:

- It is difficult to track problems with an assembly as they are being addressed by the planning, engineering, and purchasing departments.

While the two sets of problems, discussed above, that typically occur during design and manufacturing are not complete, they do indicate the fundamental lack of assistance for collaborative work. The problems presented above are *further magnified* when an attempt is made to practice integrated product/process design. Tremendous knowledge is required by each type of engineer to *collaborate* and complete every activity so that all the relevant product/process constraints are addressed to minimize errors in all downstream activities performed by other engineers.

Productivity via Assistance: Case Study Results

Productivity improvements achievable via knowledge-based assistance during a complex design process have been demonstrated by an aerospace vendor and are documented below. The automated workflow combined the existing application *software* and the turbine blade design *process* into a *blade design assistant.*

The overall goal of the blade design assistant workflow was to shorten the cycle time. None of the existing applications was altered. The blade design assistant builds the appropriate input parameter file and commands to invoke each of these applications. The engineers do not need to be concerned with the format of this file, only the content (values of the parameters). Where possible, design activities are performed for blade design and to decrease the routine effort expended by the design engineers. This workflow assists the engineers in the design of a blade and its associated attachments.

The four major engineering departments that collaborate are aerodynamics, stress analysis, dynamics, and mechanical design. Within each department, design activities are supported to include component design, data entry, coordination with other engineers, and use of software applications. In addition, dependent activities are prevented from being executed until all requisite information and approvals are available (in order to ensure that engineers are not expending effort on inappropriate activities). This was accomplished using the coordination features of the assistant.

Process flowcharts (models) developed by the expert engineers were converted to a workflow process assistant using the KI Shell described in detail in the next section. The *KI Shell development environment* contains the tools used to represent this workflow. KI Shell rules were also developed to implement analytical code to analyze application output and apply design constraints; to prepare input, submit, monitor status, and retrieve output of application on heterogeneous computers; and to suspend/initiate the workflow process for different specialists based on the design process model. An overview of the workflow between the different engineering departments is shown in Figure 16.1.

The pilot results were documented. The benefits of the blade design workflow were:

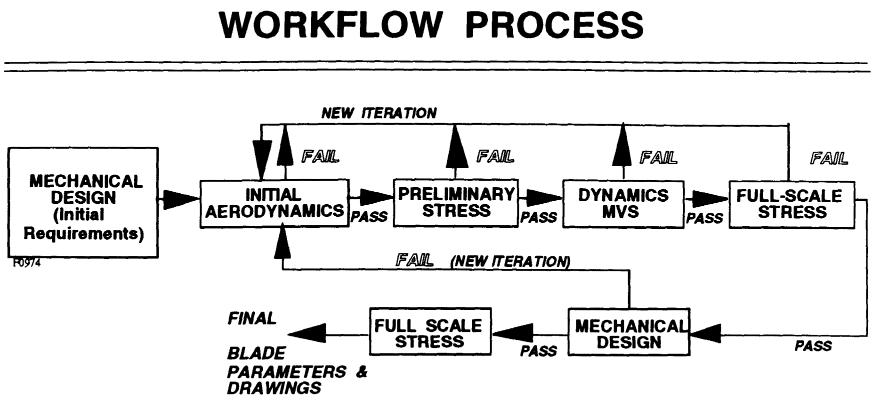


FIGURE 16.1. Overview of the workflow process control flow and interactions between the disciplines (roles) as supported by the Blade Design Assistant.

- Improved design labor hours from 5 to 1.
- Improved design time from 13 to 1.
- Captured process knowledge of experienced engineers.
- Dramatically reduced training for new users.
- Remembered and interpreted design rules consistently each time.
- Engineer-developed flowcharts converted into process management software.
- Standardized design process and reduced mistakes.
- Standardized configuration of the resulting product component.

The engineering features of the workflow process mentioned by the engineers were:

- Interface between various disciplines.
- Suspended and resumed workflow activities.
- Tied together current design applications.
- Captured design rules.
- Captured expert process knowledge.
- Interface with applications without any modifications required.
- Operated applications on existing mainframe platforms from workstations.

What Did the Workflow Process Assistant Do?

More generally, the above process benefits arose from supporting the following key features of the workflow process assistant:

- Structuring (or modeling) of the activities of the process that must be presented to each discipline and presenting the users with only those activities ready to be worked on.
- Actively guiding users by dynamically controlling creation of new activities and determining when activities must be performed by different groups of people for each project, each component, and each design iteration.
- Prohibiting activities from being performed, if indeed this is the policy of the company, until information/decisions generated at earlier activities are correctly completed and reviewed.
- Allowing the easy exchange of decisions made in earlier activities (possibly by other disciplines) and ensuring that these decisions are correctly made so that each subsequent activity can also be correctly performed.
- Automating the setting up of data and invocation of the application function as required by the policy of the company. Also, automating dispersal of application output to specified destinations—other applications, users, or processes.
- Providing decision support (based on expert knowledge) for completing the activity correctly.
- Providing status information on activities queued, completed, being worked on, and waiting on an event.

3. Workflow Assistance Requirements and Concepts

Based on the design and manufacturing user scenarios such as the ones discussed in Chapter 2, this chapter abstracts the requirements of an information system to support fundamentally collaborative approaches like concurrent engineering. These requirements are reviewed both from the perspective of the user and the implementor of the information system.

User Questions

Without collaborative process management software, the user must rely on experience and judgment to resolve the following types of questions broadly classified into three categories:

Collaboration Related:

- What is the most critical task within the process for me to work on? What choices do I have at this point?
- What process/product constraints must I satisfy at this point to reduce downstream iterations?
- What/when/how do I coordinate with other project members/departments?
- What decisions made previously affect my work?
- How do I document and archive my product and process decisions to enable process improvements in the future?

Information Related:

- What application function(s) should I use and how do I use them effectively?
- How do I transform data to run an application?
- What are the useful and relevant views of data at this step?

Management Related:

- What is the overall status of the projects?
- What process decisions can be improved in the future?

Collaboration Issues

The collaborative questions above arise due to the interdisciplinary nature and enterprise-*orientation* of the required problem-solving. *Individual* problem-solving must take place in the context of *disciplined group* problem-solving. This makes it important for the assistant to monitor the status of activities of different types of users and to enforce the policies and protocols of the enterprise while actively guiding the collaborators through the discipline reflected in the workflow process model. Thus, collaboration requires that "responsibility" for specific activities must be "*ascribed to*" and "*enacted by*" individuals in specific departments. The concept of *roles* is required to group the activities that are the responsibility of a given type of worker. A variety of different protocols might be needed to support interactions between different departments. Examples of *protocols* between roles include reviewing, circulation, waiting for completion of activities done by other roles, and providing deliverables to "customer" roles.

Because of the detailed nature of the collaboration support required, the workflow discipline must be *modeled precisely* enough to be negotiated among the different types of participating engineers. Thus, a simple yet complete notation is necessary to acquire the knowledge. The notation allows a specific approach—or *workflow*—to be recorded and negotiated up front by all the participating roles. This is when group problem-solving takes place. From a workflow modeling and enaction point of view, the collaborative process knowledge characterized above consists of

- Knowledge of roles that participate in the process and the protocols between roles.
- Knowledge of the process (structured collection of activities) enacted by each role in an organization. This includes when and under what conditions one role creates activities for another role to enact.
- Knowledge of how to perform each activity correctly so that downstream activities can "count on" it. This includes the correct use of data, previous decisions, and applications to perform the activity.

Thus, the objective of the modeling approach presented here is to support the acquisition of these types of knowledge.

Information Issues

Information, necessary to address the above questions, is large in volume, broad in scope, evolving continuously, and must usually be obtained from different *existing* sources (Figure 16.2). As a consequence, the assistant must exploit existing systems as shown in Figure 16.3.

Depending on the *evolution* of knowledge pertaining to a particular phase of problem-solving, the problem-solving may be distributed among the assistant, analytic, and database applications; expert systems; and the *user* in various ways. The varying role of the end user makes it important for the assistant to have the explanation capability in a system-dominated problem-solving activity, plus a good, precise understanding of the nature of the user's problem-solving if the system's role is one of decision support. The involvement of numerous existing applications brings in user-interface and intelligent-application-support issues such as the invocability of applications from the interface, hiding application-dependent invocation details such as providing the right data, and reasoning about the success and failure of activities based on the applications output.

The assistant software must also work with existing applications via clearly defined interfaces. Conceptually, these interfaces—called PAs or *Pro-*

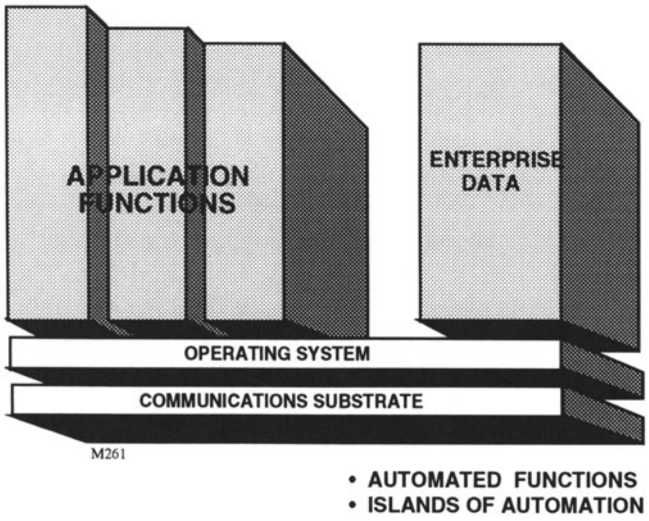


FIGURE 16.2. Today: Information systems view of the problem. Information systems consist of monolithic applications (with embedded process support) and databases that are not integrated to provide decision support for the collaborative process involving people, applications, and data.

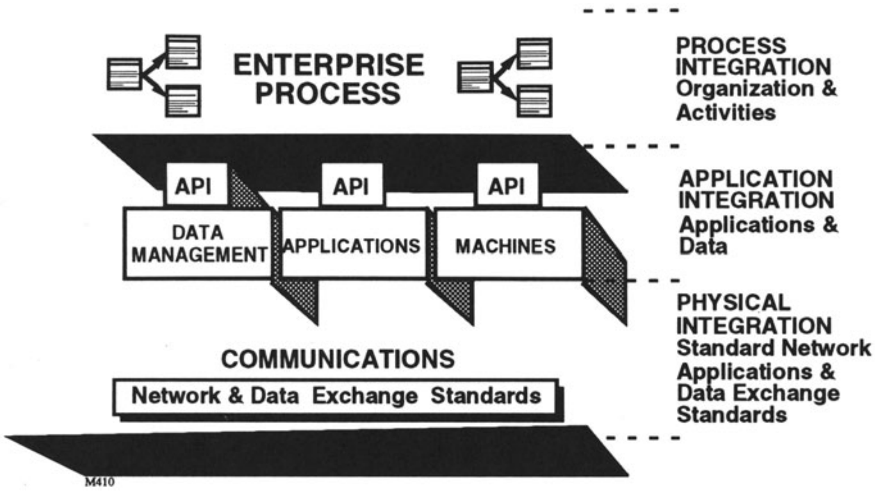

ENTERPRISE PROCESS
PROCESS INTEGRATION Organization & Activities
API
API
API
DATA MANAGEMENT
APPLICATIONS
MACHINES
APPLICATION INTEGRATION Applications & Data
COMMUNICATIONS
Network & Data Exchange Standards
PHYSICAL INTEGRATION Standard Network Applications & Data Exchange Standards
M410

FIGURE 16.3. Future (compare with Figure 16.2): Enterprise process management to provide decision support and reduce level of required training, effective use of applications and data, and coordinate between roles in the organization.

grammed Adaptors—define how the application is invoked using its API (*a*pplication *p*rogram *i*nterface), and how the application's functions are used within the process input and output. Furthermore, the PA should be invokable from the process management software. Existing and new applications can then be "wrapped" in process management software, as illustrated in Figure 16.3. Not only are benefits of improved information flow derived from this "wrapping," but also facilitation of orderly and cost-effective migrations from existing application programs to new re-architected applications.

Workflow Process Management Issues

To facilitate process management and process improvement, the exact status of the process as well as the decisions made during a process must be visible to management. Considerable research must be done in this area and the issues are discussed in the concluding section.

Assistant Architecture Issues

The *long-term nature* of collaborative problem-solving creates the need for a persistent process database for storing the *state* of the problem-solving and its byproducts along with the *relationships* between the objects involved. In a collaborative environment, shared information can become obsolete very soon, if not continually updated in the common process server. Also, assistance often involves shallow inferencing on large amounts of inhomogeneous information that is continually updated. The volume of information entailed and the collaborative nature of problem-solving requires problem-solving to be intertwined with browsing and use of already-created process and product information. This requires the assistant to have an open architecture to interface with existing applications at any point during the enaction.

4. Modeling and Enacting a Concurrent Engineering Workflow Process

The KI Shell™ is a workflow process "shell" that has two components as shown in Figure 16.4. One component being the *development KI Shell*, which is used to edit a workflow model. The other component is the *runtime KI Shell* which is designed to assist the users to *enact* the modeled workflow.

Within the development KI Shell, there are different integrated model editors (like the *Workflow* and the *Frames Editor* in Figure 16.4) which allow workflow objects (see Figure 16.6) to be edited and stored in an SQL database. The use of the database as the process server provides concurrency control and persistence. The graphical workflow editor allows the roles to be created and the overall activity structure to be determined. The Frames Editor allows rules (triggers associated with the objects) and attributes to be associated with the activities. Another tool, RuleWriter (Figure 16.8), allows

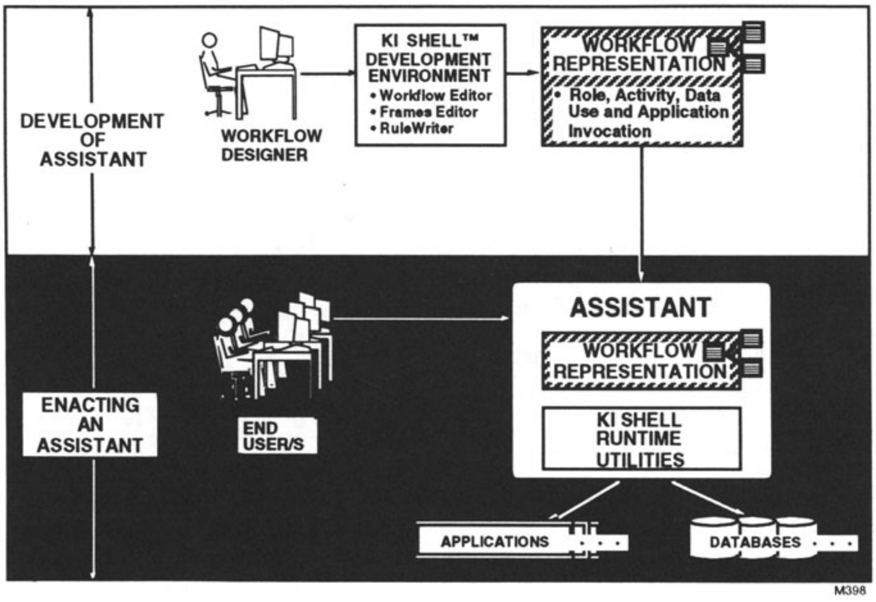


FIGURE 16.4. KI Shell components for workflow process modeling and enaction.

rules to be expanded to C procedures that invoke applications and use SIL (System Integration Library) calls. The SIL is a comprehensive library of reusable operations—including create, delete, update, etc.—on the workflow objects.

The *runtime KI Shell* is a collection of generic utilities designed to display and interpret workflow objects and, thus, *enact* a workflow representation or model created by the KI Shell development environment. The workflow model with its interfaces to applications and KI Shell runtime utilities is called an *assistant*.

An assistant, when enacted by the user, presents the subprocess instances awaiting execution for each role, maintains the state of execution for each subprocess, controls the use of multiple applications, and provides decision support for completing the activities of the process. Thus, an assistant provides active workflow process support and uses applications consistent with the information system's perspective in Figure 16.3.

In order to enforce process discipline during enaction, the KI Shell runtime monitor has *detailed control* over the activities of the users as they invoke applications, perform activities, and modify data objects. (Note, however, that no control must be exerted for interactive applications.) The detailed control is achieved by building the monitor as an intermediary program between the user, the applications, and the KI Shell's own object-based database. By having this control, the assistant can enforce a discipline by utilizing the knowledge represented in the workflow. This is illustrated in

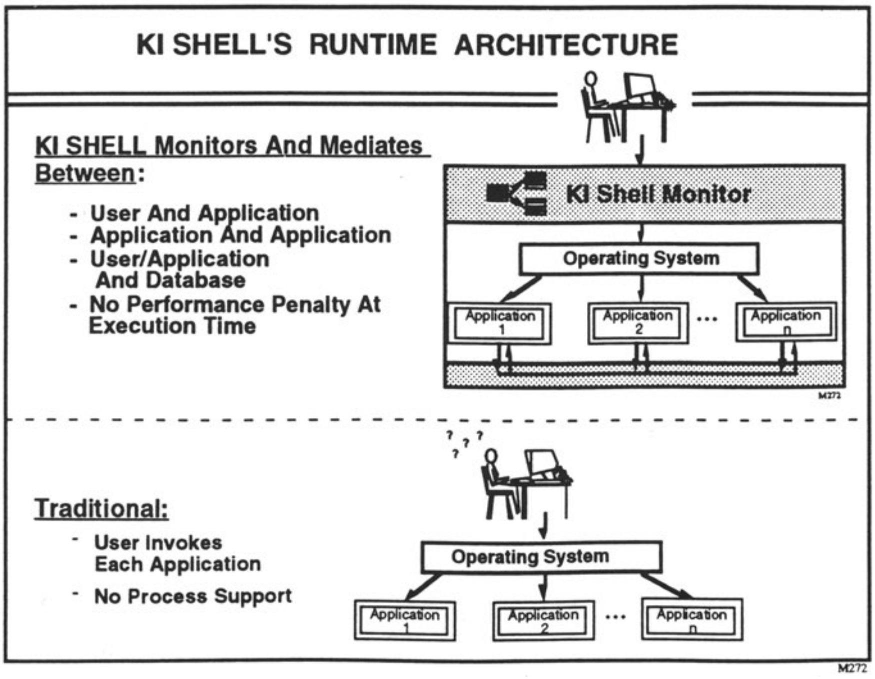

KI SHELL'S RUNTIME ARCHITECTURE
KI SHELL Monitors And Mediates Between:
- User And Application
- Application And Application
- User/Application And Database
- No Performance Penalty At Execution Time
KI Shell Monitor
Operating System
Application 1
Application 2
Application n
M272
Traditional:
- User Invokes Each Application
- No Process Support
Operating System
Application 1
Application 2
Application n
M272

FIGURE 16.5. KI Shell's runtime architecture is designed for workflow process management.

Figure 16.5 by comparing the workflow process support to the traditional approach of interacting with the operating system.

Finally, the KI Shell runtime monitors the status of activities and does not have an inference engine typical of expert system "shells." This unique *run-time* characteristic of the integration-oriented KI Shell sets it apart from other expert system shells.

Modeling and Implementation

Two basic steps—acquiring process knowledge and implementing the assistant—are involved when using the KI Shell Development Environment. The final step is enacting the assistant, which is the workflow model interpreted by the KI Shell runtime. Each of these three steps is described in detail below using a specific application example that reflects concurrent engineering concepts.

Step 1—Acquire Expert Process Knowledge

The objective of this step is to develop the streamlined workflow process model (Figure 16.6). This requires significant participation from the eventual users of the assistant.

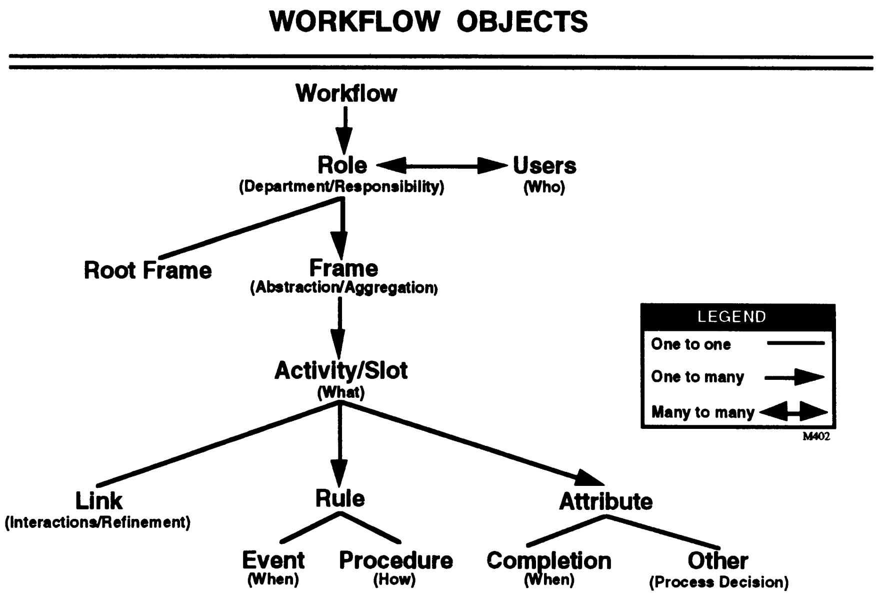


FIGURE 16.6. Workflow process object classes.

The real issue at this point is to negotiate between the interdisciplinary users of the system and develop a good workflow process model based on a knowledge of how experts perform their activities and collaborate with each other.

Modeling Concepts

Many modeling approaches exist (Curtis, 1992; Hars and Scheer, 1992; Mayer, 1989; Ross, 1977; Scheer, 1992). Since IDEF (Buffum, 1981) is used extensively within the DOD design and manufacturing efforts, it is also used as a starting point for discussion here. IDEF consists of several complementary notations for creating models. Of relevance here is IDEF0, which is designed to identify activities in an enterprise. This modeling notation helps create both an activity decomposition and, for each activity, the input-constraint-output-mechanisms (as in Figure 16.7a). The other notation—IDEF1X—uses the data requirements, identified by an IDEF0 model, to create an entity-relationship model for the enterprise data model. Figure 16.7b, when compared with Figure 16.7a, illustrates the concept of enaction.

Activity decomposition identifies the functions and the external view of data used within the activities and is a valuable starting point for workflow. Activity decomposition does not explicitly model the *flow of control* between activities and the organizational aspect of *responsibility* based on "who" enacts "which" specific activity and in "what order." The modeling nota-

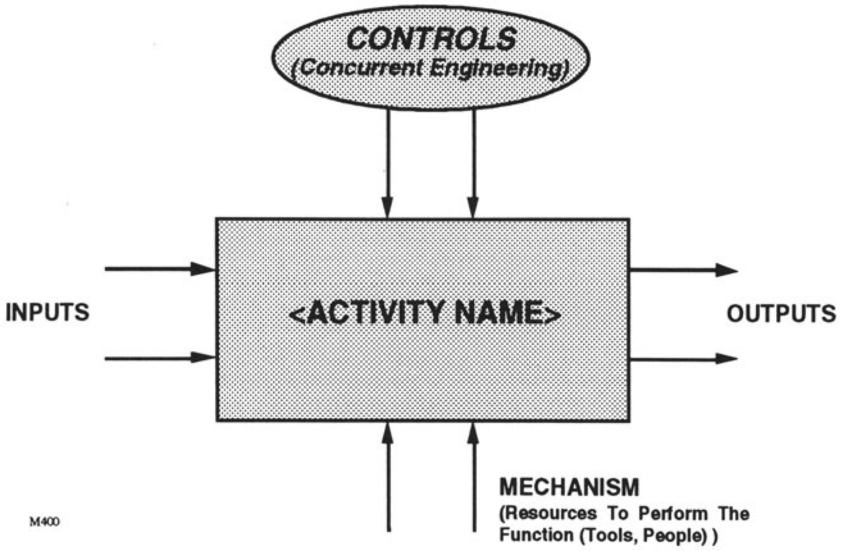


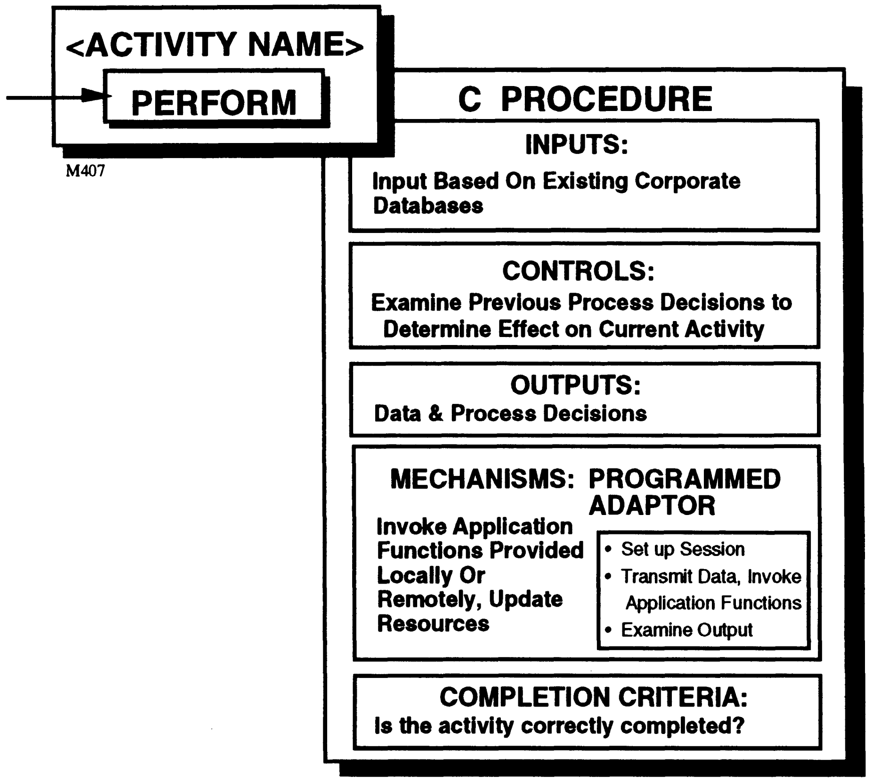
FIGURE 16.7a. Notation for single IDEF activity identifying INPUT, CONTROLS, OUTPUT, and MECHANISM. Note that the controls (data from previous activities) implement concurrent engineering philosophy of performing the activity in the correct context.

tions underlying the KI Shell include both the activity decomposition and control flow views. These complementary views are presented graphically in Figures 16.7c, d, e, and f for the example selected for discussion here.

Before examining the different views of a workflow, it is first necessary to discuss the underlying workflow model. Figure 16.6 presents an overview of the KI Shell workflow objects that must be created to represent process knowledge. These objects constitute the conical (or composite) form, based on which several different graphical views (or perspectives) of the standard form can be projected and used to model different perspectives of the workflow, as illustrated in the application discussed below. The notation details of some of the graphical views are in Tables 16.2–16.4 provided in the Appendix.

At the heart of every *interdisciplinary* workflow effort are the fundamental concepts of role and activities. Each role enacts activities related by the role's perspective of the flow of work. The activities within a role are further organized into a hierarchy of frames. Each frame aggregates conceptually related activities that hide details at the lower levels. Both activities and frames also have associated semantics reflected in the rules (methods), attributes, and links. In KI Shell, a use of frames is to control the discipline (the "when") by which activities are performed.

Each activity can have attributes, links, and rules that provide a way of



<ACTIVITY NAME>
PERFORM
M407
C PROCEDURE
INPUTS:
Input Based On Existing Corporate Databases
CONTROLS:
Examine Previous Process Decisions to Determine Effect on Current Activity
OUTPUTS:
Data & Process Decisions
MECHANISMS: PROGRAMMED ADAPTOR
Invoke Application Functions Provided Locally Or Remotely, Update Resources
• Set up Session
• Transmit Data, Invoke Application Functions
• Examine Output
COMPLETION CRITERIA:
Is the activity correctly completed?

FIGURE 16.7b. Enaction of an activity (initiated by clicking on the Perform button of a displayed activity object) actually causes a C procedure containing the "how" logic to execute. Typical logic programmed with this procedure is illustrated.

implementing the support when the activity is enacted (i.e., when the "perform" button is clicked). Conceptually, each activity must meet certain objectives and satisfy the completion criteria for the activity, so that the next activity enacted can assume the correct completion of previous activities.

The rules (triggers or methods), in turn, consist of an event + procedure (in the C programming language). The typical events monitored by the KI Shell runtime are related to the process. Examples of events are enacting an activity or entering and exiting a frame. The procedure specifies "how" an activity is enacted when the event occurs within the context of the workflow structure. The links declaratively specify different relationships between workflow objects. "Completion" is an example of a system-maintained attribute that allows the KI Shell runtime to track completed activities.

Enaction ensures that a modeled activity is correctly completed. The input-constraint-output-mechanism defined in a model is actually manipulated by the C procedure.

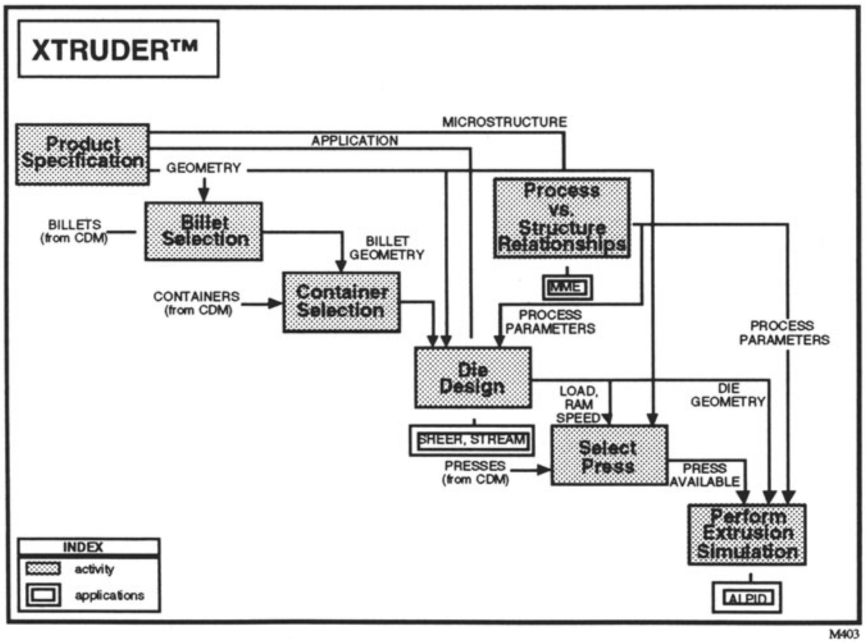


FIGURE 16.7c. Example of an activity model for die design using the IDEF0 Notation summarized in Table 16.2. CDM stands for Corporate Data Management, and MME, Shear, Stream, and ALPID are applications developed by the Air Force for materials modeling and simulation.

Application of Modeling Concepts

As an example to illustrate the application of the modeling and enaction concepts, this section uses the die design process. This application was developed under an SBIR grant supported by the Air Force Manufacturing Science Program.

In a traditional extrusion plant environment, the interactions between different roles are done on an "over the wall" basis. With the die design assistant—XTRUDER—the Production Planner is able to specify and schedule Tooling and Equipment in the context of Die Design and Process Metallurgy decisions (constraints) made by the die designer and the metallurgist.

An overview of the related models resulting from the modeling step for die design is illustrated in Figures 16.7c, d, e, and f. While the fundamental concepts of roles and activities that are illustrated here have been introduced before, these figures also illustrate how the more detailed notation in Tables 16.2–16.4 have been applied to create the different views of a die design workflow process model. These graphical notations incorporate the use of activities (rectangular boxes) and relationships (represented by different

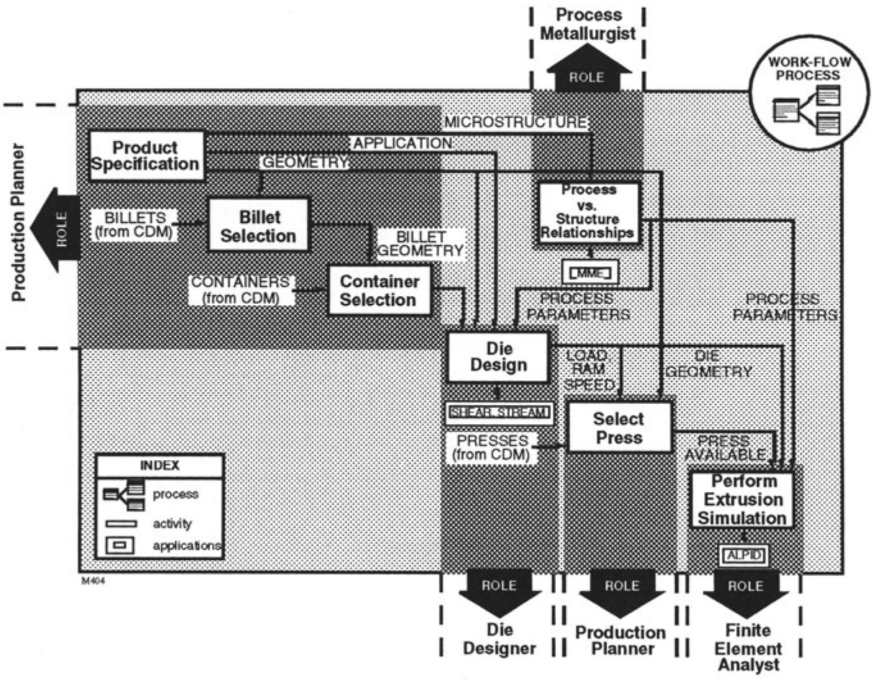


FIGURE 16.7d. Example of an activity model for die design with activities grouped by the role name (e.g. Production Planner) reflecting the discipline that is responsible for completing the activities.

links). As indicated in these tables, different relationships are the focus of the different views or perspective of the composite or conical view of workflow objects presented in Figure 16.6.

The first model (Figure 16.7c) consists of activities, input (use of corporate data), controls (previous data that constrains the current activity), output (data created during the activity), and mechanisms (use of applications and resources) similar to IDEF0.

In this view of the workflow model two concurrent engineering principles are embodied:

- Do not over-constrain the implementation during design.
- Address all necessary process constraints early on to reduce iterations.

For example, the simulation activity occurs only after the press selection is made, thus ensuring that the correct parameters are used in the simulation and no iterations will be necessary. Throughout the workflow, detailed design decisions are made based on the available components and tooling.

The next model (Figure 16.7d) illustrates the perspective of the different

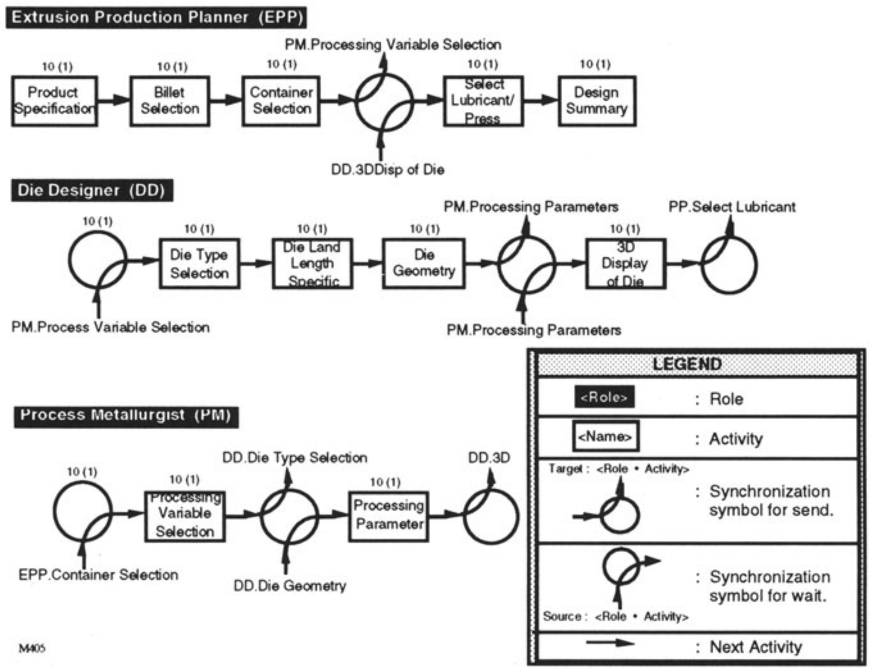


FIGURE 16.7e. Control flow, role, and synchronization view depicting the flow of work between the departments, using notation in Table 16.4.

disciplines (or roles performing the activities). This can also be obtained by identifying the mechanisms for each activity in an IDEF0 model. However, in this notation, the flow of control during collaboration is not explicitly visible.

As depicted in the model in Figure 16.7e, each role consists of many activities sequenced appropriately. Given an activity is a fundamental primitive of any workflow, it is imperative that each activity in a workflow be enacted to add value (i.e., meet the *completion criteria*) to ensure that the overall workflow—the sum of activities—eliminates unnecessary iterations. Further, by identifying the "AS-IS" and "TO-BE" times with each activity, one can estimate the actual "cycle" time.

The final view is the activity decomposition illustrated in Figure 16.7f. In this view, the framing concept is used to group activities at the same level of abstraction.

In summary, the two concurrent engineering principles are modeled by the controlling arrows of the model in Figure 16.7c, the collaboration protocols between roles as modeled in Figure 16.7e, and the user's view of the activity network in Figure 16.7f. These different views are all important for defining the structure of the workflow.

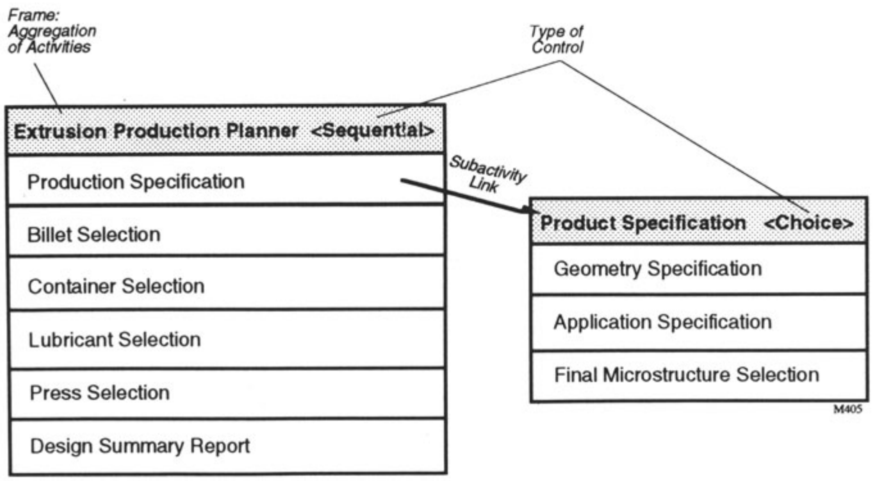
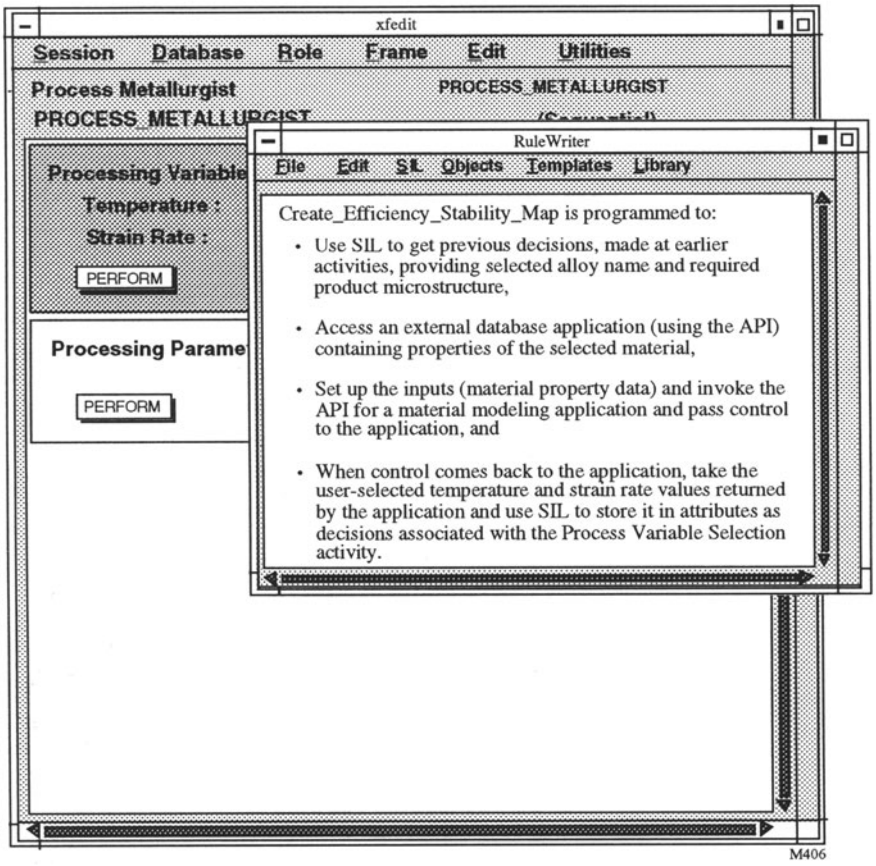


FIGURE 16.7f. Activity hierarchy with an activity refined to subactivities: The activities in a frame are typically grouped to reflect a discipline and similar level of abstraction. Subactivity link allows detailed refinement of an activity. This view is developed using the notation in Table 16.3 (see Appendix).

Creating the Model Using the KI Shell

KI Shell Development tools create a machine readable version of the workflow objects (Figure 16.6) based on the models created in Step 1. The workflow editor component of the KI Shell is used by the workflow designer to define the structure of an arbitrarily complex process. The title bar of the editor (see Figure 16.8) allows the user to select/create roles (a collection of related frames) and frames (or group of related activities), and edit these objects. The workflow designer can then edit the activities that belong in that frame. Figure 16.8 also illustrates the editing of a sequential frame called "Process Metallurgist" composed of activities called "Processing Variable Selection" and "Processing Variation." Attributes associated with the activity "Processing Variable Selection" are "Temperature" and "Strain Rate." These attributes hold the decisions made during the enaction of the workflow for a specific project. By linking frames, the editor creates the subprocess for a role.

The editor also provides the ability to express sequential, choice, conditional and repetitive execution of the activities in a frame, based on the state of the process. In the workstation environment, multithreaded options for executing activities are available. This allows the user to follow more than one path through a subprocess. With each activity, the editor allows the user to associate rules that assist the user in performing the activity. The rules (event/procedure pairs) can be triggered based on user interaction (pressing



Create_Efficiency_Stability_Map is programmed to:

- Use SIL to get previous decisions, made at earlier activities, providing selected alloy name and required product microstructure,
- Access an external database application (using the API) containing properties of the selected material,
- Set up the inputs (material property data) and invoke the API for a material modeling application and pass control to the application, and
- When control comes back to the application, take the user-selected temperature and strain rate values returned by the application and use SIL to store it in attributes as decisions associated with the Process Variable Selection activity.

FIGURE 16.8. User interface of the Development Environment illustrating the Frames Editor and the RuleWriter.

a function key or mouse click) or by the modification of the attributes by other rules. A variety of suitable events (such as "perform," "pre-modify," "post-modify," "enter frame," etc.) are also supported.

Step 2—Process Model Enaction

The implementation of enaction is illustrated Figure 16.9. When the user clicks on the "perform button," the logic in the box—programmed in a procedural programming language like C—is executed. Thus, the activity model provides the "what" context within which the procedural code—the "how"—executes.

Once a workflow structure has been defined, the procedures (e.g., the

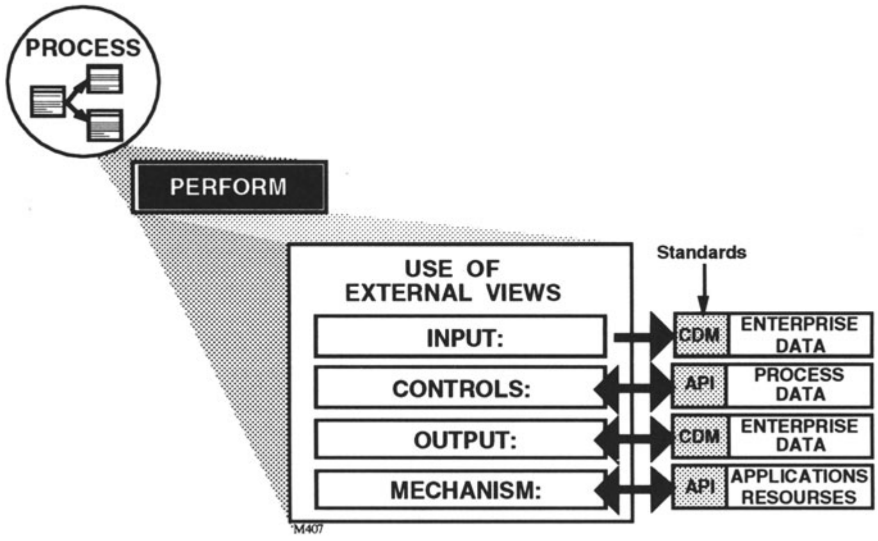


FIGURE 16.9. Enaction facilitates the correct use of information system components and their standard interfaces in the context of the activity and the role.

perform procedure) associated with the rules of each activity can be programmed using the RuleWriter component of the KI Shell. As illustrated in Figure 16.8, the RuleWriter title bar allows the programmer to edit the file containing the source code for the C procedures of rules. The declarations (for include files, etc.) are automatically inserted by the RuleWriter. C language templates are provided by the RuleWriter to eliminate syntax errors.

The RuleWriter component of the KI Shell also uses menus to assist the user in developing procedures using "SIL" for programming productivity. The KI Shell System Integration Library ("SIL" in the title bar of Figure 16.8) is a collection of functions provided for use within the rules. Within the rules/procedures, a variety of associations can also be made. Based on decisions made in earlier activities, appropriate data can be set up for user or application use. Applications can also be invoked automatically. Upon completing the execution, data generated by the application invocation can be distributed as specified.

An example is "Perform + Create_Efficiency_Stability_Map" associated with "Processing Variable Selection" activity in Figure 16.8. In this step, Create_Efficiency_Stability_Map is programmed in C to:

- use SIL to get previous decisions made at earlier activities, which provide selected alloy name and required product microstructure,

- access an external database application (using the API) containing properties of the selected material,
- set up the inputs (material property data) and invoke the API for a material modeling application and pass control to the application, and
- when control comes back to the application, take the user-selected temperature and strain rate values returned by the application, and use SIL to store them in attributes as decisions associated with the Processing Variable Selection activity.

A more general overview of the C procedure is given in Figure 16.9. KI Shell provides the ability to use standard interfaces to information systems correctly from the perspective of an activity and the role within a process.

Thus, in this step the procedures, as specified by the modeler in Step 1, for each rule are programmed using the extensive SIL library provided by the KI Shell. Often an activity has to wait on a decision by an activity of another role before proceeding. This is implemented by one of the SIL functions ("wait for signal") invoked in the perform procedure of the current activity. The "send signal" SIL function has to be executed by the appropriate activity of another role before a waiting activity can proceed. At this point, a workflow is completely implemented.

Step 3—Enacting a KI Shell Assistant

This requires linking the KI Shell Runtime with the workflow rules and *P*rogrammed *A*daptor calls (PAs) to form an assistant that can be enacted by the user. In the following, the users' view of enaction for the specific example is discussed.

Users' View of Enaction

The XTRUDER assistant coordinates the functions of three roles:

- Extrusion Production Planner,
- Die Designer, and
- Process Metallurgist.

The XTRUDER assistant actively presents the work instances that enable the correct sequence of activities and role interactions to be enacted. The KI Shell runtime interprets the XTRUDER workflow to present the XTRUDER roles on the screen as in Figure 16.10.

When a role and a role instance are selected, the user is presented the frame last executed. Figure 16.10 illustrates the Die Designer frame with the completed activities and the next executable activity.

Collectively, the three roles complete the activities in the process as follows. The first activity is "Product Specification." When executed by the Production Planner, it provides the geometry, application, and microstruc-

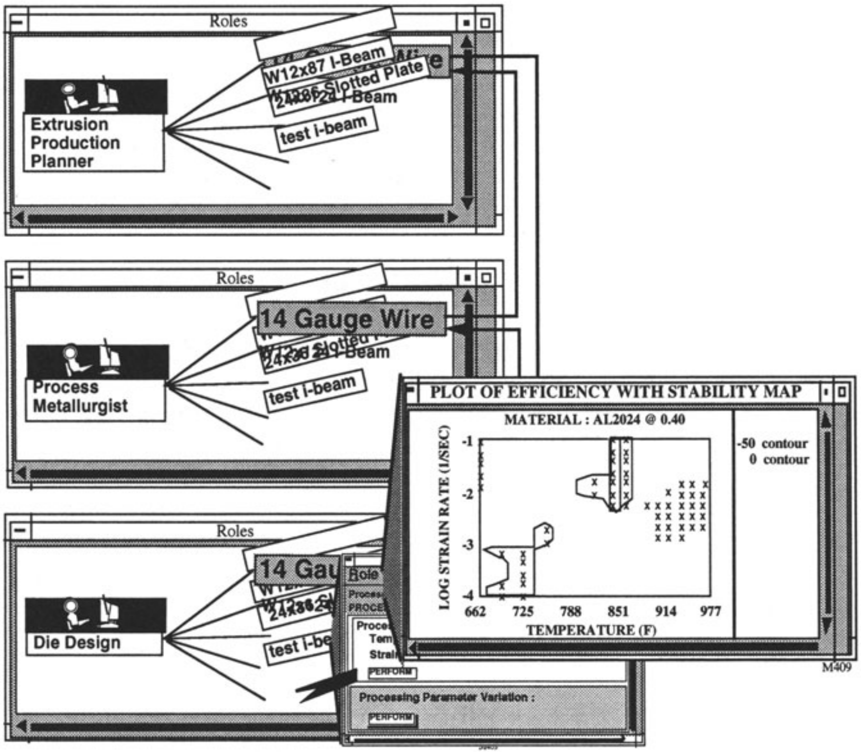


FIGURE 16.10. When authorized performers of different roles log on, they see the work queues associated with their roles. For the Die Designer, the user interface of the assistant when performing the Processing Variable Selection action is shown. Users can interact with the invoked application in another window. Perform rule name and hidden attributes are not displayed. The assistant presents activities and visible attributes as illustrated in the first activity of this frame. Activities are colored to reflect their status. When this subprocess completes control goes to the next role and process as specified.

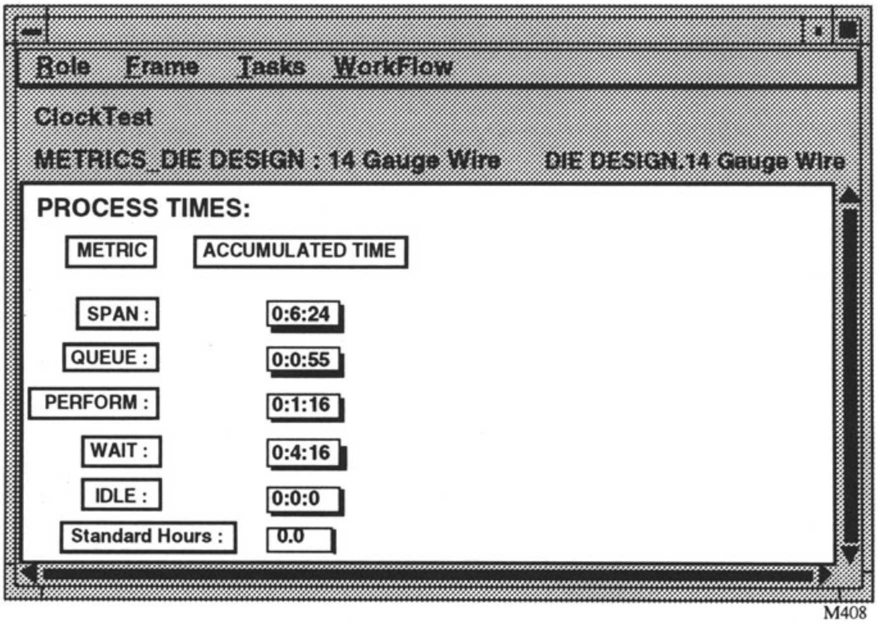
ture values for constraining decisions made at later activities by the other roles. For example, the geometry constrains the candidate billets selected from a manufacturing database during the "Billet Selection" activity. The actual geometry of the billet selected by the engineer becomes the decision value of the attribute associated with "Billet Selection." This value, in turn, constrains the next activity and so on. "Die Design" is an example of an activity performed by the Die Designer role that invokes an application for rough design prior to detailed finite element simulation during the "Perform Extrusion Simulation" activity. This activity in turn is performed with the assurance that the appropriate press guaranteeing the appropriate processing conditions is available.

By using the assistant, engineers are able to complete complex designs correctly the first time, and at a fraction of the cost. Thus, XTRUDER integrates Die Design application programs, a Materials Behavior database, and Process simulation applications that quantify the manufacturing requirements for the Production Planner.

The KI Shell also provides automatic status tracking and inter-role dependencies of work in progress, and provides graphical presentation of status as in Figure 16.10. Performance metrics are also captured automatically for KI Shell developed Assistants, providing queue time, span time, and wait time for each role/role function.

A variety of different metrics can be obtained during enaction. For example, the time taken for executing each activity on an execution path can actually be measured. When the workflow manager enacts a model, it can obtain the following kinds of data:

Queue time:	The length of time a subprocess waits before a worker selects it for execution.
Span time:	The length of time it took to execute actual activities in a role.



ClockTest

METRICS_DIE DESIGN : 14 Gauge Wire DIE DESIGN.14 Gauge Wire

PROCESS TIMES:

METRIC	ACCUMULATED TIME
SPAN :	0:6:24
QUEUE :	0:0:55
PERFORM :	0:1:16
WAIT :	0:4:16
IDLE :	0:0:0
Standard Hours :	0.0

FIGURE 16.11. Process enaction provides a unique leverage—Process Metrics: When an activity is enacted, time stamps can be automatically obtained to measure span time, queue time, and wait time. These metrics, along with others, can provide a measure of the performance of the process.

Rejections:	The number of times a design was rejected.
Resource:	Type of resource used and actual amount.

Figure 16.11 illustrates the time metrics for the Die Designer process while producing the 14-gauge wire. The overall status of the different process instances producing—14-gauge wire, W12 × 124 I-beam, W12 × 87 I-beam, 24 × 36 slotted plate, text I-beam—is readily visible to a manager by looking at the queues (Figure 16.10), color coded to reflect status such as "awaiting processing," "completed," "waiting for another role to complete," etc.

5. Workflow Process-Based Information System Architecture

Before proceeding to a discussion of future research issues, the process management technology is first positioned in the context of information system architectures. The three-schema architecture is widely accepted as a starting point for separating the concerns of

- the *users* of the information systems,
- international *standard activities*, and
- *implementors* of products that work with the standards.

Thus, the positioning begins with this architecture.

Three-Schema Architecture

Building upon the early three-schema architecture piloted by the IISS project (WRDC, 1990), most efforts—like CIMOSA (Computer Integrated Manufacturing–Open Systems Architecture)—propose the use of the "Three Schema Architecture" (AMICE, 1989; Althoff, 1990) to separate the external (the specific enterprise's view), conceptual (the standard view across enterprises), and internal (implementation view) of *functions*, *information*, *resource*, and *organization*. The external views of these four elements are combined and *used* by the collection of activities structured to support the roles and the organization of the enterprise. This is illustrated in Figure 16.9.

Within the context of the standards-based information architecture, an information system for assisting disciplined concurrent engineering can now be developed as follows:

- *Analyst*: The expert responsible for specifying the "TO-BE" models of the workflow and creating initial logical prototypes using standards (conceptual models of function, information, resource, and organization).
- *Deployment Engineer*: Responsible for installing the logical "TO-BE" workflows, customizing the *logical assistants* to access enterprise-specific

data using the three-schema architecture, and augmenting the "TO-BE" workflows with enterprise-specific policies. This will create a *deployed assistant*.
- *Manufacturing Engineer*: Enacts the *deployed assistant*.

This chapter has presented an approach where, beginning from a basis of standards and modeling notations to describe CIM system behavior (the generic constructs), useful models can be created for industry segments. These models describe the specific "TO-BE" CIM system behavior. Once developed, these models can be deployed into an industry using a software platform or "shell" for model *enaction* and integration. This reduces the cost of developing information systems by:

- Providing the ability to generalize and standardize CIM process support provided to an industry segment, and reduce custom software developed.
- Permitting the cost of development of a "TO-BE" model to be amortized over the industry segment.
- Associating with model enaction benefits that are clearly identified and can be provided for entire industry segments.

Why is the above scenario for CIM system development made possible by process management? To understand this, we must consider the historical perspective. Since the beginning of the information systems age, application systems tied together three elements:

Data: Includes data definition and management.
Application Functions: Operate on data and automate certain activities.
Workflow Process: Steps by which users are guided to perform work activities and decisions.

Problems arose as a result of the duplication of data definition and management logic within each application. Consequently, data modeling (Chen, 1976; Codd, 1979; Navathe, 1992) and management systems were developed during the eighties to eliminate duplication of data, avoid describing the same data in different ways, and provide common reusable data management software to all applications. Thus, data modeling and management technology allowed the separation of data handling logic from applications. This, in turn, allowed the independent management and evolution of data and applications. It has also facilitated the development of draft standards (IRDS, 1991).

Figure 16.2 illustrates the separation of data and applications in current information systems. Because the application functions are so isolated, they still do not support the actual process of doing work. Therefore, the users continue to face process-related questions, which today they must either address manually or by applications that incorporate a workflow that often does not meet their workflow requirements.

Workflow management represents a technology development of the nineties (Ramanathan, 1992). Its development is analogous to the development and transitioning of databases into common practice. During the past few years, the commercial significance of process management, independent of product data management, has been widely recognized (Seybold, 1992).

By clearly separating the workflow processes from applications (as illustrated in Figure 16.3), workflow technology provides fundamental technical advantages. A workflow process "bridges" across the islands of automation by supporting the end-users with knowledge-based integration. To reiterate, an architecture where there is separation of the process knowledge and its management has several advantages:

- Enterprise workflow process models modified/maintained independent of application and data objects.
- Process used to control when/how applications are used.
- Different process logic can be applied to same data objects/applications—based on the roles and responsibilities. The same applications can be used by different processes (e.g., a finite element analysis application can be used differently for a forging die design process than for an extrusion die design).
- Time-variant logic separated from time-invariant enabling separate evolution of process objects. Process decisions made during activities executed for specific projects (e.g., I-beam vs. wire) vary, but data objects (e.g., billet, process) do not vary.
- Process used to maintain global state.
- Separation makes it easier to associate and maintain use of process-related data.
- Process decisions made during a project can be stored and managed separately from applications. This provides a history of the process for analysis and improvement.
- Process metrics data (e.g., when an activity begins and terminates, how long it took for a subprocess to execute, how often did a subprocess get executed, etc.) can be obtained and presented for process management and process improvement.

6. Future Research Issues

At least two significant research issues arise from the ability to clearly separate the process layer:

- In what way should an integrated tool-set for process improvement be implemented to provide feedback and control during enaction, and
- What are the suitable mechanisms for process reuse?

TABLE 16.1. Types of features to be provided by process technologies.

Process Modeling

- Provide the ability to create different views or perspectives that are all consistent with respect to an underlying canonical model based on activities. Some views are:
- Activity decomposition view—this relationship is commonly modeled in several business process modeling tools.
- Input and Output data created by each activity—also provided by several business process and information engineering modeling tools.
- Resources necessary for each activity—this is also provided for in tools that support IDEF modeling.
- Control flow between activities over time—this is similar to the petri net model used by simulation tools but not typically provided for workflow enaction.
- Roles that define the view of the activities that must be enacted by a type of personnel and is the responsibility of each member of that group.
- Protocols (supplier/customer, reviews, routing, coordination—send and receive—among others) that must be modeled as other relationships between activities of roles.

Process Simulation

- Ability to examine consumption of resources and rate at which queued tasks are processed based on different routings.
- Ability to study other cause and effect relationships.

Process Enaction

- Enforces process enaction discipline between roles in compliance with company policies and practices.
- Delivers necessary information to perform the activity correctly and controls the steps to complete an activity.
- Simplifies determination of which activity to perform next.
- Provides data for decision support and for activity execution.
- Automates required data setup and invokes the application process.
- Disperses application output to specified destinations.
- Accumulates actual process execution data.

Process Management

- View enaction status—the actual time taken to execute a task versus allocated time and the actual versus planned consumption of resources.
- Obtain metrics to answer questions like how often was a process for rejected parts enacted and for which part.
- Process decisions made during enaction provides a history.

Project Management

- Plan consisting of tasks, allocated time and resources.
- Critical path analysis.
- Manufacturing and Resource Planning.
- Comprehensive support for planning and scheduling of resources to meet demand.
- Process reuse.

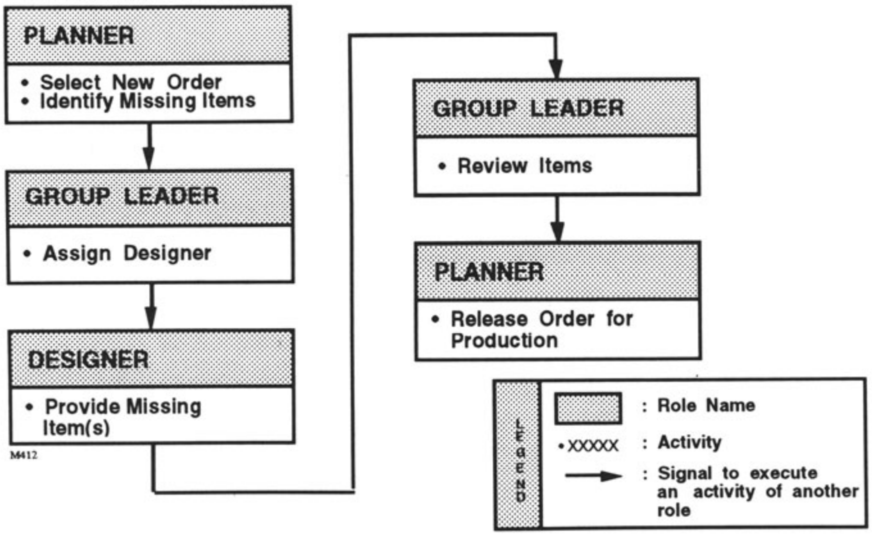


FIGURE 16.12a. Activity and roles: Generic process for collecting product information.

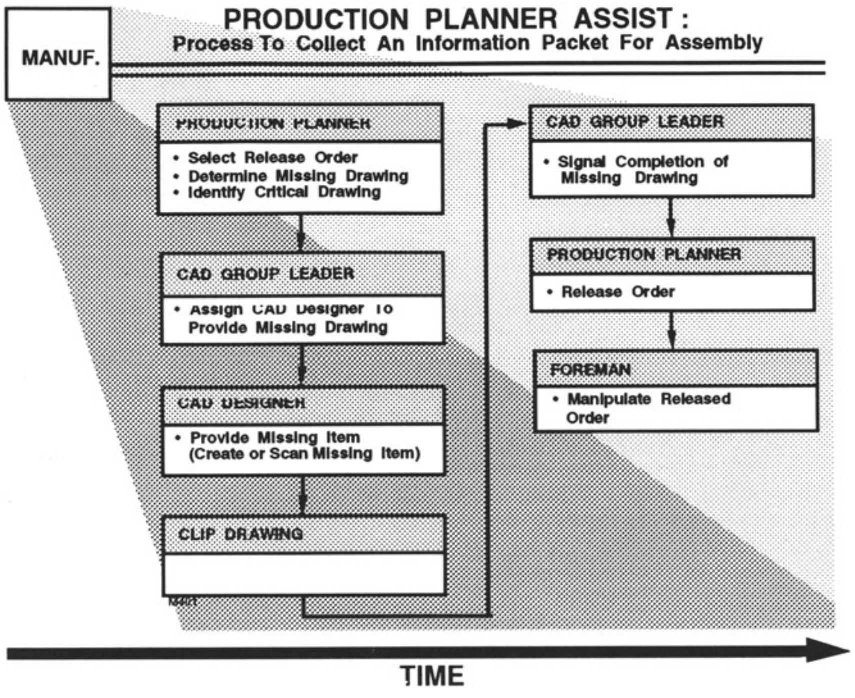

MANUF.
PRODUCTION PLANNER ASSIST :
Process To Collect An Information Packet For Assembly
PRODUCTION PLANNER
• Select Release Order
• Determine Missing Drawing
• Identify Critical Drawing
CAD GROUP LEADER
• Assign CAD Designer To Provide Missing Drawing
CAD DESIGNER
• Provide Missing Item (Create or Scan Missing Item)
CLIP DRAWING
M421
CAD GROUP LEADER
• Signal Completion of Missing Drawing
PRODUCTION PLANNER
• Release Order
FOREMAN
• Manipulate Released Order
TIME

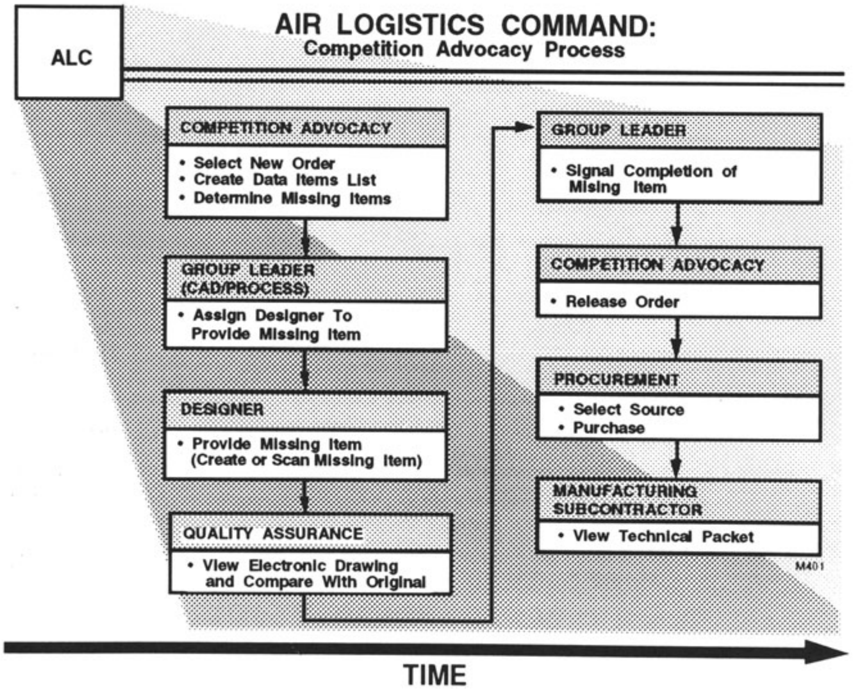
FIGURE 16.12b. Production Planner Assist: Process to collect an information packet for assembly.

Continuous Process Improvement via Process Metrics, Feedback, and Control

Today, process simulation is a way to understand system behavior, and enaction provides accurate process metrics for each project. While aspects of manufacturing and factory-floor, characterized by repetitive processing of tasks that are very similar in nature, can be simulated, many enterprise processes (e.g., conceptual design) cannot be accurately simulated. Thus, actual enaction metrics could play a significant role in identifying areas for improvement. However, the type of process data that might be collected and synthesized for presenting process performance results is not well known.

Table 16.1 lists the features that are provided by *isolated* process technologies today.

To facilitate rapid process improvement, the first step is to integrate process-related tools (STARS, 1991). Several advantages result from an integrated set of process tools. With data integration—all process tools working on the same consistent database—it is possible to provide tools to enable a greater degree of process control:



ALC
AIR LOGISTICS COMMAND:
Competition Advocacy Process
COMPETITION ADVOCACY
• Select New Order
• Create Data Items List
• Determine Missing Items
GROUP LEADER (CAD/PROCESS)
• Assign Designer To Provide Missing Item
DESIGNER
• Provide Missing Item (Create or Scan Missing Item)
QUALITY ASSURANCE
• View Electronic Drawing and Compare With Original
GROUP LEADER
• Signal Completion of Mising Item
COMPETITION ADVOCACY
• Release Order
PROCUREMENT
• Select Source
• Purchase
MANUFACTURING SUBCONTRACTOR
• View Technical Packet
M401
TIME

FIGURE 16.12c. Air Logistics Command: Competition advocacy process.

- Simulation can use actual process enaction data preserved in the enaction database. Therefore, simulation provides more accurate visualization of the actual process characteristics providing a way for dynamic replanning. For example, a process manager could animate (instead of simulate) the manner in which the activity queues change over time; thus, identifying bottlenecks. This facilitates process improvement.
- Process management is based on actual resource usage. As costs are incurred, project management tools provide exact status by providing early information about cost over-runs.
- Global awareness of exact process status allows resources to be redirected to problem areas before the problem compounds to more unmanageable ones.
- By enacting a modeled process and capturing precise model-based metrics in the database, areas for real process improvement are easier to identify.

However, such an integrated toolset does not exist. If it is prototyped, it will facilitate research in the type of enaction data to be collected and synthesized for process improvement.

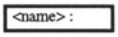
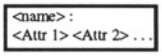
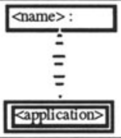
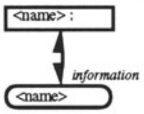
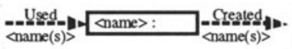
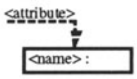
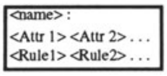
Process Reuse

Patterns rapidly emerge when examining process models. Figures 16.12a, b, and c illustrate variations of the same process designed to accumulate the components of an information packet. Figure 16.12a is a generic, site-independent version. Figures 16.12b and 16.12c illustrate variations for different industries. Over time, a library of logical workflows, like the one illustrated in Figure 16.12a, must be developed to provide a rapid, cost-effective way to provide custom software solutions like the ones in Figures 16.12b and 16.12c to the industry. While it is clear object-oriented technology must be exploited for such reuse, the actual design of objects for reuse is a difficult task (Monarchi, 1992) and must be researched.

Appendix: Modeling Notations

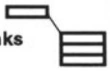
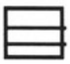
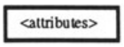
The three tables in this appendix provide notational systems for defining different views of a workflow model.

TABLE 16.2. Modeling notation for use of applications, data, and resources within an activity.

Notation	Definition
Activity 	A basic unit of work in terms of "what" and "how." An activity is a named group of attributes, subobjects, and rules. It also has an associated completion state. The "how" is implemented in a Rule or Procedure with SIL (KI Shell System Integration Library) calls.
Attributes 	Placeholder for values generated when executing an activity. These attributes can hold process decisions that impact which of the later activities are performed and how they are performed. An attribute can also hold names of information objects necessary to perform an activity or bookkeeping data.
Application Use 	External applications (analysis, databases, etc.) invoked to automate aspects of an activity. Applications 'APIs' are used to invoke, retrieve data, or store data.
Information Use 	Information frames that describe external information, resources, costs, and metrics.
Information Use 	Product data (information objects) used or created when performing activities.
Process Decisions 	Previous process decisions (i.e., attributes of earlier activities) that control how the current activity is completed.
Rules 	Rules can also be associated with an activity. The rule contains the logic of how an activity is executed.

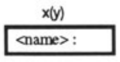
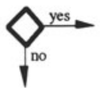
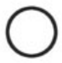
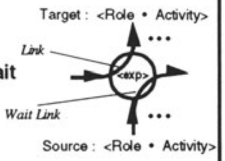
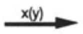
M399

TABLE 16.3. Modeling notation for the activity hierarchy/information hierarchy.

Notation	Definition
Subactivity Links 	Bi-directional links between an activity and a frame.
Frame 	Aggregation of activities structured by a control construct which determines the sequencing of activities. The order of activities can be "sequential," "choice," "if-then," "while," etc.
Information Frame 	Aggregation of attributes that hold information.

M399

TABLE 16.4. Notation for the time view.

Notation	Definition
Activity 	Activity name. Estimated times—x and y—for activity execution, without workflow support and with workflow support, respectively, are used to execute the activity to provide a basis for cycle-time reduction estimates.
Decision Activity/ Next Activity Link 	Decision activity is a special type of activity with the next activity based on the outcomes "yes" or "no."
Synchronization Activity 	Synchronization activity is a special type of activity used to suspend further activities within a role until it receives a signal sent from other activities in other roles.
Send/Wait Link  Target : <Role • Activity> Link <exp> Wait Link Source : <Role • Activity>	Links to send a signal or wait for signal. Could have <Role.{userid.}Activity> when specific names are chosen. <exp> is the expression that specifies which wait links must "fire" before proceeding with the condition by which the activity completes.
Next Activity Link 	Links an activity to another. Estimated times—x and y—for activity execution, without workflow support and with workflow support, respectively, are used to provide a basis for estimating cycle-time reduction. An activity can have many next activities. If an activity is already completed, this link indicates the activity is redone.
Role <Role Name> <Identifier>	A collection of activities performed by a prototypical department/project member. This is an "entry point" into a group of activities in the total activity network. When a worker is assigned to a role, the responsibility of the worker is to perform the grouped collection. The <identifier> reflects the manner in which role instances are given unique identifiers.

M399

References

[ACM, 1991] Computer supported cooperative work. *Communications of The ACM*, *34*(12).

[Almy, 1991] Almy, D., and Ramanathan, J. "Enterprise Workflow Management: Implementation and Observations." IDEF Users Group Conference.

[Althoff, 1990] Althoff, J. L. CPIM. An implementation of the ANSI/SPARC three-schema architecture. Conference Reading, Society of Manufacturing Engineers.

[AMICE, 1989] Open system architecture for CIM. ESPRIT Consortium AMICE (Eds.). New York: Springer Verlag: Project 688, AMICE, Vol. 1.

[Ashok, 1987] Ashok, V. A knowledge-based software development assistant. DECUS Symposium, 1987.

[Blattner and Ramanathan, 1979] Blattner, M., and Ramanathan, J. Attributed metaforms for high level design and analysis of algorithms. Proceedings of the Conference on Information Sciences and Systems, April.

[Buffum, 1981] Buffum, H. E. Air force computer-aided manufacturing (AFCAM) master plan. Volume III Analytic Tools, AFML-TR-74-104, AFWAL/MLT, WPAFB, OH 45433, 1981.

[CALS, 1991] A framework for concurrent engineering. CALS Technical Report 003, CALS Industry Steering Group, Suite 300, 1025 Connecticut Avenue, NW Washington DC 20036, Tel. (202) 775-1440, Report of the CE Framework Task Group of the CALS/CE Industry Steering Group, CALS TR 003 March 13, 1991.

[Chandrasekaran and Johnsonson, 1992] Chandrasekaran, B., and T. R. Johnsonson. *Communications of the Association for Computing Machinery*, *35*(9), 124–136.

[Chen, 1976] Chen, P. P. S. The entity-relationship model: Towards a unified view of data. *ACM Transactions, Database Systems*, *1*(1), 9–36.

[Codd, 1979] Codd, E. F. Extending the database relational model to capture more meaning. *ACM Transactions, Database Systems*, *4*(4), 395–434.

[Curtis, 1992] Curtis, B. Process modeling. *Communications of the ACM*, *35*(9).

[EIF, 1990] Enterprise integration framework workshop briefing. IBM Federal Sector Division, Route 17C, Owego, NYB827-1298, 1990.

[Fisksel, 1989] Fisksel, J., and Kayes-Roth, F. Knowledge systems for planning support. Cimflex Teknowledge Corporation, IEEE Expert, Fall, 1989.

[Gupta and Madnick, 1987] Gupta, A., and Madnick, S. Knowledge-based integrated information systems development methodologies plan. Massachusetts Institute of Technology, Knowledge-based Integrated Information Systems Engineering (KBIISE) Report, Vol. 2, 1987.

[Hars and Scheer, 1992] Hars, A., and Scheer, A.-W. Extending data modeling to cover the whole enterprise. *Communications of the ACM*, *35*(9).

[IRDS, 1991] Information Resource Dictionary System. ANSI X3H4, Working Draft, ATIS, October, 1991.

[Ishii, 1991] Ishii, H. Toward an open-shared workspace: Computer and video fusion approach of team workstation. *Communications of the ACM*, *34*(12).

[Kannapan, 1993] Kannapan, S. M. Structuring and coordinating information in product development.

[KIDS, 1990] Knowledge integrated design system, WRDC, Contract No. F33615-89-C-5619, 1990.

[Klein, 1993] Klein, M. Capturing design rationale in concurrent engineering terms. *Computer*, *226*(1), 39–47.

[Krasner, 1992] Krasner, H. Lessons learned from a software process modeling system. *Communications of the ACM*, *35*(9).

[Kyung, 1991] Kyung, M. Designing for cooperation. *Communications of the Association for Computing Machinery*, *34*(12), 65–73.

[Mayer, 1989] Mayer, R. J. Ed. Analysis of methods. Knowledge-based Systems Laboratory, Department of Industrial Engineering, Texas A&M University, College Station, Texas, 77843, 1989.

[Mayer, 1993] Mayer, R. J. Information integration for concurrent engineering (IICE). IDEF3 Process Description Capture Method Report, AL-TR-1992-0057, Knowledge-based Systems Incorporated, 2726 Longmire, College Station, Texas 77845, 1993.

[Monarchi, 1992] Monarchi. A research typology for object-oriented analysis and design. *Communications of the ACM*, *35*(9), September, 1992.

[Navathe, 1992] Navathe, S. B. The evolution of data modeling for databases. *Communications of the ACM*, *35*(9), 112–123.

[Ramana, 1993] Ramana, Y. V. Reddy, Srinivas, K., Jagannathan, V., and Karinthi, R. Computer support for concurrent engineering. *Computer*, *26*(1).

[Ramanathan, 1987] Ramanathan, J. Knowledge-based assistance for design-in-the-large. Second International Symposium on Knowledge Engineering, Spain, 1987.

[Ramanathan, 1992] Ramanathan, J. Process versus data management technologies. Conference Proceedings, Autofact, Society of Manufacturing Engineers, P.O. Box 930, Dearborn, MI, 48121, 1992.

[Ramanathan, 1993] Ramanathan, J. Object-based integrated design workstation. Final Technical Report, 1993.

[Ramanathan and Sarkar, 1988] Ramanathan, J., and Sarkar, S. Providing customized assistance for life-cycle approaches. *IEEE Transactions on Software Engineering*, *SE 14*(6), 749–758.

[Ross, 1977] Ross, D. T. Structured analysis (SA): A language for communicating ideas. *IEEE Transactions on Software Engineering*.

[Sarkar, 1989] Sarkar, S. The design of a software environment architecture based on executable process descriptions. The Ohio State University, Computer and Information Sciences Department, 2036 Neil Avenue Mall, Columbus, Ohio 43210, 1989.

[Seybold, 1992] Seybold, P. B. Office Computing Group Report, Vol. 15, No. 9, 1992.

[STARS, 1991] Software Technology for Adaptable, Reliable Systems (STARS) Program, Software Process Tools and Techniques Evaluation Report, Version 1.0, Contract No. F19628-88-D-0032, ESD/AVS, Electronic Systems Division, Air Force Systems Command, Hanscom Air Force Base, MA 01731-5000, 1991.

[U.S. Department of Commerce, 1993] Functional Process Improvement, DOD 8020.1-M, Office of the Assistant Secretary of Defense, Washington, DC 20301-3040. U.S. Department of Commerce, National Technical Information Service, 5285 Port Royal Road, Springfield, VA 22161, Commercial Telephone: 1-703-487-4650, 1993.

[Waldron, 1988] Waldron, M. B. Modeling of the design process. In Yoshikawwa and Gossard (Eds.), *Proceedings of IFIP Working Group 5.2 Workshop on Intelligent CAD*. North Holland, 1988.

[Waldron and Waldron, 1988] Waldron, M. B., and Waldron, K. J. A time sequence study of complex mechanical system design. *Design Studies*, 0142-694X188102095-12, Butterworth & Company (Publishers) Ltd., Vol. 9, No. 2, April, 1988.

[Williams, 1990] Williams, T. J. The purdue reference model for computer integrated manufacturing from the viewpoint of industrial automation. SME Conference Proceedings, 1990 Update Standards in Industrial Automation Status and Future, May 14–16, 1990.

[WRDC, 1990] DAPro project integrated information support system (IISS) enterprise integration framework. Technical Report Prepared by Control Data Corporation for Manufacturing Technology Directorate, WRDC, Air Force Systems Command, WPAFB, Dayton, Ohio, Vol. V, Part 50, September 30, 1990.

17
Improved Total Development Process: Overcoming the Ten Cash Drains

Don Clausing

The Ten Cash Drains

The existing development process of a product in the United States delivers products that are only average in quality and cost, and are delivered to the market late. (Japan is the benchmark.) This is the result of the Ten Cash Drains:

1. Technology Push, but Where's the Pull?
2. Disregard for Voice of the Customer
3. Eureka Concept
4. Pretend Designs
5. Pampered Product
6. Hardware Swamps
7. Here's the Product; Where's the Factory?
8. We've Always Made It This Way
9. Inspection
10. Give Me My Targets; Let Me Do My Thing.

The total cash drain is easily 10 percent of corporate revenues; often much more when the full costs are accounted

Total Development Process

The improved Total Development Process plugs the Ten Cash Drains, greatly improving the financial positions of the corporation. Good products are brought to market in a timely fashion. By continuously bringing better products to market, the improved Total Development Process provides dynamic rejuvenation. This chapter describes the improved Total Development Process actions in overcoming the Ten Cash Drains.

A zero-level block diagram is shown in Figure 17.1. Product and process technologies are generated and incorporated into specific product developments, which are then placed into production. This is shown in more detail

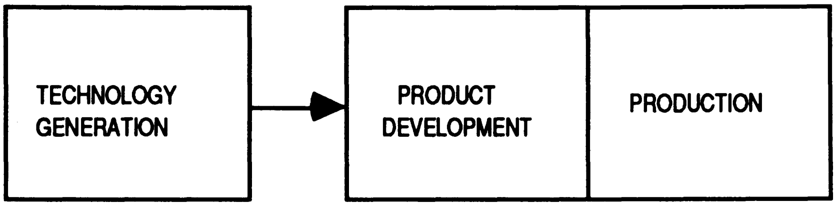


FIGURE 17.1.

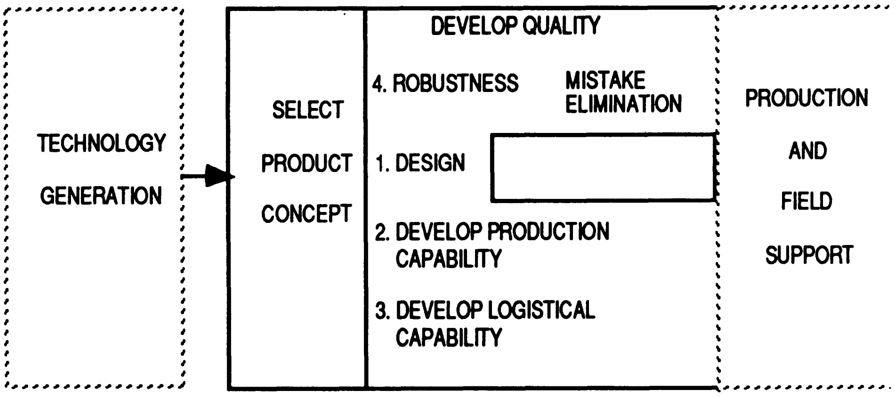


FIGURE 17.2. Concurrent development with multifunctional team: 1. production design of product; 2. production capability; 3. logistics capability; 4. robustness.

in Figure 17.2, which displays the fundamental concept of concurrent engineering. The product design, production capability, and logistical (field support) capability are developed concurrently to have robust quality. The improved Total Development Process is shown in operational detail in Figure 17.3, with the first 14 months shown in still more detail in Figure 17.4. Figures 17.3 and 17.4 show 38 months to complete the development of a specific product, from the start of system design until the start of production. This is the improved, shorter time that will be achieved for complex products, such as a car, or a very large, complex copier or printer. Simpler products have a scaled-down schedule, approximately 1 year for a VCR for example. Large aerospace projects will take longer, perhaps 5 years. The schedule of 38 months is to be compared with typical schedules of 60–70 months in previous practice, slowed by the Ten Cash Drains.

The general structure of Figures 17.3 and 17.4 is common to all development activities. In Figure 17.3 the major activities (system design, etc.) have been separated for clarity. Of course, in actual practice they are tightly integrated together, and performed by one multifunctional product development team. The most important details are described in the following sections in the context of overcoming each of the Cash Drains.

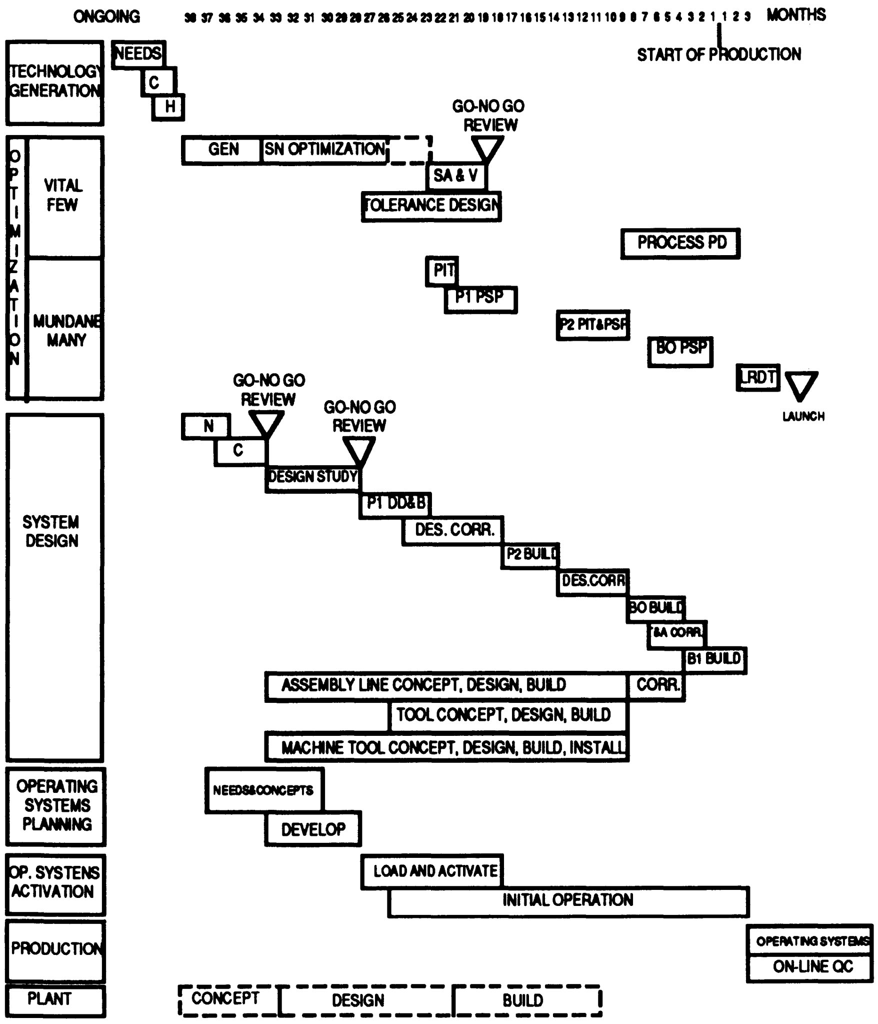


FIGURE 17.3. Improved Total Development Process. N = needs; C = concepts; H = hardware; GEN = generic. Here it refers to SN optimization of generic technological concepts, which precedes optimization of specific product embodiments of the technology; Process PD = process parameter design (SN optimization of production processes); PIT = problem identification test; P1, P2, B0, B1 = Four successive iterations of prototype; PSP = problem-solving process; LRDT = launch readiness demonstration test; DD&B = detail design and builds; SN = signal-to-noise ratio. The signal is the performance that we want. The noise is undesirable deviations from the intended performance; SA&V = system adjustment and verification; Des.Corr. = correction of design mistake; T&A = tool and assembly; QC = quality control.

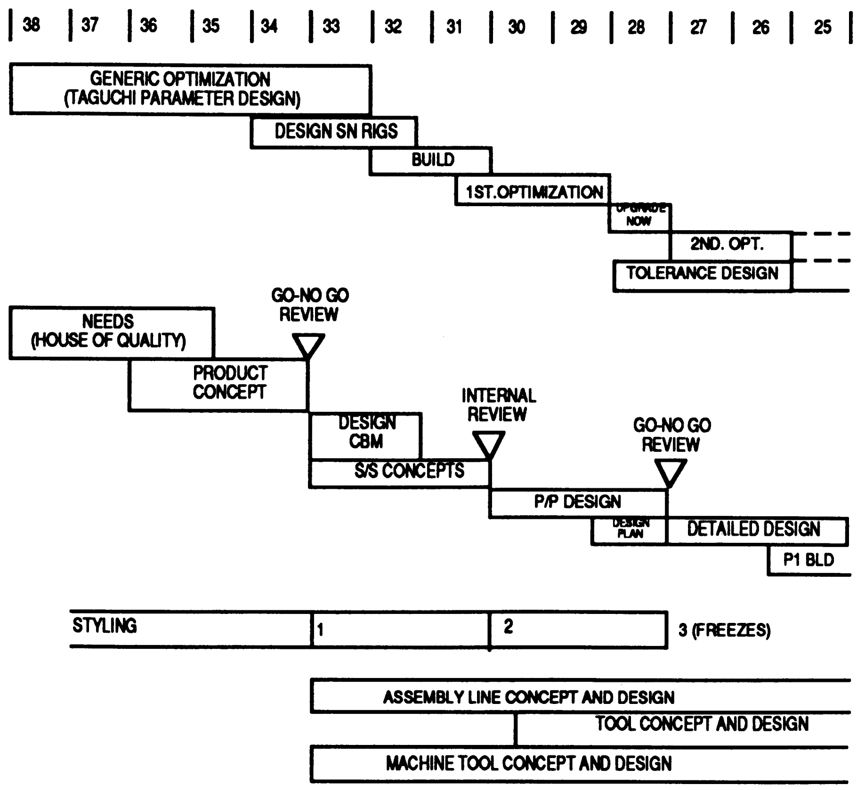


FIGURE 17.4. First 14 months of Figure 17.3 in more detail (for system design and optimization).

Overcoming Cash Drain 1: "Technology Push, but Where's the Pull?"

Technology Push, but Where's the Pull?

The United States is very strong in technology generation. However, even in this area of strength, there are three significant problems: (1) new technological concepts are developed and often major resources are spent, but no discernable customer need can be identified; (2) there are strong customer needs for which technology generation activities are lacking; (3) good concepts are developed for which there are clear customer needs, but the technological concepts are inadequately transferred into the development of a specific product.

Technology push results in many "one but" concepts. These concepts are fantastic, but ... they fail to meet a major customer requirement. Often the cleverness of the concept or its attractiveness in meeting other customer requirements results in much money being drained into its development before it is totally recognized that the concept has no potential because it is inherently incapable of satisfying an outstanding customer requirement. At the other extreme, there are often major customer requirements and needs for product or process improvements for which there is no ongoing technology generation activity. This often leads to creation of a significant new concept during the system design. Usually, this leads to disaster. Such concepts are too immature to be developed on a product development schedule during system design. It should be a strong operating principle that significantly different new concepts are not selected during system design unless sufficient prior technology generation activity has occurred to develop the concept to a sufficient level of maturity.

Often, good new concepts are generated that have the potential to meet customer needs but they are only poorly transferred into the system design activity. These concepts often go down a technology drain and never make it to the market. This usually leads to prolonged blame-giving between the technology generation organization and the system design organization.

In summary, the United States is very strong at technology generation, but much of this strength is dissipated by excessive technology push and inadequate focus on the strategic needs of the corporation and its customers.

Technology Generation

Technology generation is strong in the United States, based on many people with a deep understanding of natural phenomena and an independent drive for creativity. There is great opportunity to build upon this base to achieve a more complete technology generation activity. The Needs Phase (see Figure 17.3) identifies the technologically strategic needs of the particular industry and market segment. The Concepts Phase generates new concepts that are responsive to the needs and selects the best concepts for further development. The Hardware Phase is a continuation of the invention activity in the laboratory. At the completion of technology generation, the concept has been demonstrated to work very well at one operating condition, and to be strongly attractive in meeting customer needs.

The coherent technology strategy, which identifies major customer needs that require new technological thrusts, is the prime activity to assure that there is sufficient technology generation activity addressing all major strategic customer needs. This strategy will also go far toward helping to quickly identify the "one but" concept that should not receive further investment. Of course, the invention of new technology requires creativity. Volumes have been written on the subject, with uncertain conclusions. The author believes that the major requirements for creativity are (1) deep involvement in identifying the needs, (2) strong understanding of physical phenomena, and (3) creative environment (rewarding, not bureaucratic).

The only way to achieve successful technology transfer is to form integrated teams to do the initial steps in the system design. The integrated teams have people from both the technology generation activity and the systems design activity. This integration leads to successful transfer of the new concept into the system design activity so that it is enabled to go downstream starting from a strong system design base. All attempts at a formal hand-off procedure without team integration have failed.

At the conclusion of technology generation, the concept has been shown to work very well at one operating point and to be very attractive at meeting customer needs. The remaining tasks are then to integrate the concept into a specific system design and to optimize the concept for low cost and high quality, so that it will work well when subjected to a wide range of operating conditions (robust design). The initial work on robust design (good performance throughout a wide range of operating conditions) is the important transitional link between technology generation and the development of a specific product. Before a new technology is finally selected for a specific product there should be a clear demonstration of the potential for robustness.

The term *technology generation* is too restrictive to describe the ongoing activities to provide the corporation with increased capabilities. These activities can best be thought of as "generic improvement," with technology generation as a very important element. Other generic improvements can be achieved by a thorough attention to technical detail, coupled with close attention to the needs of the customers. This can be started by preparing a House of Quality (introduced in the next section). An example of this evolutionary type of generic improvement is the rust-prevention improvements that were made at Toyota Auto Body in the late 1970s.

Summary

Strong technology strategy and technology transfer assure attractive new technological concepts to meet major customer needs.

Overcoming Cash Drain 2: "Disregard for Voice of the Customer"

Disregard for Voice of the Customer

The first step in system design is the determination of the needs of the customer. Often, in the United States, the product is doomed to mediocrity by the date of a completion of this Needs Phase (see Figures 17.3 and 17.4). The biggest culprit in this cash drain is deployment of the voices of corporate specialists rather than the voice of the customer. This further aggravates the technology push problem that is Cash Drain 1. Frequently, the criteria that are used during the ensuing Concept Phase have already lost or distorted much of the voice of the customer.

House of Quality

The House of Quality (Hauser and Clausing, 1988) and the procedure for developing it, which have evolved in Japan since 1971, provide an excellent method for deploying the voice of the customer. The House of Quality is a planning table that shows explicitly the deployment from the voice of the customer to product planning characteristics. Also shown are competitive benchmark evaluations for existing products.

The House of Quality must be prepared during the Needs development (see Figures 17.3 and 17.4) by an integrated team consisting of people knowledgeable about marketing, market research, product planning, product design, process engineering, service, and perhaps other functions. This integrated team brings together all of the best information on customer needs for the market segment for which the new product is intended. They work together to achieve consensus on the required product planning characteristics and quantitative target values that are fully responsive to the needs of the customer and will lead to a superior product relative to competitive benchmarks. This systematic process, which is guided by the format of the House of Quality, leads to understanding by all of the major functions of the corporation as to the required product characteristics. This consensus and understanding leads to commitment by all functions and, therefore, eventually by all of the involved people within the corporation. Gaining the full understanding and commitment of all people to the deployed voice of the customer is critical to successful development of commercially viable new products. The result is a major improvement over what usually happens in the United States today. The House of Quality deployment is the beginning of several steps of systematic Quality Function Deployment. Quality Function Deployment assures that the voice of the customer guides work in all functions of the Total Development Process, and manages the information so that all of it is utilized in producing the actual product.

Summary

The House of Quality and the subsequent steps of Quality Function Deployment assure that the voice of the customer is deployed and all activities are guided by and are responsive to the needs of the customer.

Overcoming Cash Drain 3: "The Eureka Concept"

The Eureka Concept

Often the selected product concept is the result of someone shouting "Eureka, I have this great new concept." It becomes the only concept that is given serious consideration. All too often it does not stand the test of time. It does not even stand the test of time to bring the product to the market, much less the time of its actual production. Many concepts look good in the initial flush of creation. However, when such concepts are quickly accepted and a strong run is begun heading for the marketplace, they are usually found to be vulnerable to an intrinsically superior concept. It is a tremendous cash drain to waste nearly all of the Total Development Process upon a concept that has been recognized as highly vulnerable by the time the product actually reaches the market.

Pugh's Concept Selection Process

The search for the invulnerable product concept can be greatly improved by the use of the concept selection process that has been developed by Stuart Pugh (Pugh, 1981) and his colleagues in Great Britain. In this process, we carefully avoid a hasty running away with a singular concept. It is required that a large number of concepts that are in real contention and have a chance of being selected are available before selection is started. A large number is at least 10 and preferably 20–30. These concepts are then carefully evaluated with respect to each other by using the criteria that have been developed during the Needs Phase. This process is thus strongly focused on the voice of the customer, and it avoids the excessive and premature quantification that is a glaring weakness in many selection processes now used in the United States. This process is a team activity and is designed and facilitated to keep everyone thinking about the concepts and the criteria. This process leads to great clarification of both the concepts and the criteria that are being used for their selection. New concepts and new criteria emerge and some existing criteria are found to be not relevant. This process often employs four to six iterations of the formal matrix that is used for concept selection. At intermediate stages of the process, the total number of concepts may grow or shrink, but eventually the team converges on an invulnerable concept. This process may take $1\frac{1}{2}$–3 months. At the end of this time, the team is confident that they have picked a winning concept and they are committed to its success.

Summary

Use of the Pugh Concept Selection Process leads to a selected concept that is invulnerable to being quickly surpassed in the marketplace, and achieves team commitment that is crucial to the success of the remainder of the Total Development Process.

Overcoming Cash Drain 4: "Pretend Design"

Pretend Designs

Pretend Designs are not production intent,* are often simply new but not better, and become focused on the creation of experimental hardware. This initial design comes to have as its objective the building of the first prototype (P1), rather than the achievement of the best possible design of the final product in the product marketplace. These designs are usually motivated by a strong desire to be new and different, but all too often the result is demonstrably inferior to a design already in the field. A lack of production intent leads to the attitude, "Oh well, this is just the first design—I'll fix this all up later." This is a sure road to disaster.

Design

To avoid pretend designs, it is critical to separate the initial design into two phases. The first phase is the design study (see Figure 17.3), which is aimed at achieving the best possible design. At the completion of this design study, there is a "go–no go" review, and if the resulting design does not meet criteria for a successful product, the P1 prototype will not be built. This is a sure way to avoid concentration on experimental P1 hardware, and to free everybody's creativity to achieve the best possible production-intent design. The second phase, after review has been successfully completed, is the completion of the detailed dimensioning, and the building of the first (P1) production-intent prototypes.

The design must start with a concentrated activity of design competitive benchmarking by the engineers who are on the design study team. The competitive benchmark products are disassembled by the design study team down to the individual piece parts. The function of each part in the total competitive product is analyzed, the cost of each part is carefully estimated, and a best evaluation is made of the probable production processes that were used to make each part. The parts are arranged on piece-part boards with the estimated cost next to each part. This design competitive benchmarking must be done by the engineers on the design study team. It cannot be done by anyone else to achieve the full beneficial effect upon the eventual product design. Furthermore, each engineer is challenged to beat the competitive benchmark design or use it. This has a tremendously beneficial effect. It assures that no designs will be used simply because they are new. This avoids the common problem of designs that are new, but clearly inferior to existing designs already in the field. With the challenge of beating the competitive

* Engineering term that means that the design is not intended for production.

benchmark design or using it, each engineer becomes strongly concentrated upon achieving a better design. Sometimes the initial reaction is one of awe and respect for a competitive design. For a few weeks, the competitive design may be carried as the selected design concept at the detailed level for the new product. However, engineers are never happy to use someone else's design and, in this situation, are strongly motivated to come up with a superior design. Inevitably, they do so. As a result, every functional area of the product is superior in its design concept.

The design team is strongly trained in the methods of design for assembly, design for piece-part producibility, and value analysis/value engineering (VA/VE). They apply this training in carrying out the design activity to achieve the best possible design, very producible and serviceable. The process is done in a continuous style, avoiding the setting apart of small time periods to do specific activities such as VA/VE. Instead of setting aside 2 weeks to do VA/VE and then feeling that that chore has been completed, this improved process emphasizes the continuous application of VA/VE throughout the entire design process.

The design activity itself is divided into two phases (33–28 months in Figure 17.4). Halfway through the design study, the subsystem concepts are selected and frozen so that nothing on the subsystem concept drawings can be significantly changed for the remainder of the Total Development Process. At this time, there is a small internal review to assure that everyone understands the concepts that have been selected and how they integrate together to create the superior total product.

Next, attention is concentrated upon creating best possible piece-part designs and the selection of the right components.

In doing the design work the skills in "partial design" that have been learned in school and by experience are fully utilized. Such skills as design of machine elements and circuit design are employed, with emphasis upon engineering fundamentals. When physics and design are in conflict, physics always wins. Here we are concentrating on "total development," in which "partial design" is embedded.

As soon as attention is focused on piece-part design, potential suppliers are brought in and, to the fullest possible extent, utilized as full participating members of the design study. It is important to fully engage the technical expertise of the suppliers at this early stage. If necessary, key suppliers with special expertise should be given design consultation contracts to assure their full participation and the utilization of their technical knowledge. (In some products, entire subsystems may be designed by a supplier. Of course, in this case, the supplier would have been involved from the beginning of the design study.)

In the design activity, it is always necessary to have a drawing of the complete product, often referred to as the "big picture." It is extremely important that this big picture drawing be continuously updated, at least daily, in the most efficient way. This can easily be done with modern computer-aided design systems. The design files of the team members can be readily transferred to the big picture to assure that integration of all aspects of the design is occurring.

At the completion of the design study (after month 28 in Figure 17.4), there exists detailed layout drawings of each subsystem, and drawings of almost every piece part. However, the piece-part drawings may be dimensioned only with respect to critical parameters. Also, at this time, a detailed cost estimate has been prepared, functional analyses including failure modes and effects analysis (FMEA) have been completed, initial processing decisions have been made, and a styling model is ready. All of this information is incorporated into a design plan to assure that all design activities during the remainder of the Total Development Process are strongly guided by the crucial work that has already been completed. This information is presented at a crucial "go–no go" review. This review determines that the design is inherently superior to competitive benchmarks and meets all aspects of the business strategy, or the development is stopped and returned to the beginning of the process to redefine the required characteristics.

Summary

The design activity focuses everyone's attention on creating the best possible design for the actual production product, and thus avoids the cash drain of the "Pretend Design."

Overcoming Cash Drain 5: "Pampered Product"

Pampered Product

Most products work well at one operating condition. The old-fashioned approach in the United States pampered the product concept to enable it to look good, especially in demonstrations for vice presidents. The product was not seriously challenged, but rather, was pampered by special tuning and tinkering so that it would put on a very good demonstration. The pampered product approach has been improved upon by a rigorous application of reliability growth and problem-solving methodology. However, the reliability growth and problem-solving-process methodology is not an adequate approach to the optimization of the vital few design parameters in order to achieve robust performance. This misapplication of reliability growth and problem-solving process is inspection of the design. It has all the faults of inspection during production (see Cash Drain 9). It results in countermeasures being brought to bear too late when they are very expensive and often do not catch all of the problems before the product is in the hands of the customer. In this approach of problem identification and problem solving to grow the reliability, purposeful improvements are not made until the product has been detected as being defective. This approach is very ineffective in optimizing the vital few design parameters that are the most unique aspects of the new design and control its most important performance characteristics. The problem with the problem-solving process approach to optimization of the vital few design parameters is that it is incapable of detecting situations that are very close to being a problem but are not actually a problem on the specific hardware being tested at the specific operating point of the problem-identification test. Therefore, situations that are just about to go over the cliff will not be detected. However, if further improvement is not made, such a design will perform poorly in the field where new conditions of use following realistic production conditions will cause a product to go over the cliff and have serious problems. Therefore, it is very important to have a problem prevention approach in the optimization of the vital few design parameters that will assure that the design is not only performing well in some limited test, but is actually very far away from any problem-causing cliff and, therefore, under realistic conditions, will remain on safe operating ground and not fall off the cliff.

The problem-solving process is very satisfactory for correcting simple mistakes in the design. However, a systematic optimization process is required to achieve robust performance by finding the best values for the critical design parameters.

Optimization of Quality

Quality has two aspects:

1. Elimination of mistakes
2. Robust performance.

Robust designs keep performance close to the ideal customer satisfaction value, even when the design is subjected to the actual conditions of customer use. The two aspects of quality are associated with the two styles of decision making:

1. Experience is sufficient
2. Decisions must be optimized, experience is not sufficient (some ignorance remains).

When experience is sufficient, the only problem is simple mistakes (human error). A complex system may have 10^7 design decisions, many of them mundane. An error rate of 0.01%, which seems very good, will still create 1000 mistakes. These then must be found and eliminated. The common process of problem-identification tests, problem-solving process, and design correction, guided by historical reliability-growth curves, is successful. Here we will concentrate on the second aspect of quality, robust design. Robust performance is controlled by the critical (vital few) design variables. Here it is not a question of correcting mistakes. These parameters are sufficiently unique to the present design that is being developed, so that it is not possible for even the best engineers to get it right the first time. Therefore, a systematic decision-making process is needed to select the best numerical values for the vital few design parameters that are critical to the success of the product. (Although the word

The optimization process (see Figures 17.3 and 17.4) for the vital few design parameters must have two characteristics. First, it must be capable of systematically and rapidly making the right numerical choice for the values of each critical parameter, even though total understanding of the phenomena that control the function of the system is not available. Waiting for total understanding will inevitably take much too long for the completion of the total development process and the product will reach the market too late for its unique characteristics to be especially attractive to customers. Secondly, the optimization process for the vital few design parameters must be capable of preventing problems by recognizing that although the system may actually be working satisfactorily, it is very close to a problem and will fall off the cliff under actual customer conditions. The outstanding optimization method for rapidly and economically achieving problem prevention by a systematic process of decision-making that utilizes all available understanding, but does not wait for the time-delaying arrival of new understanding, has been developed by Taguchi (Taguchi and Clausing, 1990).

The most important single improvement in the Total Development Process is the optimization of the signal-to-noise ratios. The signal-to-noise ratios have been developed by Taguchi as a measure of robustness (the proximity of potential problems). As the systematic optimization process increases the values of the signal-to-noise ratios, the system design moves farther and farther away from the occurrence of any potential problems. The signal-to-noise (SN) optimization is known as parameter design because the nominal values for the vital-few design parameters are optimized by this process.

SN optimization is initially done on the concepts that have shown promise during technology generation. During this initial SN optimization these concepts are often still generic rather than having been applied to a specific system design. The generic SN optimization is very beneficial in readying a concept for application in a specific system design. For more complex and unique concepts it is a requirement that some generic SN optimization has occurred before the concept can be selected for a new system design.

It would be easy to first do SN optimization of the generic concepts that have emerged from the technology sensation activity, then design these concepts into a specific system design and build prototype hardware, which would then be followed by again performing SN optimization on this prototype hardware. Although this approach, carried out in a competent manner, would be foolproof and result in the best possible system design, it would have the glaring shortcoming of taking too long in getting the product to market, after the market window had slammed shut. Therefore, in the interest of efficiency, it is necessary that the SN optimization be done simultaneously with the critical initial portion of the system design. This is clearly shown in Figures 17.3 and 17.4. Although a considerable challenge, this simultaneous optimization and system design can be done. Therefore, to have a world-class total development process, it must be done.

The design of the SN rigs begins near the end of the Concept Phase of the system design. By that time, the product concept is sufficient to guide the design of the SN rig. As much as possible, the SN rig should be based on hardware that already exists. This existing hardware will be hardware that was developed during technology generation and generic SN optimization, and mules (existing production hardware that is modified to accept new subassemblies for the purpose of doing the SN optimization). The first SN optimization will usually consist of two or three iterations. It is completed in time for the information to be easily incorporated into the new product design.

After the completion of the first SN optimization, the SN rigs are often upgraded to incorporate important design changes that more closely reflect the current system design. The second SN optimization is then completed. With close coordination to achieve quick design and hardware implementation of the results of the second optimization, the P1 prototype hardware will completely capture all of the design decisions that are made during the SN optimization.

The early completion of the SN optimization, so that its results are completely captured in the P1 hardware, is a critical and extremely beneficial feature of the improved Total Development Process. In the past, failure to do adequate SN optimization has led to the presence of many borderline problems that greatly plague and complicate the ensuing elimination of mistakes. The borderline problems come and go intermittently. When they come, they cause major shortfalls in performance to the point where it is difficult to work on the mistakes. When these intermittent problems go away temporarily, they cause a false sense of security, which is then demolished when the problems again return. This constant going and coming of major problems has a demoralizing effect upon the entire development activity. By eliminating these major intermittent problems very early, the SN optimization allows subsequent concentration on simply correcting the many mundane mistakes.

After the completion of the SN optimization, the best nominal values have been established for each of the vital few critical design parameters. Then tolerance design is performed. The most economical level of precision for each of the critical production processes is selected. Economy is assured by using Taguchi's Quality Loss Function (Phadke, 1989) to predict the quality loss that will occur in the field as a result of the selected level of precision. This is added to the manufacturing cost and the precision level is selected that minimizes the total cost. Previously, at this stage of the development process, there has been no ability to put a dollar value on the loss that would be incurred because of the selected level of precision. Taguchi's Quality Loss Function has enabled this rational tolerance-design process to replace what had previously been an emotional and trying negotiation between product designers and process engineers. After the best level of precision for each critical production process has been selected, then production tolerances are placed on the drawing. However, in the improved Total Development Process, the production tolerances on the drawings are almost superfluous for many products.

After the P1 prototypes are available, system adjustment and verification is performed. Some of the P1 prototypes are devoted to this activity. (The other P1 prototypes will be used to correct the mistakes.) The SN optimization has minimized the critical variances in the outputs of the subsystems of the product. However, the nominal or "mean values" for these subsystem outputs that were chosen in the system design may not be optimal. System adjustment is the adjustment of the mean values of the critical subsystem output performance characteristics to assure that the total system has the best possible performance. This system adjustment is easy to do. After the system adjustment is completed, then a system verification test (SVT) is performed. In the system verification test, the P1 prototypes are tested in a head-on showdown with the competitive benchmark product(s). In this head-on competitive benchmark system verification test, the critical SN ratios are compared between the new product and the competitive benchmark product. The new product should show improvement in the critical SN values. The amount of improvement that is required to have a world-class product when the new product appears in the market can be easily estimated. At the completion of the system verification test, a "go–no go" review is held. If the product fails this head-on comparison of SN ratios with the competitive benchmark, it means that the product is inherently incapable of being a major success in the marketplace. If this should happen, this specific product development activity should be terminated. A return should be made to the beginning of the system design to redefine the system concept that will be superior for the new market entry date. Based on the experience that has just been attained, a better system design should be achieved. This is very preferable to throwing good money after bad. Of course, by using the improved Total Development Process, the specific product development activity will almost never have to be cancelled. After this review, there should never again be any serious consideration given to the possibility that the specific product development activity will be cancelled. Instead, all effort should be concentrated in getting the best possible product to the market at the scheduled date, or earlier.

Simultaneously with the system adjustment and verification activity, other P1 prototype machines will be devoted to problem identification testing and the problem-solving process. In this activity, the mundane mistakes will be weeded out to achieve reliability growth. This is then repeated on the P2, B0, and B1 hardware.* All product design changes should be completed at the end of the P2 problem-solving process. The B0 iteration is to correct mistakes in the production processes. The B1 iteration is to verify that the production processes have been correctly implemented in factory operations.

At the completion of the optimization, a launch-readiness demonstration test (LRDT) is performed on the B1 hardware. In this test, simulated customer use conditions are utilized. This test should not be regarded as a significant step in the development of quality. If the entire process has been done well, the LRDT will provide no new significant insights into the new product. It is done strictly to convince the management of other major corporate functions, and to provide data to guide the fine tuning of the sales and service plans.

Summary

Taguchi's methods of optimization challenge the new product instead of pampering it, and ensure that the design is as far away as possible from all potential critical problems.

Overcoming Cash Drain 6: "Hardware Swamps"

Hardware Swamps

Hardware swamps occur when the prototype iterations are so numerous and so overlapping that the entire team becomes swamped by the chores of debugging and maintaining the experimental hardware. The hardware swamp can become so severe that no time remains to improve the design. The prototype hardware has become an end unto itself, rather than its being used to improve the design. A hardware swamp can be recognized by laboratories packed full of experimental equipment and an inability to complete any organized experiments because of the voracious appetite of the hardware to be debugged and maintained.

Prototypes Enable Optimization

Only enough prototypes are built to enable the successful completion of the optimization process. Only four iterations of the prototypes are needed (see Figure 17.3) to achieve successful optimization, two design iterations, and

*Four major iterations of prototypes is the most that should be required. Many products only require two. Of course, if the production volume is small, in the limit only one, then prototypes may not be appropriate. In this case there may be many minor iterations (changes to parts and subsystems) on the few units of production, the objective being to eliminate mistakes and achieve some final improvement in robustness.

two production iterations. (For products that are significantly simpler than cars, copiers, and computers, only two iterations are required.) The early optimization of robustness (before prototypes are built) enables the mistakes to be eliminated in a few iterations of prototypes. There is sufficient time between each iteration of prototypes to enable the data from the previous prototype to be incorporated in the build of the next prototype. This is essential for an efficient process.

Prototypes are built only to enable the development of quality. Building prototypes for the sake of building prototypes or to achieve some ideological number of iterations is rigorously avoided. Some of the P1 prototypes are used to verify robustness. The remaining P1 prototypes and the P2 prototypes are used to eliminate mistakes from the product design. The B0 and B1 prototypes are used to eliminate mistakes from the production equipment and processes. There should be continuous efforts to reduce the number of iterations that are required.

Summary

Early optimization of robustness enables the number of prototypes to be greatly reduced.

Overcoming Cash Drain 7: "Here's the Product. Where's the Factory?"

Here's the Product. Where's the Factory?

Past practices have all too often developed a product to an almost final stage before looking at how it might produced. This is a sure road to failure. The production capability must be developed along with the product design. It is not a design if we don't know how to make it. In doing the design study, design for assembly and design for piece-part producibility must be emphasized. Here we are describing the need to simultaneously develop the production capability in close coordination with the product design. If the design of the production capability starts only a few months before actual production, many severe problems are guaranteed to occur. There is a similar requirement for field operations, particularly service.

Development of Production Capability

In parallel with the product system design, the capability must be developed (see Figure 17.3) to produce piece parts, assemble the product, and have the operating systems that are necessary for factory and field operations. The development of the piece-part production and assembly capability can be best thought of as a part of the system design. The result is a total product-and-process system.

Piece-part production capability has three degrees of required development, depending on the portion of the process and equipment that is new:

1. The piece-part production process is a new, unique, clean-sheet (starting with a clean sheet of paper) process. In this case, the development process for the clean-sheet production process is the Total Development Process itself. The clean-sheet production process must go through all of the steps of the Total Development Process, including system design and optimization. (Instead of P2 prototypes, the actual production equipment is built).
2. Dedicated capability. In this case, the design of the machine tools themselves must be tailored to the specific product system design. However, the function of these machine tools is conventional and requires little or no development.
3. The machine tools are standard and in place in the factory, and only fixtures must be designed and built for the specific product.

The lead time becomes progressively shorter. A clean-sheet production process must go through the Total Development Process. The dedicated processes can start a bit later, but still must go through a long development activity. The third case, where only the tools and fixtures must be designed and built, can have very short lead times. This is the advantage of flexible manufacturing. Of course, most piece-part production has for a long time been somewhat flexible in the sense that one machine tool could make parts for many different products. The shorter development time for tool design and build that is displayed in Figures 17.3 and 17.4 reflects the advantage of conventional flexible manufacturing. An increased emphasis upon flexible manufacturing can greatly reduce the time that is shown in Figure 17.3 and 17.4. However, one must be aware of the longer production cycle times that frequently accompany increased flexibility. Therefore, the shorter development time and the ability to more smoothly and rapidly enter production must be balanced against increased cycle times.

The piece-part production tools and the assembly equipment must go through a system design activity and correction of mistakes that is quite similar to the system design and problem-solving process that has already been shown for the product and clean-sheet processes in Figures 17.3 and 17.4. It is important that all of these production capability development activities be started well before the P1 build so that they can influence the style of the P1 build, and so that much information can be attained from the P1 build to guide the design and development of the piece-part production tools and production assembly line.

Many operating systems are necessary for factory and field operation. Although such systems have long been in use, there is continuous opportunity to improve them, and many developments are now producing major improvements. Examples of such operating systems are configuration management, change management, prototype build, problem management, service documentation and training, spares management, unit manufacturing cost (UMC) tracking, process sheets, CNC, production routing and scheduling, ordering, the manufacturing quality system, and distribution. These operating systems are usually in the curriculum and area of expertise of business schools in the United States. There is a need for closer involvement and participation of business school graduates and engineers in the development and implementation of these operating systems. In many companies, these operating systems have taken on a life of their own and have become excessively complex, and too difficult and time-consuming for the average design and process engineer. The design and process engineers must load the operating systems with the design data for the specific product design, activate the systems, and initially operate them for prototype builds, and, most importantly, pilot production (B0 build). There is much opportunity to improve the flexibility of operating systems through the use of computers. The design database is created during System Design. Then it must be very easy to load, activate, and operate the operating systems.

Summary

Development of production capability and logistical (field) capability in parallel with product development assures smooth and efficient transition into factory operations and field operations.

Overcoming Cash Drain 8: "We've Always Made It This Way"

We've Always Made It This Way

The process operating points (speeds, depth of cut, feed rates, pressures, temperatures, etc.) are specified on process sheets or NC programs. The values for the process parameters have often been fixed for a long time, and even originally were the result of little development, if any. "We've always made it this way and it works." Yes, but has the process been optimized to achieve minimum cycle time and maximum quality?

Process Parameter Design

Taguchi's methods of parameter design have proven to be very successful in the optimization of production processes. This action is completely analogous to the SN optimization of the product. Here the parameter design improves the precision of the process, while holding or reducing the cycle time. This greatly improves product quality and reduces production costs.

Process parameter design is done shortly before start of production (see Process PD in Figure 17.3). This enables production tooling to be used. (For clean-sheet production processes, the process is SN optimized much earlier, at the same time as the product.)

Process parameter design can be successful and very beneficial even after start of production. Ford Motor Company and the American Supplier Institute have demonstrated this type of success.

Summary

Taguchi's method of process parameter design greatly improves production processes.

Overcoming Cash Drain 9: "Inspection"

Inspection

Inspection in the factory means sorting the good from the bad after production has been completed. This is now widely recognized to be a poor process for most products, largely through the efforts of Deming (Deming, 1986).

On-Line QC

On-line quality control (QC) (see Figure 17.3) eliminates the waste of inspection. This was first recognized by Walter Shewhart during the 1920s, when he created his famous control chart. Japanese companies used the control chart with great benefit from 1950 until recently. During the 1980s, the control chart has been increasingly employed in the United States. However, Taguchi has developed a method of on-line QC that is more active in improving quality than the control chart, and which is now widely used in Japan. This is a method of optimal checking and adjusting. The machine operator measures every *n*th part, and if it deviates from the target by more than a predetermined adjustment limit, the process is adjusted back to the target. The checking interval, *n*, and the adjustment limit are set at values that minimize total cost (quality loss in the field plus factory cost). Taguchi's Quality Loss Function is used to estimate the quality loss in the field.

The method of optimal checking and adjusting greatly reduces many different cash drains: cost of inspection, cost of scrap, cost of rework, cost of adjustment, and quality loss in the field. It is a simple process to perform, integrating cost-reduction discipline with the machine operator's natural tendency to check and adjust.

Summary

Taguchi's method of optimal checking and adjusting minimizes the direct and indirect costs of inspection.

Overcoming Cash Drain 10: "Give Me My Targets, Let Me Do My Thing"

Give Me My Targets, Let Me Do My Thing

Targets seem good. However, Deming points out that they can tend to restrain improvement. Also, the early allocation of targets down to a detailed level tends to destroy teamwork. The writing of contracts so that each person can then work in isolation seems to have a fatal fascination for the American psyche. It leads to subsystems that cannot be integrated, products that cannot be produced, production capacity that cannot produce modern products, operating systems that attempt to enslave their users, managers who cannot lead, and employees who wait to be told what to do.

Integrative, Participative Management

Employee involvement has made significant progress in the United States since 1980. Integrated, multifunctional teams that have authority commensurate with their responsibility are a key success factor. Managers who lead the process instead of reacting to problems are essential. Holding targets at the highest feasible organizational level produces emotional stress in the American psyche, but it promotes teamwork.

Product development teams should be responsible for development, product, and processes. Once development is complete, the products and processes go through a transition into production operations and field operations.

Summary

Teamwork and competitive benchmarking win over contracts and targets. Management must lead the process.

Strategies

The improved Total Development Process that has been described is carried out in the context of corporate strategies, usually business strategy, product strategy, and technology strategy. These three strategies need to be integrated. The improved Total Development Process is the means by which the implementation of the next phase of the strategies is begun. The development of a new product is started with the intention of bringing a new product to market on a certain date, with quality, costs, and features that appeal to the customers in a certain market segment, as planned in the product strategy. The new product will implement leadership technologies, as planned in the technology strategy. The product will be capable of achieving the financial goals that are stated in the business strategy.

The improved Total Development Process cannot succeed if the strategies are seriously flawed. A beautiful product might be produced for which there is little profit potential. More commonly the strategies are sound, but the development process is weak. The products come too late, with quality, costs, and features that do not excite the potential customers.

It is important that the development people and the rest of the enterprise have a clear consensus about the strategies that are being implemented. This consensus needs to be continuously reaffirmed throughout the development period. The development people sometimes make sound tradeoffs that are not adequately communicated to the sales and service organization. If the sales and service people are surprised by the tradeoffs when production is about to start, much internal resistance may arise.

It is essential to success that the improved Total Development Process be carried out with a clear consensus about the business strategy, product strategy, and technology strategy that are being implemented.

Root Causes

The ten cash drains are the major problems in the development process in the United States. This leads to the question of the root causes for these problems. There is certainly not a consensus about the root causes. It seems to the author that there are two primary root causes:

1. Expectations that are not high enough.
2. Segmentalism; cloistered specialists looking inward within their specialty.

Segmentalism is especially pernicious. Americans have produced elegant solutions to problem definitions that have become increasingly obsolete. Japanese have produced pragmatic solutions to problem definitions that they have made increasingly relevant. Segmentalism makes it difficult to do the integrative thinking that leads to better objectives.

Some people believe that the root causes must be overcome before improvements can be made in the development process. It seems to the author that it is better to improve simultaneously the process to overcome the ten cash drains and work to mitigate the effects of the root causes.

Benefits of Improved Total Development Process

By greatly reducing the Ten Cash Drains, the improved Total Development Process will lead to major improvements in quality, cost, and delivery. This in turn will lead to greater market share and unit profit margins. There is no escape, the technical and managerial details must be mastered, to be followed by continuous improvement. As Deming has written, "Study and hard work will be required."

References

Deming, W. E. (1986). *Out of the Crisis*. Center for Advanced Engineering Study, Cambridge, MA: MIT Press.

Hauser, J. R., and Clausing, D. P. (1988). The house of quality. *Harvard Business Review*, May–June, pp. 63–73.

Phadke, M. S. (1989). Quality engineering using robust design. Englewood Cliffs, NJ: Prentice Hall.

Pugh, S. (1981). "Concept selection, a method that works. Proceedings ICED, Rome, pp. 497–506.

Taguchi, G. and Clausing, D. P. (1990). "Robust quality," *Harvard Business Review*, Jan.–Feb., pp. 65–75.