Computer-aided systems engineering methodology support and its effect on the output of structured analysis

Jankowski, David John, Ph.D.

The University of Arizona, 1994

COMPUTER-AIDED SYSTEMS ENGINEERING METHODOLOGY SUPPORT AND ITS EFFECT ON THE OUTPUT OF STRUCTURED ANALYSIS

by

David John Jankowski

A Dissertation Submitted to the Faculty of the

COMMITTEE ON BUSINESS ADMINISTRATION

In Partial Fulfillment of the Requirements
For the Degree of

DOCTOR OF PHILOSOPHY

In the Graduate College

THE UNIVERSITY OF ARIZONA

1 9 9 4

THE UNIVERSITY OF ARIZONA
GRADUATE COLLEGE

As members of the Final Examination Committee, we certify that we have read the dissertation prepared by David John Jankowski

entitled Computer-Aided Systems Engineering Methodology Support and Its Effect on the Output of Structured Analysis

and recommend that it be accepted as fulfilling the dissertation requirement for the Degree of Doctor of Philosophy

[illegible]	4/15/94
Dr. Douglas R. Vogel	Date
[illegible]	4/15/94
Dr. David E. Pingry	Date
[illegible]	4/15/94
Dr. Sarma Nidumolu	Date
[illegible]	4/15/94
Dr. Titus D. Purdin	Date
	Date

Final approval and acceptance of this dissertation is contingent upon the candidate's submission of the final copy of the dissertation to the Graduate College.

I hereby certify that I have read this dissertation prepared under my direction and recommend that it be accepted as fulfilling the dissertation requirement.

[illegible]	4/15/94
Dissertation Director Dr. Douglas R. Vogel	Date

STATEMENT BY AUTHOR

This dissertation has been submitted in partial fulfillment of requirements for an advanced degree at The University of Arizona and is deposited in the University Library to be made available to borrowers under the rules of the Library.

Brief quotations from this dissertation are allowable without special permission, provided that accurate acknowledgement of source is made. Requests for permission for extended quotation from or reproduction of this manuscript in whole or in part may be granted by the copyright holder.

SIGNED: [illegible]

ACKNOWLEDGEMENTS

I would like to begin by thanking my committee (Drs. Doug Vogel, David Pingry, Suzanne Weisband, Ram Nidumolu, and Titus Purdin) for their efforts in making this dissertation a reality. None of this research would have been possible without the support and resources provided by Dr. Jay Nunamaker. Without the help and cooperation of Drs. Craig Tyran and Ram Nidumolu this experiment would never have occurred. I would also like to thank Drs. Mark Silver and Sirkka Jarvenpaa for answering my questions concerning their work with restrictiveness and guidance. Thank you, also, to Dr. Ron Norman for providing mentoring and support. Dr. Joy Egbert provided valuable editing advice.

My professional life has been made much easier with the help of several of my colleagues. First, I would like to thank Drs. Leonard Jessup and Joseph Valacich for their support, faith, and wisdom. These two are the "total packages." Dr. Mark Fuller served as a sounding board for ideas and as an island of sanity in the months before I left Tucson. I would also like to recognize the support and advice from my colleagues in the College of Business Administration at California State University, San Marcos (especially Drs. Tom Anderson, George Diehr, Robert Black, and Regina Eisenbach) and thank them for their patience.

My friends in San Diego and Tempe have always been there for me. Several friends whom I met while living in Tucson deserve special mention: Bob Mercier, Tyson Henry, Lisa Whitney, Kathy Harbour, Chuck Duerr, Frank DiMaggio, John Sharp, and Keith Coleman. We've gone our separate ways but I will always remember you.

Two women, for better or worse, supported me through part of this endeavor. From Andrea Albi I learned the difference between what I want and what I need. From Mary McAuliffe I learned that when things appear too good to be true they probably are. "I'm still standing, better than I ever did ..."

My family has been especially supportive; I love them very much. I wish to give special thanks to my sister Kathleen and her husband Michael for helping to make my stay in Tucson more enjoyable.

To the Arizona State University Sun Devils, the Phoenix Suns, Queen, Queensryche, U2, and Van Halen -- thank you for helping to make me happy.

Finally, for my scholars -- past, present, and future -- a special thanks for helping to make my chosen profession a more enjoyable one.

DEDICATION

Tucson? The University of Arizona? A die-hard Sun Devil fan? Isn't that a little bit like oil and water? Coming to the U of A has turned out to be one of the best moves I have ever made. Upon my arrival I quickly realized how much I really love the academic life. The campus, the students, the football games, the basketball games, the baseball games, sorority row, the Mall, the Rec Center -- this is what college is all about. Tucson, what can I say? From Sam Hughes to downtown, from West University to North University, from Sabino Canyon to Tanque Verde Falls, from the Santa Catalinas to the Tucson mountains, from the flooded streets to the suicide lanes -- I enjoyed every minute of it. Sunny days, gorgeous sunsets, refreshing monsoons -- I could not have asked for better weather. Dirt Bag's, Shanty, Buffet, Bob Dobb's, El Corral, El Dorado, Cross Roads, Downtown Saturday Night, Pig Pen, Someplace Else, Gentle Ben's, Bum Steer, Wildcat House, Scarlet's, Mud Bugg's, Green Dolphin, Luke's, Mama's, Mike's, Sanchez, Caruso's, 4th Avenue Street Fair, 6th Street Pub, Hutch's, Spot Deli -- yeah, I think I had a good time. As I bring this part of my life to an end I must stop and say "thank you" to the University of Arizona and to Tucson. I will always have fond memories of you.

TABLE OF CONTENTS

TABLE OF CONTENTS *(continued)*

TABLE OF CONTENTS (*continued*)

LIST OF FIGURES

LIST OF TABLES

ABSTRACT

This dissertation investigates the effects of computer-aided systems engineering (CASE) tool methodology support on the system specification output from structured analysis. A replicated project study was employed to allow for control of the requirements specification. Sixteen groups of four upper-division, undergraduate MIS students developed a system specification from the requirements specification of a hotel information system. The groups developed the specifications by following the Yourdon structured analysis methodology, with the aid of two popular, personal computer-based CASE tools. Both CASE tools claim to support the methodology but the methodology support offered by the two tools is different. Specifically, the size of the rule base varies between the tools, and the implementation mechanism for enforcing a specific methodology rule varies both between tools and between rules. It is hypothesized that the number of violations of a particular methodology rule is a function of the implementation mechanism, i.e., rules that are rigidly enforced will be violated less frequently than rules that are not rigidly enforced or are not a part of the CASE tool's rule base. The results indicate that, regardless of the type of methodology support supplied by the CASE tool, there are very few violations of the methodology rules that apply to the internal consistency of a data flow diagram. However, when the system is examined by verifying the hierarchical consistency of the data flow diagrams, the number of specification errors increases. Further, for

some of the methodology rules, the number of violations is proportional to the amount of support provided by the CASE tool. One consequence of these results is that rules applying to the hierarchical consistency of data flows diagrams should be enforced by the CASE tool in as strict a manner as possible to assist in preventing errors from propagating down to the primitive process specifications and corrupting the construction of structure charts.

CHAPTER 1

INTRODUCTION

Software is inherently complex (Brooks, 1987).

1.1 Problem Statement and Research Motivation

It has been estimated that the backlog for developing mainframe information systems is roughly four years (Runyan, 1989). Coupled with this is what Stamps (1987) refers to as the "invisible backlog", i.e., those systems that never get formally requested because users are unable to wait for them. This backlog may be attributed to the fact that maintenance of current information systems consumes up to 80 percent of information systems development resources (Runyan, 1989). With the life-span of application systems estimated to be eighteen months (Kendall, 1977), "... it is critical that the time and effort expended on systems development be minimized in order to reap the benefits of the system as soon as possible and with minimum development cost" (Guimares, 1987, p. 494). Despite this sense of urgency, twenty-five percent of large system development projects are never completed (DeMarco, 1982). The average information system that does get built is one year late and 100 percent over budget (Jones, 1991), and often exhibits many symptoms of information systems quality problems, including unfulfilled requirements, a high learning curve, existence of errors, and maintenance difficulties (Zultner, 1988). Clearly, developing quality information systems while utilizing a prudent amount of resources is one of the major challenges facing the information systems profession.

There are many factors that affect information system quality. Many of these factors pertain to the software component of a system. This is due to the fact that software now constitutes approximately 90 percent of the functionality of a typical information system (Teresko, 1990). McCall, Richards, and Walters (1977) have identified eleven quality factors, which focus on the following three aspects of the software component of a computer information system: its operational characteristics, its ability to undergo change, and its adaptability to new environments. Unfortunately, these factors are too high-level to be meaningful or measured directly (Fenton, 1991). Therefore, a set of quality criterion have been defined and functional relationships established between the quality criteria and quality factors. System specification consistency, or adherence to a development methodology, has been identified as one of the criteria affecting the software quality factors mentioned above. Consistency is defined to be the software attributes that provide uniform design and implementation techniques and documentation (Arthur, 1985). Consistency has been found to directly impact the following software quality factors: correctness, flexibility, maintainability, and reliability.

Computer-aided systems engineering (CASE)[1] tools have been proposed as one of many "silver bullets" to improve system quality and eliminate the development and maintenance backlog of information systems applications (Brooks, 1987). In the past several years many companies have developed CASE products which, they claim, will fully or partially automate one or more phases of the classical systems development life cycle.[2] These CASE vendors tout revolutionary advances in both analyst productivity and system quality; however, there is little empirical evidence to back these claims (Everest & Alanis, 1992; Kemerer, 1989).

One important role of a CASE tool is to serve as a methodology companion, i.e., to assist an analyst in the creation of documentation passed to succeeding phases of the life cycle, and to guide the analyst through a particular systems development

[1] There is no accepted standard for what the "CASE" acronym means. The "A" may be interpreted to be "Aided" or "Assisted" while the "S" may be interpreted to be "Systems" or "Software".

[2] The proliferation of CASE tools is indicated in the following three statistics: 1) A recent CASE product survey shows that there are over 100 vendors offering more than 400 products (Lindholm, 1992); 2) Sales of CASE tools are projected to exceed $3.5 billion in 1993 with an annual growth rate of 25 percent (Nash, 1992); 3) By the middle of the 1990's it is estimated that 50 percent of software engineers will own personal CASE tools (Yourdon, 1992).

methodology (McClure, 1989a). Systems analysts use the development methodologies embedded in CASE tools to support systems development tasks, including communication with the users. These embedded methodologies are capable of producing a high quality product (Chou, Kelley, & Landram, 1992). The level of methodology assistance provided by a CASE tool varies from product to product and may include a graphics tool to support various diagramming techniques, a data dictionary for storing entities associated with a systems project, and automated checks, which serve to enforce a particular methodology and help ensure completeness and consistency of the resulting specifications. For example, if structured analysis is the chosen methodology, and if the analysis specifications are consistent with the chosen methodology, structured design and structured programming techniques (or code generators) may be used to convert the specifications into source code.

Unfortunately, while many vendors claim their product supports a particular information systems development methodology (e.g., Yourdon's structured analysis), the actual level of methodology support varies greatly from one CASE tool to another. The level of support ranges from the simple existence of methodology diagramming symbols to methodology rules that are enforced in real-time.

Teresko (1990) describes the current status of methodology support thusly: "CASE tools impose a development methodology as a primary benefit to developers. Some may dictate very strict methods. Others may be more flexible" (p. 83). However, there are many advocates of CASE who believe the level of methodology support provided by the tools is insufficient. Alavi (1993) suggests that CASE might be more well-received if systems development methodology support was embedded in the CASE tools. Loy (1993) states, "The tools do not yet adequately support the methods as promised" (p. 31). Page-Jones (1992) further adds that the current level of methodology support provided by CASE tools is unacceptable. Yourdon (1992) states that a "CASE tool without methodology support is nothing more than a glorified drawing tool" (p. 144). A lack of methodology support was cited by information systems faculty as one reason why they do not use CASE to support information systems coursework (Jankowski & Norman, 1992). A recent survey of CASE experts indicates that the integration of CASE tools and methodologies must be improved in the future (Crosslin, Bergin, & Stott, 1993), while Henderson and Cooprider (1990) state, "CASE products support a weak level of analysis functionality ... increasing the level of analysis technology holds strong promise" (p. 251). Falkner (1991) identifies the improper use of both CASE and the methodologies automated by CASE as one of the organizational diseases that causes poor information systems quality. The idea of analysts using the systems development methodologies together with the CASE tools is enforced by Crockett, Hall, and Wheeler (1992), who add that, "Through the selective utilization of CASE tools ... information systems design may be optimized in both efficiency and effectiveness. All analysts must be versed in the use of CASE and the methodology being supported by the CASE tool" (p. 983). Sumner's (1993) case study of thirteen firms that use CASE concludes that one of the top two success factors for implementing CASE is the adherence to the chosen methodology.

In this dissertation a framework for classifying CASE implementation of structured analysis methodology rules will be proposed. This framework builds upon previous work in the field of information systems restrictiveness, particularly decision support system (DSS) restrictiveness. Structured analysis methodology rules will be examined and the feasibility of automating them will be discussed. Two best-selling commercial CASE products have been selected for further study. These products have been described as being at opposite ends of the structured analysis methodology support spectrum (Vessey, Jarvenpaa, & Tractinsky, 1992). One of the CASE tools (Visible Systems Corporations's Visible Analyst Workbench Version 3.0) was classified as being "methodologically restrictive" while the second CASE tool (Intersolv's Excelerator Version 1.8) was classified as being "methodologically flexible." The Vessey et al. (1992) research serves to eliminate the notion that methodology support is a binary question, i.e., "Does the CASE tool support methodology X?" Rather, they have shown that methodology support is much more variable than previously believed. However, simply stating that one tool supports a methodology more than another tool does not accurately describe the level of methodology support offered by a CASE tool. The methodology rules supported by Visible Analyst Workbench are, in general, implemented differently than the corresponding methodology rules in Excelerator. Additionally, while the methodology rule base supported by Visible Analyst Workbench is larger than the methodology rule base in Excelerator, there are some rules that Excelerator supports that Visible Analyst Workbench does not support.

The two CASE tools under study have been used to support projects in systems analysis. Each tool was used by eight systems development teams in support of structured analysis. Analysis specifications for a hotel information system were produced and the number of methodology violations has been determined from the specifications. It is hypothesized that the number of violations of a methodology rule is determined by the CASE tool's enforcement of the rule. For example, a methodology rule violation that is automatically flagged as soon as the rule is violated should occur less frequently than if the rule violation is flagged only upon the request of the analyst. Similarly, a methodology rule that is not a part of a CASE tool's methodology rule base should be violated more often than if the rule is a part of a CASE tool's methodology rule base. The research question being posed is:

What effect does the implementation level of a methodology rule have upon the analysis specifications of a system?

1.2 Research Contribution

Wynekoop and Conger's (1991) review of CASE research indicates a lack of evaluative research that examines, among other things, the various levels of methodology support provided by CASE tools. The results of this dissertation will advance the literature concerning CASE tools and their impact on system specification quality, and will provide some insight into how CASE can improve analyst productivity by eliminating much of the labor intensive task of completeness and consistency checking of specifications. By providing designers, programmers and/or code generators with methodologically correct specifications, system quality will be improved. The results are a first step in determining the "optimal" level of methodology support provided by CASE tools. Further, CASE tool selection criteria, of which support for a particular methodology is a particularly important criteria (Amundsen & Christoffersen, 1987; Baram & Steinberg, 1989; Boström, 1988; Burkhard, 1989; Everest & Alanis, 1992; Linos, 1992; McClure, 1989b; Rozman, Györkös, & Rizman, 1992; Shafer & Shafer, 1993; Subramanian & Gershon, 1991; Zucconi, 1989), will be refined by identifying differences in the level of methodology support provided by various CASE products.

1.3 Organization of Dissertation

The chapters of this dissertation are organized as follows: Chapter 2 provides background material regarding the systems development life cycle and structured analysis, and their effects on system quality. The chapter proceeds to introduce CASE technology and explains how CASE automates structured analysis. Next, previous classification schemes for CASE methodology support will be reviewed. Chapter 3 presents the theoretical framework for this research. A theoretical model for systems development is described. A new framework for classifying CASE methodology support will be proposed. Next, the research constructs and variables are outlined and the chapter concludes with the research hypotheses. Chapter 4 describes the research methodology adopted for this dissertation. Chapter 5 presents the experimental results while Chapter 6 discusses the implications of these results as they apply to systems development and CASE technology. Chapter 7 concludes this dissertation with a discussion of the limitations of the experimental study and suggestions for future research opportunities.

CHAPTER 2

LITERATURE REVIEW

2.1 Introduction

Continuous advances in hardware capabilities have enabled increasingly complex systems to become technically feasible. These complex systems have necessitated the introduction of frameworks for managing the systems development process. Together with these frameworks, methodologies are used to help guide an analyst through the life cycle of a system.

This chapter explores the evolution and use of the structured techniques and the underlying systems development life cycle. Particular attention will be paid to structured analysis. The use of CASE tools to aid analysts with the systems development process will also be discussed. In addition, some methods of classifying CASE tool methodology support will be reviewed.

2.2 Systems Development Life Cycle

Purvis and Sambamurthy (1992) provide the following description of systems development:

The goal of systems development is to enable an organization to produce systems that meet cost, schedule, and quality objectives. The underlying premise is that although each project has different characteristics, there exist similarities as well. These similarities justify frameworks for organizing, planning, controlling, and executing projects in a consistent manner that build on the experience of prior systems development efforts (p. 864).

The systems development life cycle was developed and documented in the 1960s to provide defense contractors with a documentation standard for Department of Defense projects (Conger, 1994). The life cycle paradigm grew out of the stagewise model for developing software (Boehm, 1986). The stagewise model contained the following successive stages: operational plan, operational specifications, coding specifications, coding, parameter testing, assembly testing, shakedown, and system evaluation (Benington, 1956/1983). The stagewise model was developed to avoid the "code and go" approach that was prevalent in the early days of high-level programming languages. The model attempted to add structure and discipline to what was, at the time, an undisciplined process.

The idea behind the life cycle paradigm is twofold: 1) the systems development process evolves in distinct phases and 2) each phase is completed before the next phase begins (Ledgard, 1987). The most commonly used model of the life cycle paradigm is the waterfall model (see Figure 1).[3] This model provides an orderly, predictable set of phases, which can facilitate resource allocation and project management (DeGrace & Stahl, 1990). Further, entry and exit conditions for each phase are defined (Carmel & Becker, 1993). Developed in 1970 (Boehm, 1986), the waterfall model added an element of feedback that was missing in the stagewise model. After each phase, feedback is given to the previous phase in order to verify specifications. This feedback is constrained to successive phases. The waterfall model consists of the following phases:

1. Requirements -- In the requirements phase a thorough understanding of the problem is acquired. Users of the system, and their data requirements, are identified as well as the system's inputs, outputs, files, and so on. If an existing system is present, problems with the current system are identified. During this phase a feasibility study may be performed and alternative systems may be proposed.

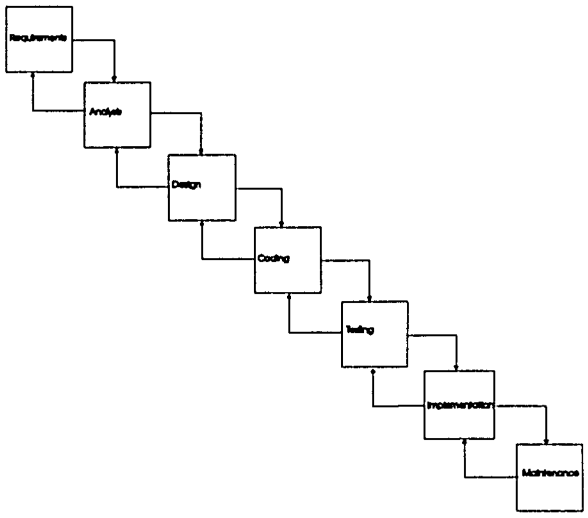


Figure 1 Systems Development Life Cycle -- Waterfall Model

[3] Carmel and Becker (1993) and DeGrace and Stahl (1990) each describe several other life cycle models.

2. Analysis -- In the analysis phase the proposed system's functionality is determined. A high-level view of the system is maintained, concentrating on *what* the new system will do rather than *how* it will do it. Functional components and their data requirements are identified during the analysis phase (Weinberg, 1980; Cougar, 1973).

3. Design -- In the design phase, detailed data structures and algorithms are proposed. File formats as well as the formats for input data screens and output data reports are developed. The logical data identified in the analysis phase is described as a physical implementation.

4. Coding -- The detailed design delivered from the design phase is converted into computer code. Each primitive module identified in the analysis phase, and then refined in the design phase, is coded. The modules are integrated into subsystems and the subsystems are integrated into the software system.

5. Testing -- Testing of the code developed in the coding phase is done at three levels. At the unit level each coded module is tested independently to assure that it performs according to its specifications. After the modules have been tested, integration testing is performed to assure that modules properly communicate with each other. System testing is then conducted until it has been determined that the system meets the originally specified objectives (Beizer, 1990).

6. Implementation -- During the implementation phase the new system is installed on the customer's hardware platform and the system is brought on-line. Implementation may take one of four forms: parallel conversion, direct conversion, phased conversion, or pilot conversion (Long & Long, 1990).

7. Maintenance -- The maintenance phase is on-going for the life of the system. Maintenance activities include the diagnosis and correction of errors (corrective maintenance), the modification of the system to interface with a changing environment (adaptive maintenance), the enhancement of system functionality (perfective maintenance), and the modification of the system to improve future maintainability (preventive maintenance) (Pressman, 1992).

2.2.1 Use of the Life Cycle

Unfortunately, few studies have been undertaken comparing the life cycle paradigm (and the waterfall model) with other systems development paradigms, such as prototyping (Mahmood, 1987). Further, some of the results are contradictory in nature. For example, Mahmood's (1987) study of 61 matched pairs of designer/user interviews found that user satisfaction with a system is greater when the system is developed under the life cycle rather than with prototyping. This contradicts both the Boehm, Gray, and Seewaldt (1984) and Alavi (1984) studies, which found that user satisfaction is greater if a system is developed using the prototyping paradigm rather than the life cycle. Teng and Sethi's (1990) experiment using doctoral students as users and designers found user participation in the design process to be greater when using the life cycle as opposed to prototyping. This finding concurs with Boehm et al.

(1984) but conflicts with Mahmood (1987). The Boehm et al. (1984) study also found that project teams using the life cycle produced more coherent designs and systems that were easier to integrate than did teams using prototyping. Clearly, the evidence supporting one particular systems development paradigm over another is inconclusive.

Regardless of the lack of conclusive empirical evidence in support of any particular systems development paradigm, the life cycle paradigm is clearly being utilized in the professional world. Necco, Gordon, and Tsai (1987) examined systems development practices in 97 organizations and found that nearly all of the organizations use the waterfall model. They conclude, "Organizations will continue to develop their information systems within the framework of the systems development life cycle" (p. 472). Guimaraes's (1985) survey of systems development practices in 43 organizations concludes by recommending the life cycle paradigm for the development of systems that will have a "long life." This includes transaction processing systems, well-defined decision support systems, large systems, and systems that primarily serve one functional area in a business. This recommendation is reinforced by Palvia and Nosek's (1990) survey of MIS professionals, which found the life cycle to be preferred for transaction processing systems, management information systems, and well-structured problems. Palvia and Nosek conclude by stating, "The life cycle has high use and applicability during all stages of systems development" (p. 27).

2.3 Structured Techniques

The waterfall model of the systems development life cycle paradigm serves as a framework for developing systems. However, it does not provide the systems analyst with a process for determining systems requirements, analyzing, designing, coding, or testing a new system. Moher and Schneider (1982) describe the early period of software engineering as "a discipline dominated by points of view ... based on personal observation and intuition, which grew to become 'rules of thumb'" (p. 65). Since that time, several methods and collections of methods (methodologies) have been developed that apply to one or more phases of the systems development life cycle. Everest and Alanis's (1992) study of systems development practices in seventeen organizations found that methodologies were used for the following reasons:

1. Improved system quality
2. Improved communication between end users and developers.
3. Reduced IS development effort.
4. Reduced maintenance effort.
5. Introduce discipline and structure.
6. Better documentation.
7. Better tracking of user requirements (p. 349).

An organization chooses a systems development methodology based upon the following criteria:

1. Coverage of a broad range of the life cycle.
2. Compatibility with current in-house practices.
3. Support by automated tools.
4. Ability to modify or customize for in-house variations.
5. Ease of use.
6. Wide-spread use (Everest & Alanis, 1992, p. 349).

The most commonly used set of systems development methodologies are collectively known as the structured techniques (see Figure 2).[4] The structured techniques were developed in an attempt to change systems development from an informal craft to an engineering-like discipline. By the middle of the 1960s, the ability to manage the systems development process could not meet the need for increasingly complex systems (Colter, 1982). During the later part of the 1960s and through the 1970s, computer-aided techniques such as Problem Statement Language/Problem Statement Analyzer (PSL/PSA) (Teichroew & Hershey,

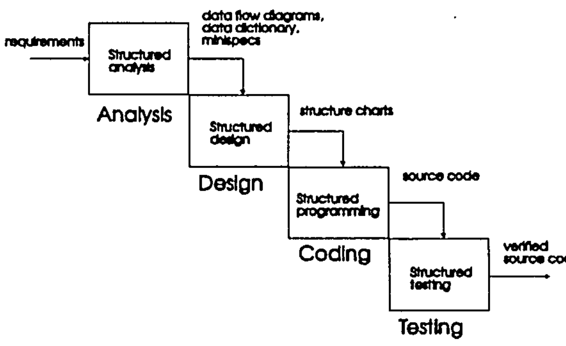


Figure 2 The Structured Techniques

[4] *The Second Annual Report on CASE*, a survey conducted in 1990, indicates that the structured techniques are used more than any other systems development techniques by a factor of approximately 5 to 1 (Yourdon, 1992).

1977) were introduced in an attempt to automate portions of the systems development life cycle. These computer-aided techniques did not catch on as expected due to their high level of sophistication. Rather, a new set of manual techniques called the structured techniques (or structured methodologies) was introduced to provide an easier transition to the computer-aided techniques (Couger, 1982).

The initial work toward developing a set of methodologies to apply to the waterfall model was confined to the familiar arena of computer programming. Structured programming is based upon the concepts of modularity, i.e., the idea that a system consists of a set of small components (modules), and stepwise refinement, i.e., the process of partitioning a system into its component modules. Additionally identified were the primitive constructs (sequence, selection, and iteration) that are necessary to implement any logical process. At the same time, structured testing techniques were developed, which enabled the system to be verified in a modular, hierarchical fashion.

It was soon realized that a well-coded implementation of a poor design did not solve any problems. This led to the introduction of structured design methodologies that sought to produce a good design from a set of requirements or analysis specifications.[5] Understanding the need for a complete and consistent specification of the logical system led to the introduction of structured analysis (Colter, 1984).

2.3.1 Principles of the Structured Techniques

The fundamental objective of the structured techniques is to produce high-quality systems that are easy to maintain. Further, these high-quality systems should be built as quickly and as cheaply as possible. These objectives are met by adhering to four philosophies: the principle of abstraction, the principle of formality, the divide-and-conquer concept, and the hierarchical ordering concept (Martin & McClure, 1988). The principle of abstraction dictates that the system be simplified by concentrating on *what* the system is to do rather than *how* the system does it. The principle of formality suggests that a "rigorous, methodical approach" (Martin & McClure, 1988, p. 17) be followed, i.e., to treat systems development as an engineering-like discipline. The divide-and-conquer concept treats a system as a set of smaller systems. Each of these smaller systems, or modules, is inherently less complex than the entire system. The hierarchical ordering concept is a method for arranging the system modules into a tree-like structure.

[5] For a thorough treatment of structured design and its relationship with structured analysis see Yourdon and Constantine (1979).

2.3.2 Structured Analysis

Early attempts at developing structured methodologies for systems development concentrated primarily on technical aspects related to coding and testing. However, a system that is technically sound may not satisfy user requirements due to errors made in the early phases, especially analysis, of the systems development life cycle. Any errors in the analysis phase may propagate through design, coding and testing.

The consequences of neglecting analysis are well documented. Kapur's (1986) study of system maintenance concluded that 82% of all changes made after programming has begun can be traced to specification omissions, ambiguities, and errors. It is estimated that a specification error may be 100 times more expensive to correct when detected in the testing phase than if it had been detected and corrected in the analysis phase (Haase & Koch, 1982). Yourdon (1992) estimates that an error in phase "N" of systems development is ten times cheaper to detect and correct in phase "N" than if it is allowed to propagate to phase "N+1". In a study of system errors detected during or after testing, only one third of the errors were attributed to coding mistakes (Boehm, McClearn, & Unfrig, 1975). McKeen's (1983) study of 32 business application systems found that projects that spend more time in the analysis phase were more successful with respect to both user satisfaction and completing the system within the budgeted time. Khailany, Sanchez, and Lee's (1985) study of 38 data processing companies found that 23 percent of maintenance costs can be attributed to changing requirements to reflect the original system goals or needs. The study also indicates that analysts perceive that a better system definition and specification will lead to reduced maintenance costs. Failure to adequately complete a system specification is cited by Ginzberg (1981) as a leading cause of system failure. Boehm (1976) paints a gloomy picture regarding the consequences of neglecting analysis and the resulting analysis specifications. He notes:

1. Only 40 percent of software errors can be traced to the coding phase.
2. A review of systems analysis specifications will typically reveal one to four nontrivial errors per page.
3. Top-down design is impossible due to an inadequately defined "top".
4. There is nothing to test against.
5. Users have no clear statement of what is being produced for them.
6. Management has no clear statement of what the project team is working on (p. 1227).

Further justification for complete analysis specifications is provided by Golden, Mueller, and Anselm (1981) who state, "Good requirements and system specifications are necessary in order to estimate system size and development costs" (p. 14). Despite the evidence concerning the importance of developing a correct system specification, Khailany et al. (1985) report that only 5 percent of the life cycle (time and cost) is devoted to defining and developing the system specification.

The need for complete and consistent analysis specifications led to the development of structured analysis. Structured analysis is a systematic, step-by-step approach to performing systems analysis and producing a system specification. Structured analysis techniques were primarily devised by Ed Yourdon, Chris Gane, and Trish Sarson in the middle of the 1970s. From their work, two similar sets of structured analysis methods (the differences, largely cosmetic, are described in Gane, 1990) have emerged: Yourdon structured analysis and Gane & Sarson structured analysis. The output of structured analysis, the system specification, is passed to the design phase of the systems development life cycle. The system specification consists of data flow diagrams, a data dictionary, and primitive process specifications (minispecs) (Yourdon, 1989). The components of the system specification are briefly discussed in the remainder of this section.

2.3.2.1 Data Flow Diagrams

Adhering to the principles discussed in Section 2.3.1, structured analysis uses the divide-and-conquer and hierarchical ordering concepts to graphically represent the functionality of the proposed system. In particular, the emphasis is on the data that flows through the logical components of the system and is transformed from input data to output data. A set of data flow diagrams is created by first analyzing the system from a high-level perspective. The resulting diagram, called a context diagram, views the system as a single entity. Most importantly, the context diagram establishes a boundary between the system under study and the rest of the world. Along with the system, the context diagram displays the external entities, or terminators, that will give input to, and receive output from, the system.

After the system boundary has been established, and external entities and their data requirements have been defined, the system is decomposed into its primary subsystems using a top-down approach. A new diagram, denoted as Level 0, is created that displays the primary subsystems (major activities) as well as their data requirements. The Level 0 diagram is derived from the context diagram through event partitioning (McMenamin & Palmer, 1984) and leveling (see Figure 3).[6] At the next level down, the data flow diagrams depict the working of each process in the level 0 diagram. At still lower levels, the data flow diagrams show the workings of the processes in more detail. Each process is continually decomposed until it reaches the point where it is in its most primitive form, i.e., a process that could be represented by one page of computer code (Intersolv, 1989).

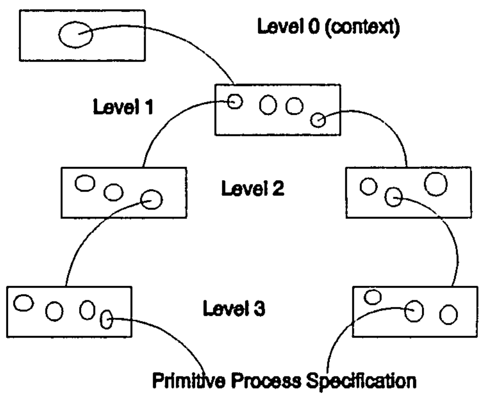


Figure 3 Leveling a Data Flow Diagram

[6] For a thorough discussion of the leveling process see Yourdon (1989).

2.3.2.2 Primitive Process Specifications

A primitive process specification, or minispec, describes what happens inside a process symbol on the lowest level of a data flow diagram (Martin & McClure, 1988). Specifically, the process specification details how the input data are transformed into output data. Process specifications typically consist of a one page (or shorter) structured English description of the data transformation with both inputs and outputs from the corresponding data flow diagram process being clearly referenced.

2.3.2.3 Data Dictionary

Along with the set of data flow diagrams and primitive process specifications, a data dictionary is maintained which organizes the system's data elements. The data dictionary (sometimes referred to as the data repository or data encyclopedia) is a centralized repository for system components and contains specifications that describe and influence the system being modeled (Gibson, 1989). By centralizing project data, changes to any part of the system can be propagated to the effected areas. The contents of the data dictionary may include: file specifications, record definitions, data element definitions, data relations, external entity descriptions, process specifications, input data screens, and output reports (Rob & Coronel, 1993).

2.3.3 Research on the Use of Structured Analysis

The bulk of the literature that reports on the use of the structured techniques is primarily confined to structured programming. This is due to some difficulties inherent in software engineering experimentation which serve to limit the number of experiments that have been conducted (MacDonell, 1993; Moher & Schneider, 1982). Most of the studies investigating systems development are performed post-hoc, i.e., projects are chosen for inclusion in a study well after their completion (see, for example, Brooks, 1981, or Card, McGarry, & Page, 1987). As a result, much of the collected data is based on retrospect rather than concurrent observations made during the development period of the system. The controlled experiments that have examined the structured techniques by using a replicated project (see, for example, Basili & Reiter, 1981, or Benander, 1990) typically involve small, student programming projects. Basili and Reiter (1981) acknowledge this problem by stating that there is a trade off between toy experiments, which enable a large sample size but involve programs consisting of 30 lines of code, and production experiments, which enable a great degree of realism but utilize a weak experimental design and almost no chance for replication. Boehm (1981) points out the difficulty of making reliable statistical inferences based on data collected from production projects. DeMarco (1982) suggests, however, that any data is better than no data.

While the empirical studies of programming indicate that more efficient and less error-prone code is the result of using structured programming, as opposed to a "spaghetti code" approach, the few empirical studies of structured analysis are less conclusive. Along with the difficulties of software engineering research described in the preceding paragraph is the further complication that many analysts and/or organizations simply do not keep documentation relating to the initial phases of a systems development project. As a result, empirical studies concerning the effects of the presence or absence of a particular methodology, tool, or technique may rely on the memory of the analysts, i.e., their ability to recall the likely effects if similar projects from the past had also been carried out with the use of a particular methodology (Lempp & Lauber, 1989). Mann (1992) indicates that the issue is further complicated by a lack of definitional standards for terms such as "methodology" and "life cycle", as well as a failure to identify appropriate metrics for experimental studies.

The above difficulties in performing software engineering research are made evident by the results of literature reviews of systems development methodology research. Both Mann (1992) and Purvis and Sambamurthy (1992) turned up no instances of studies comparing the structured techniques to the absence of any methodology. However, conclusions made in some of the structured programming studies lend credence to the idea that the structured techniques must be utilized from the outset of a systems development project in order to obtain a quality system. For example, Card et al.'s (1987) study of 22 software systems concludes that all phases of the systems development process must be effectively documented and that this documentation improves software reliability. Card, Church, and Agresti's (1986) study of five large software projects reaches conclusions regarding system module size, strength, coupling, and descendants that are consistent with the principles of the structured techniques described in Section 2.3.1.

Despite the lack of experimental evidence regarding the use of structured analysis, it is clear that many analysts and organizations view structured analysis as the best technique for handling the analysis phase of the life cycle. Several surveys of systems development practices of individual analysts and organizations reveal that structured analysis is extremely popular. Necco et al.'s (1987) survey of 97 organizations indicate that 62 percent of the organizations use structured analysis and an additional 20 percent of the non-users were considering adopting structured analysis. The survey also reveals that 94 percent of the users of data flow diagrams find them to be at least "adequate" as a tool for systems analysis. Similar figures of 95 percent and 94 percent were given for users of a data dictionary and process specifications. The study concludes that, "Structured approaches will increasingly supplement traditional/classical approaches to develop computer-based information systems" (p. 472). Beck and Perkins's (1983) survey of systems development practices in 63 firms indicates that 75 percent of the firms use data flow diagrams. Further, data flow diagrams are found to be the most successfully used documentation aid for determining and analyzing the user's requirements. Carey and McLeod's (1988) survey of systems development methodology and tool usage at 121 firms indicates that data flow diagrams are used by 79 percent of the firms, more than any other tool. This level of usage is consistent with that reported in Khailany et al. (1985). Kievit and Martin's (1989) survey of systems analysts finds that 45 percent of the analysts use a data dictionary at least "most of the time" while 90 percent of the analysts use data flow diagrams at least "sometimes." Additionally, the analysts report to be at least "satisfied" with data flow diagrams 83 percent of the time, more than with any other analysis tool. Sumner and Sitek's (1986) survey of 36 firms indicates that both data flow diagrams and the data dictionary are popular with the systems analysts. Perceived benefits of the tools, as indicated by the analysts, include: better requirements analysis, better user understanding and participation, increased user satisfaction, ease of maintenance, and more flexible systems. Sumner's (1993) case study of systems development practices in 28 firms reports that structured analysis is the most widely used development methodology and data flow diagrams are the most popular analysis and design tool. Rozman et al. (1992) surveyed twelve analysts who use multiple systems development methodologies to support analysis and design. The structured techniques are preferred by eleven of the twelve analysts. Structured analysis and structured design are specifically cited as being "easy to learn, simple, and efficient" (p. 45). A 1987 survey of over 1000 firms, as reported in McClure (1989a), indicates that structured analysis is the most popular methodology for systems analysis, and that more firms planned to switch to structured analysis than any other methodology. Jankowski and Norman's (1992) survey of information systems faculty indicates that structured analysis is being taught in the information systems curriculum more than any other analysis methodology. Finally, Sakthivel's (1991) survey of 27 information systems professionals indicates that structured analysis provides uniform support for all factors (versatility, functionality, productivity, and efficiency) associated with choosing a systems development methodology. Despite the lack of empirical evidence that states that structured analysis is the definitive technique for constructing the system specification, structured analysis remains the technique of choice for information systems professionals.

2.3.4 Drawbacks of Structured Analysis

Despite surveys that indicate the popularity of structured analysis as a methodology for performing systems analysis, there are many analysts and organizations that have abandoned the methodology. Guimaraes's (1985) survey of 43 companies found that only six were using structured analysis. Two of the six were discontinuing usage due to the fact that "it does not seem to be as user friendly as it has been promised to be" (p. 497). Structured analysis has been criticized as being "unwieldy and time consuming" (Sumner & Sitek, 1986, p. 18), and too slow and expensive (Henken, 1988). Yourdon (1986) and Chapin (1979) both indicate that the primary reason for rejecting structured analysis is the frustration encountered by analysts when faced with manually checking data flow diagrams for internal consistency. These complaints helped contribute to the introduction of automated tools to support systems development and, especially, the structured techniques.

2.4 Computer-Aided Systems Engineering

Yourdon (1986) makes quite clear the general frustrations that can occur when using structured analysis to create the analysis specifications:

> Any reasonable systems analyst is willing to draw a diagram once. Drawing a hundred such diagrams is a different matter, especially because each diagram typically has to be revised and redrawn several times as the analyst and user change the model. To further complicate matters, the diagrams must be kept consistent with the data dictionary and primitive process specifications (p. 133).

Clearly, a new technology was needed to reduce the labor-intensive nature of structured analysis.

2.4.1 What is CASE?

While tools to automate the systems development life cycle first appeared in 1965 with the introduction of PSL/PSA (Dennis, George, Jessup, Nunamaker, Vogel, 1988), the term "CASE" did not come into use until the appearance of computer-aided diagramming and documentation tools in the early 1980's. Computer-aided systems engineering (CASE) tools were developed in order to automate all phases of the systems development life cycle, with an emphasis being placed on making the structured techniques, and especially structured analysis, more feasible (McClure, 1989a). Automation of the structured methodologies evolved out of a natural maturing of the software engineering discipline (Bordoloi, Courtney, Paranusiwaean, 1992). Early proponents of CASE argued that the ability to easily and cheaply perform more detailed analysis would lead to an emphasis on the initial phases of the life cycle, which would increase the probability of a more correct analysis specification and decrease the probability of errors appearing in later phases of the development process (McDermid, 1990).

Unfortunately, the systems development literature has not settled on a standard definition for CASE. It is frequently described as software (Davis, 1987), a technique, a technology, a mechanism (Glass, Hughes, Johnston, & McChesney, 1989), a tool, a toolkit, a workbench, a platform (McClure, 1989a), a methodology (Teresko, 1990), and a philosophy (Gibson, 1989). Further distinctions are now being made between CASE and integrated CASE (I-CASE), which claims to automate the entire development process (primarily in support of the life cycle paradigm), and upper-, middle-, and lower-CASE, which apply to particular phases of the life cycle. The following definition will be used by the author when referring to CASE: *Any software tool that automates a portion of the systems development process.*

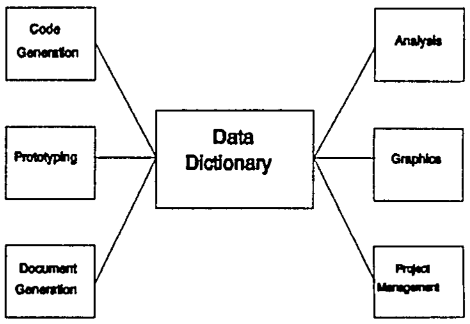


Figure 4 CASE Tool Architecture

While the above definition implies that stand-alone tools such as a compiler or text editor qualify as CASE tools, the definition is more traditionally applied to tools that are designed around an automated data dictionary (see Figure 4) and support one or more systems development methodologies. As defined in Section 2.3.2.3, the data dictionary serves as a repository for the system entities. Other components of the CASE tool have access to the dictionary in such a way that any change to the data dictionary propagates to the rest of the tool. Windsor (1986) states that this architecture allows for changes to be made easily to the system as well as alternative systems to be investigated.

2.4.2 Benefits From Using CASE

Many academics and information systems professionals believe that CASE can improve information system quality. Intuitively, it would appear as if using CASE tools to support structured analysis would alleviate any problems analysts may have previously had with the technique. There are many CASE tools that claim to support the Yourdon or Gane & Sarson structured analysis methodology. By supporting this methodology, the CASE tools should make it much easier to draw the complex set of data flow diagrams associated with a system specification. Further, the existence of an integrated data dictionary should allow for easily checking the completeness and consistency of the diagrams vis a vis the data dictionary elements and the minispecs. This is supported by Sumner's (1993) study of thirteen firms that use CASE, which reveals that analysis is supported by CASE more than any other life cycle activity and that data flow diagram and data dictionary construction are the most commonly used analytical tools supported by CASE.

Unfortunately, there is very little empirical evidence that shows that CASE tools can improve the productivity of a systems analyst and/or improve the quality of the resulting system (Everest & Alanis, 1992; Kemerer, 1989). The majority of the evidence that does exist has not been obtained via rigorous experimentation and, instead, relies on the perceptions of systems analysts. Much of the evidence in support of CASE is anecdotal in nature (see, for example, Shultz, 1989)[7] and is supplied by the CASE vendors themselves. The problems with collecting experimental evidence in support of CASE are similar to those discussed in Section 2.3.3 for the structured techniques. Despite the lack of conclusive evidence regarding CASE's impact on systems development, Necco et al.'s survey of systems development practices concludes, "Automated tools will be used increasingly to support specific systems analysis and design tasks" (p. 473). McGaughey and Gibson (1990) reviewed the information system development literature and have identified CASE as having the potential to improve information systems performance in the areas of effectiveness, reliability, and efficiency. The remainder of this section examines the impact CASE can have on IS effectiveness, reliability, and efficiency.

2.4.2.1 CASE's Impact on Information System Effectiveness

Effectiveness is the achievement of desired results over time. For an information system, the term "effectiveness" can be applied to the quality of the information being presented by the system to the end-users. Necco, Tsai, and Hogelson's (1989) survey of fifteen organizations using CASE found that CASE contributed to a "significant" improvement in system quality in 60 percent of the organizations and a "moderate" improvement in system quality in the remaining 40 percent. Everest and Alanis's

[7] Publications such as Datamation and Information Week often contain recollections of successful and unsuccessful applications of CASE technology.

(1992) study found that 100 percent of 19 organizations surveyed justified their acquisition of CASE by their expectations for an improved end product.

A lack of communication between developers and users during the initial phases of the life cycle is frequently cited as a major cause of errors in newly installed systems (Andrews, 1983; Cerveny, Garrity, & Sanders, 1985). CASE may contribute to IS effectiveness through improved communication between the analysts and users (McGaughey & Gibson, 1990). Necco et al.'s (1989) study found that 100 percent of organizations surveyed reported at least a "moderate" improvement in communications between analysts and users, while 87 percent of the organizations reported at least a "moderate" improvement in communications between analysts. Stamps (1987) reports that Du Pont realized greater user/analyst communication due to the graphical nature of the CASE tool being employed. Everest and Alanis's (1992) study found that 67 percent of the organizations surveyed experienced "a positive change in the established working relationship with end users" (p. 350) by using CASE tools.

There is only one experimental study that quantitatively evaluates the influence of CASE technology on system quality. Granger and Pick (1993) had eleven student teams develop a 1500-2000 line Pascal print program. Four of the teams used CASE to develop their program; the remaining seven teams did not. Each of the eleven teams received an identical set of requirements. The teams using CASE applied the technology to the analysis and design phases of the lifecycle. The final source code was compared to determine the influence of the presence of CASE on system complexity, size, and completeness. Systems developed by CASE users were more complete, i.e., met the user requirements, than the systems developed by non-CASE users. There was no statistical significance between the CASE systems and the non-CASE systems for system complexity or system size. While the results of the sole experimental study are not conclusive, organizations are perceiving that their system quality is improving by using CASE.

2.4.2.2 CASE's Impact on Information System Reliability

Reliability is the extent to which a system performs in the manner intended by the designers and users, i.e., the degree to which it is complete, consistent, and correct (Martin & McClure, 1988). Improved system reliability is cited in Everest and Alanis's (1992) study as a justification for acquiring CASE tools. Rowe's (1993) survey of 76 information systems groups finds that analysts who use CASE conform to a methodology and have standards more rigidly enforced than those without CASE. Lempp and Lauber's (1989) study indicates that 67 percent of the project managers surveyed report finding fewer errors in the later stages of the development effort due to better analysis specifications created with the aid of CASE tools.

Three experimental studies have been performed to determine the impact of CASE on the completeness and correctness of analysis specifications. Baram, Steinberg, and Nosek (1990) looked at the data flow diagrams drawn by 26 information systems students. Fifty percent of the students drew their diagrams with the aid of a CASE tool. The resulting diagrams were analyzed for their correctness, completeness, balance, and readability. The diagrams drawn with the aid of a CASE tool were more correct than those drawn without a CASE tool. However, the diagrams drawn without a CASE tool were more complete than those drawn with a CASE tool. There was no statistical significance for either balance or readability. The authors attribute the lack of a major performance difference to the simplicity of the required diagrams.

In the second experimental study, Yellen (1990) examined the impact of CASE on the creation of both data flow diagrams and the data dictionary. Thirty information systems students were provided with a business description and required to produce a context diagram, level 0 diagram, and all related data dictionary entries. Fifty percent of the students used a CASE tool to support their work. The resulting specifications were analyzed for correctness, completeness, and communicability. The specifications completed with the aid of the CASE tool were found to be more correct than those completed manually. There was no statistical significance for the attributes of completeness and communicability.

The results of the third experimental study (Frolick, Wilkes, & Rainer, 1993) contradict those reported in the preceding two studies. The authors had 31 information systems students produce a small set of deliverables (a context diagram, four level one diagrams, a report specification, and a file specification) over a three hour period. Approximately half of the students used a CASE tool to create their deliverables while the other half manually created their deliverables. The authors found that the overall quality of the manually produced specifications was better than the overall quality of the specifications produced with the aid of a CASE tool. The authors replicated this study and observed the same results. The authors acknowledge that three hours may not have been sufficient time to conduct the study, as many of the subjects who used a CASE tool experienced a great deal of frustration while using the tool. While the three experimental studies that examine reliability are inconclusive (in part due to the simple nature of the experimental task), surveys of organizations that use CASE indicate that system reliability is increased by using CASE tools.

2.4.2.3 CASE's Impact on Development Efficiency

Efficiency is the achievement of desired results with a minimum of resources. Sumner's (1993) case study of thirteen firms that use CASE and fifteen firms that do not use CASE examines the perceived benefits of CASE technology. The thirteen firms that use CASE report that CASE makes the structured techniques feasible and allows them to quickly redraw diagrams. The fifteen non-CASE users perceive CASE as having the capability to provide several tangible benefits, including making the structured techniques feasible, creating a centralized data dictionary, easily redrawing diagrams, and adhering to methodology standards. Necco et al.'s (1989) survey finds that 47 percent of the firms using CASE experience a "significant" gain in productivity, while the remaining 53 percent of the firms using CASE experience a "moderate" gain in productivity. The productivity impact reported in Necco et al. (1989) is seen in such areas as the ability to make more changes to a design, the likelihood of future maintenance being easier, and the ease of developing graphical representations. Finally, the survey reports that 93 percent of the organizations found systems development to be "more enjoyable" when using CASE tools. A Price Waterhouse survey reported in (Statland, 1989) indicates that over 50 percent of the CASE users surveyed anticipate productivity enhancements of at least 100 percent to occur within five years of adopting CASE. When determining how firms rationalize their acquisition of CASE, Everest and Alanis (1992) found that 100 percent of the firms anticipate reduced IS development efforts and 84 percent anticipate reduced maintenance efforts. Thirty-two percent of the firms surveyed indicate that using CASE will enable them to more easily reuse software components. Pallatto (1989) reports the results of a survey of nineteen firms using CASE, which concludes that the most significant benefit derived from CASE is the labor savings during software maintenance. McClure (1989a) performed three case studies using Intersolv's

Excelerator CASE tool.[8] In the first case study, a United States government organization reported that structured analysis specifications that previously took two years to complete manually could be completed in four months with the use of a CASE tool. In the second case study, ARCO reported a 10-to-1 gain in productivity by using Excelerator to perform data modeling. In the third case study, Touche Ross used Excelerator to assist in the creation of structured analysis specifications and found productivity to be increased due to the ease of revising diagrams. Norman and Nunamaker (1989) surveyed 91 users of the Excelerator CASE tool in order to determine the functions of the CASE tool that provide the greatest increase in analyst productivity. The study reveals that the graphics and data modeling tools offered by Excelerator are all perceived by the analysts to improve their productivity. In particular, in preference rankings, the data flow diagram tool and the data dictionary were selected more often than any other function. Lempp and Lauber's (1989) survey of 22 projects in fourteen organizations reveals that all phases of the systems development life cycle are felt to be easier when using CASE tools. One particular result of note is that 75 percent of the system documentation is done automatically with the CASE tools. Freeman's (1993) on-going case study of 20 information systems projects (10 projects were supported by CASE; 10 projects were not supported by CASE) finds that the projects using CASE tools are implemented with

[8] At the time of the case studies the Excelerator CASE tool was a product of Index Technology.

only 2 percent more function points than initially estimated after analysis. Projects that did not use CASE are implemented with 12 percent more function points than initially estimated after analysis. While experimental studies examining CASE's effect on development efficiency have not been conducted, there is data available from many case studies and surveys that indicates that CASE has a positive impact on information system development efficiency.

2.5 CASE as a Methodology Companion

Despite the beginnings of a base of research that indicates that CASE offers improvements in system quality and analyst productivity, there are still many professionals and academics who are reluctant to commit to CASE technology. The literature cites many problems associated with CASE technology, including a steep learning curve, the expense of acquiring CASE tools, the expensive hardware platforms required to run the tools, and the lack of integration between tools. Another frequently cited complaint with CASE tools is the lack of methodology support provided by the tools (Alavi, 1993; Crosslin, Bergin, & Stott, 1993; Henderson & Cooprider, 1990; Jankowski & Norman, 1992; Loy, 1993; Page-Jones, 1992).

It is important that the CASE tool and the systems development methodology be fully integrated to avoid "rule bending" or "tool bending" (Boström, 1988). Rule bending occurs when a CASE tool does not adequately support a particular methodology, allowing the analyst to incorporate individual modifications to the methodology notation, syntax, and/or process. Tool bending occurs when a CASE tool is chosen that is inappropriate for the task at hand. Unfortunately, the CASE tool vendors provide little support in helping the analyst and organization determine if a particular tool and methodology are integrated. For the most part, CASE methodology support has been a binary decision, i.e., "Does CASE tool X support methodology Z?" For example, a 1991 survey of 45 CASE products reveals that 23 of them claim to support Yourdon's structured analysis while 22 of the products claim to support Gane & Sarson structured analysis (DBMS, 1991).[9] The extent to which the process and product of structured analysis is supported by these tools is not provided by the vendors. To further complicate matters, six tools that claim to support structured analysis do not support data flow diagrams and six tools that support data flow diagrams do not support structured analysis. Clearly there exists a need for a better classification of methodology support as provided by a CASE tool.

2.5.1 Classifying CASE Methodology Support

CASE is able to provide support for a systems development methodology by automating the methodology's process steps and by assisting in the preparation of a methodology's products (required documentation) (McClure, 1989a). Automation of

[9] $Y \cup G/S = 26$; $Y \cap G/S = 19$.

documentation may be accomplished by supporting, for example, the symbol set associated with a particular diagramming technique. Process support may be accomplished by enforcing a methodology principle such as top-down design by requiring the analyst to begin a set of data flow diagrams with a context diagram.

The systems development literature contains very little discussion on defining and/or determining the level of methodology support provided by CASE tools. Boström (1988) proposes classifying CASE tools based upon the number of life cycle phases supported by the product: "one phase, some phases, most phases, or all phases" (p. 3). Bordoloi et al. (1992) propose a three-dimensional framework for evaluating CASE tools, which uses the phases of the life cycle as one of the dimensions. Amundsen and Christoffersen (1987) propose classifying CASE tools by examining the level of integration the tools provide between the life cycle phases, i.e., "Can each phase be separately automated or can the entire life cycle be automated?" (p. 2). However, all three of the above classification schemes fail to take the issue of methodology support beyond that of a binary question. Further, none of the proposed classifications actually addresses the issue of methodology support; all are, instead, examining life cycle support.

Along with the above mentioned life cycle support classification, Boström (1988) has proposed a ranking system for CASE tools based upon their level of methodology support:

1. Drawing tools where the symbols cannot be put into methodological context.
2. Rule-based tools that can check the validity and consistency of a single document.
3. Rule-based tools that can check an entire set of documentation for validity and consistency.
4. Tools that allow the analyst to input new rules (p. 2).

While this classification does specifically address development methodologies, as opposed to models or paradigms, it does not go far enough toward helping an analyst determine the true level of methodology support offered by the CASE tool. The distinction between items 2 and 3 in the above list is minor and the existence of item 4 would almost certainly lead to the "rule bending" mentioned previously.

A classification similar to the one above is described in (McClure, 1989a). The author proposes three levels of knowledge that can be supported by CASE: documentation, guidance, and process. Documentation-level knowledge is equivalent to item 1 in Boström's ranking. Guidance-level knowledge is a combination of Boström's item 2 and item 3. Process-level knowledge can be described as knowing how to get started, what to do next, and when you are finished. While McClure's classification scheme certainly addresses methodology support, it is essentially a ternary scale providing little information beyond that already provided by answering the binary question, "Does CASE tool X support methodology Z?"

A combination of the classifications described above is suggested by Page-Jones's (1988) description of a well-planned CASE tool. Page-Jones describes a CASE tool as an "onion" (see Figure 5), with an outer layer being defined as "a structure of a set of items from the next (inner) layer" (p. 346). For example, a deliverable such as a data flow diagram may be described as a set of objects: processes, data stores, data flows, and external entities. Page-Jones suggests that when choosing a CASE tool the organization should decide which layers are the most important. However, the onion model still treats methodology support as a binary question.

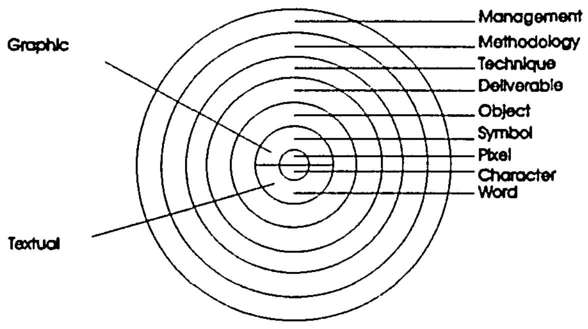


Figure 5 Layered Structure of a CASE Tool (Page-Jones, 1988)

The first substantial work aimed at classifying CASE tools based upon their level of methodology support is presented in a study by Vessey et al. (1992). Using terminology taken from the decision support system (DSS) literature, the authors describe methodology support as being either restrictive, guided, or flexible. A restrictive CASE tool is described by Vessey et al. as being "designed to encourage the user to use it in a normative manner" (p. 92). A restrictive CASE tool is characterized by checks for methodological consistency that automatically occur while the specification item (e.g., a data flow diagram) is being created. If a violation is discovered by the tool the analyst must correct the violation before proceeding. A guided CASE tool is described by Vessey et al. as being "designed to encourage, but not to enforce, the user to use it in a normative way" (p. 92). A guided CASE tool is characterized by checks for methodological consistency that automatically occur while the specification item is being created. However, if a violation is detected by the tool the analyst may ignore the error and proceed. A flexible CASE tool is described by Vessey et al. as being "designed to allow the user complete freedom in using it" (p. 92). A flexible CASE tool is characterized by checks for methodological consistency that occur at the request of the analyst and then only after work on the specification item has been completed. All methodology violations detected by the CASE tool may be treated by the analyst as warnings. Table 1 summarizes the categorizations of methodology support proposed in Vessey et al. (1992).

Table 1 Methodology Checks for CASE Tools with Different Philosophies (Vessey, Jarvenpaa, & Tractinsky, 1992)

	When	**How**	**What**
	creation/technique/phase	auto/request	error/warning
Restrictive			
process	creation	automatically	error
hierarchical consistency	creation	automatically	error
internal consistency	creation	automatically	
Guided			
process	creation	automatically	warning
hierarchical consistency	creation	automatically	warning
internal consistency	creation	automatically	warning
Flexible			
hierarchical consistency	technique or phase	request	warning
internal consistency	technique or phase	request	warning

After defining their classification scheme, Vessey et al. apply it to twelve PC-based CASE tools. Each CASE tool was examined with respect to a set of methodology rules that apply to analysis and design specifications. Nineteen rules are identified for data flow diagrams and four rules are identified for minispecs.[10] CASE tools were labelled as "restrictive", "guided", or "flexible" based upon the number of rules implemented by the tool and the manner in which they are implemented.

[10] For the purposes of data analysis the authors examined seventeen of the nineteen data flow diagram methodology rules. None of the results of the survey concerning primitive process specifications, data structure diagrams, structure charts, or entity/relationship diagrams were reported.

While the scheme described in Vessey et al. (1992) is a positive step toward classifying CASE methodology support, there are several areas where there is room for improvement. First, the authors are using Silver's (1988a, 1990, 1991a, 1991b) definitions of "restrictive" and "guided" out of context. By doing this they have taken a two-dimensional classification and incorrectly oversimplified it by reducing it to one dimension. Second, there is virtually no difference between methodology checks made at the request of the analyst after exiting a technique (such as data flow diagramming) and a phase (such as analysis). Third, their distinction between "restrictive" and "guided" relies entirely upon the semantical difference between the words "error" and "warning" (when looking at how a violation is displayed to the analyst), i.e., Tool A may list a methodology violation as an "error" while Tool B may list the same violation as a "warning". However, the end result is the same -- the CASE tool leaves it to the analyst to decide whether or not to fix the violation. Fourth, the classification scheme is based entirely upon counting data, i.e., Tool A is considered to be "more restrictive" than Tool B simply because it has more methodology rules in its rule base or has more rules that are implemented in a particular fashion. This assumption fails to take into account the fact that there may be a particular rule, or subset of rules, that contribute more to the consistency of the specifications than any other rule(s). While Tool B may have less rules in its rule base than Tool A, Tool B may have rules in its rule base that are not found in the rule base of tool A. Finally, none of the CASE tools implement every rule in their rule base in the same fashion. For a particular CASE tool, some of the rules may be enforced in a "restrictive" manner, some may be enforced in a "guided" manner, and some may be enforced in a "flexible" manner. As will be discussed in Section 6.5, it is not feasible to implement every methodology rule in a restrictive manner.

2.6 Summary

This chapter has reviewed the relevant research on structured analysis and CASE support for structured analysis. While empirical research concerning the systems development life cycle and the structured techniques is limited, the life cycle and the structured techniques are still preferred for systems development. However, many organizations and analysts have criticized the structured techniques due to their labor-intensive characteristics. CASE tools were introduced in the early 1980's to answer these criticisms and make the structured techniques more feasible. As with the structured techniques, empirical research concerning CASE's ability to affect system quality and analyst productivity is limited. The literature makes clear, however, that CASE tools need to provide more support for their underlying methodologies. The classification schemes proposed for categorizing CASE tools based upon their level of methodology support have been reviewed. Chapter 3 introduces a new framework for classifying CASE methodology support. Rather than trying to describe an entire tool, the unit of study will be the individual rules being supported by the tool. A general model for systems development research will also be presented along with the research concepts, hypothesis, and variables.

CHAPTER 3

CONCEPTS, VARIABLES, AND HYPOTHESES

3.1 Introduction

This chapter presents the general guiding framework for this research. A framework for examining CASE methodology support will be proposed. This framework will be coupled with a model for systems development, which will serve as the basis for the experimental study. The systems development model describes how independent variables may influence the outcome of systems development activities. Next, the dependent measure, and how it will be operationalized in this study, will be described. Finally, the research hypotheses will be presented.

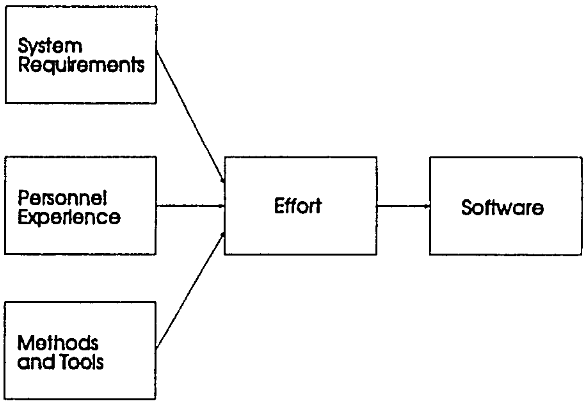


Figure 6 Systems Development Effort Model (Wrigley & Dexter, 1988)

3.2 A Research Model for Systems Development

A research model for studying the systems development process is given by Wrigley and Dexter (1987, 1988) and is presented in Figure 6. The model is derived from a collection of 74 factors, taken from five existing models, that have been found to affect the systems development process.[11] The model presents the important classes of variables and their relationship to the effort required to develop a computer-based system. The model treats the effort required to develop a system as the consequence of the properties of the system being developed, the properties of the personnel comprising the system development team, and the methods and tools being used to develop the system. The model is designed to be parsimonious and offer estimations of effort at an early point in the systems development life cycle. The Wrigley and Dexter model postulates that:

1. As system requirements increase required effort will also increase.
2. For a given set of requirements, as personnel experience increases required effort decreases.
3. For a given set of requirements, as development technology advances required effort decreases (Wrigley & Dexter, 1987, p. 12).

The model described by Wrigley and Dexter was developed for the purpose of determining the effort required to build an information system. They acknowledge, however, that design quality may be a missing factor or variable. A modified version

[11] The five models chosen are Walston and Felix's (1977) IBM FSD, Putnam's (1978) SLIM, Boehm's (1981) COCOMO, Rubin's (1983) ESTIMACS, and Jones's (1986) SPQR/20.

of Wrigley and Dexter's model is displayed in Figure 7.[12] In this model, design quality is added and shown to be dependent upon the factors previously identified: requirements, personnel, and methods and tools. Wrigley and Dexter (1987) state, "Design quality has been largely ignored in all systems development models" (p. 9). They attribute this to the fact that it is assumed that the use of the "structured techniques and modern programming practices" will naturally lead to a quality design. Further justification for adding design quality to the model is provided by Wrigley and Dexter, who indicate that "good design quality will lead to systems which require less effort to implement, test, and maintain" (p. 9). The remainder of this section will describe the model components as they pertain to this study.[13]

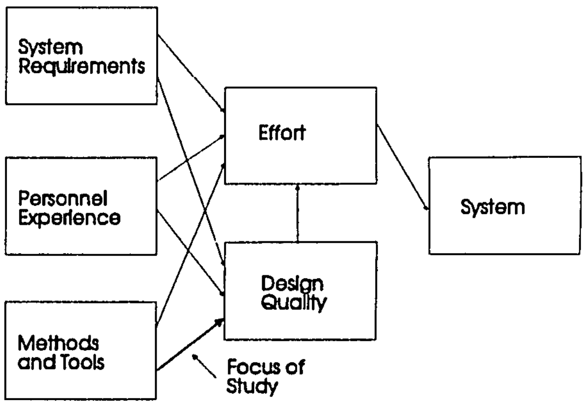


Figure 7 Extended Systems Development Effort Model

[12] A less detailed model, similar to the one displayed in Figure 7, is used by Purvis and Sambamurthy (1992) for the purpose of comparing systems development methodologies. The model presented in Figure 7 is also similar to that presented by Batra, Hoffer, & Bostrom (1990).

[13] This study will only investigate the portion of the model concerned with determining the design quality of the system. Sections 6.3 and 7.4.3 discuss extensions of this research to the effort component of the model.

3.2.1 Properties of System Requirements

Among the properties of system requirements that affect design quality are the application domain, the number of files comprising the database, the number of records in the database, the number of data elements contained in the individual records, the number of system inputs, the number of system outputs, and the number of business functions to be implemented (Wrigley and Dexter, 1987). System requirements were controlled in this study by assigning an identical set of requirements for a proposed hotel information system to each project team (discussed in Section 4.5).

3.2.2 Properties of the Development Personnel

Among the properties of the development personnel that affect design quality are the amount of analysis experience for the personnel, the amount of programming experience, and the degree of familiarity with the application domain (Wrigley and Dexter, 1987). Variations in the backgrounds of the development personnel were controlled in this study by randomly assigning individuals to development teams (discussed in Section 4.4).

3.2.3 Properties of the Tools and Methods

Among the properties of the development tools and methods that affect design quality are the use of structured methodologies such as structured analysis, the use of software tools such as CASE, process constraints placed upon the design of the system, and computer access. In this study, all development teams were trained in the same development methodology and used a CASE tool to assist them in developing a system specification. Additionally, the same hardware platform was utilized by all teams. However, the CASE tools differed in the way they constrained the development teams to follow the prescribed methodology. The level of methodology support provided by the tools is the independent variable in this study and will be discussed in detail in the following section.

3.3 A Framework for Classifying CASE Methodology Support

The Vessey et al. (1992) scheme for classifying CASE methodology support, described in the previous chapter, is based upon work done in the area of decision support systems (DSS). Specifically, two attributes of a DSS, system restrictiveness and decisional guidance (as defined in Silver, 1988a, 1990, 1991a, 1991b), have been redefined and applied to CASE tools. The concepts of restricting and guiding users of software applications has been applied to a diverse area of applications, including decision support systems (Silver, 1988b), group decision support systems (DeSanctis, D'Onofrio, Sambamurthy, & Poole, 1989), word processing (Carroll & Carrithers, 1984), expert systems (Carroll & McKendree, 1987), and computer-aided instruction (Hannafin, 1984).

The remainder of this section examines the concepts of restrictiveness and guidance as they apply to CASE tools. By examining both restrictiveness and guidance, it will be possible to not only answer the question, "Does CASE tool X support methodology Z?", but also, "*To what extent* does CASE tool X support methodology Z?".

3.3.1 CASE Tool Restrictiveness

CASE tool restrictiveness may be formally defined as follows:

The degree to which and the manner in which a CASE tool limits its users' systems development processes to a subset of all possible processes (adapted from Silver, 1990).

A CASE tool may restrict both the structure of the systems development process as well as the execution of the systems development process (see Figure 8). The structure of the systems development process may be restricted by a CASE tool by limiting the analyst to a subset of all possible development activities. A systems development activities hierarchy can be constructed and applied to CASE tools in order to determine the relative level of restrictiveness *with respect to systems*

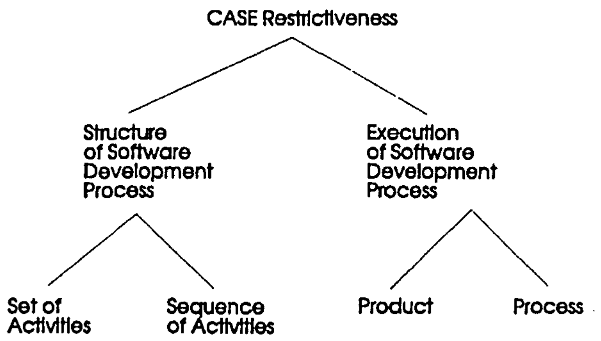

CASE Restrictiveness
Structure of Software Development Process
Execution of Software Development Process
Set of Activities
Sequence of Activities
Product
Process

Figure 8 CASE Tool Restrictiveness (adapted from Silver, 1990)

development activities between different CASE tools (see Figure 9). Each inner layer in the hierarchy is a subset of the immediate outer layer. For example, a particular CASE tool may support all activities associated with analysis, such as data flow diagramming, a data dictionary, and primitive process specifications, while a second CASE tool might only support data flow diagramming. By allowing an analyst to use the tool to only construct data flow diagrams, the later CASE tool is restricting the analyst to a subset of the development activities supported by the former CASE tool.[14] A CASE tool can also restrict the structure of the systems development process by imposing an order upon the activities supported by the tool. For example, a CASE tool may require an analyst to complete a set of data flow diagrams before beginning to construct a set of structure charts.

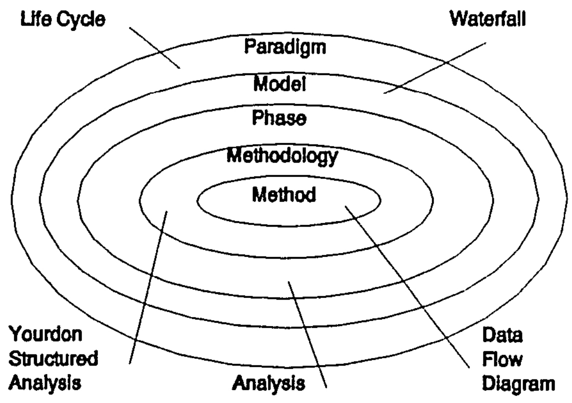


Figure 9 Restricting the Set of Systems Development Activities

[14] It should be noted that a comparison of the restrictiveness (with respect to the set of systems development activities supported) of two CASE tools is possible only when one tool's set of supported activities is a subset of the supported activities of the second tool. See (Silver, 1991b, pp. 115-116) for a more thorough treatment of this topic.

Along with the structure of the systems development process a CASE tool may also restrict the execution of the systems development process. This may be accomplished by placing restrictions upon the procedures used to create the product of a particular systems development activity as well as restrictions upon the products themselves. To illustrate this consider Table 2, which contains a list of methodology rules for structured analysis. The first rule listed in Table 2 applies to the *process* of structured analysis, i.e., ensuring that the specification is constructed in a top-down manner. The remaining rules apply to the *products* of structured analysis, i.e., syntax rules for data flow diagrams and minispecs. The product rules can be further categorized as those rules that ensure the internal consistency of a product (e.g., ensuring that data stores are connected to processes) and those rules that ensure the hierarchical consistency between products (e.g., ensuring that data flows are balanced between levels) (Vessey et al., 1992). This study examines the effects of restricting the execution of the systems development process.

Table 2 Structured Analysis Methodology Rules

Process Rule

1. A parent process must be specified before a child process.

Product Rules

Internal Consistency

Data Flow Diagram

2. At least one process.
3. No more than seven processes.

Context Diagram

4. Must exist.
5. Must contain only one process.
6. At least one input from an external entity and one output to an external entity.
7. The process must be numbered 0.

Process

8. At least one input data flow and one output data flow.
9. Must be connected to a data store, process, or external entity.
10. Must be labeled.

External Entity

11. Must only appear on the context diagram.
12. Must be connected to a process.
13. Must be labeled.

Data Flow

14. Must be an interface between a process and either a second process, a data store, or an external entity.
15. A data flow into (from) a data store must have a composition that is a subset of the data store's composition.
16. Must be labeled.

Table 2 (*Continued*)

Data Store

17. Must be an interface between two processes.

18. Must be labeled.

Hierarchical Consistency

Data Flow Diagram

19. A parent data flow diagram must exist unless it is a context diagram.

Process

20. Must decompose to either another data flow diagram or a primitive process specification.

21. Must be numbered with respect to its parent.

Data Flow

22. An input (output) data flow on a parent data flow diagram must appear on a child data flow diagram as input (output).

23. An input (output) data flow on a child data flow diagram must appear on a parent data flow diagram as input (output).

24. A set of data flows on a child data flow diagram that were split from a data flow on a parent data flow diagram must match the parent data flow's composition.

25. A data flow must decompose to either a record definition or an element definition.

Data Store

26. A data store must decompose to either a file definition or a record definition.

Primitive Process Specification

27. All inputs and outputs must match those of the corresponding primitive process on the data flow diagram.

28. Must be labeled with the same identifier as the corresponding primitive process on the data flow diagram.

In order to restrict the execution of systems development, rules must be embedded within a CASE tool that serve to force the analyst to rigorously follow the rules of a chosen operator (e.g., data flow diagramming). In order to determine if a rule has been implemented in a restrictive fashion, three properties of the rule must be examined: the timing of the rule, the invocation of the rule, and the enforcement of the rule (see Figure 10). The next paragraph discusses these properties within the context of a data flow diagramming tool.

The *timing* of a rule refers to when a rule violation is presented to the analyst. A rule that is implemented in a restrictive fashion by a CASE tool will allow violations to be checked as soon as is feasible to do so. Restriction implies that an analyst is being forced to conform to the rules of the chosen operator. Therefore, it is imperative that the analyst be given the opportunity to fix a violation as soon as the violation is detected by the CASE tool in order to keep the violation from propagating through the system specification. This implies that rule violations must be detectable while in the process of using the operator (Level 1 Restriction). Other rule violations may not be detectable until the analyst is finished

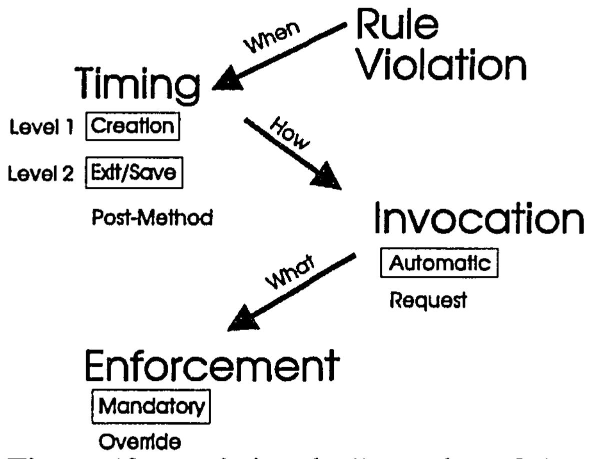


Figure 10 Restricting the Execution of the Systems Development Process

using an operator (e.g., while saving and/or exiting) (Level 2 Restriction). The *invocation* of a rule refers to the mechanism by which the rule violation is presented to the analyst. A rule that is implemented in a restrictive fashion (Level 1 or Level 2) by a CASE tool will be automatically presented to the analyst as soon as it is detected by the CASE tool. However, it is very important that a rule violation be correctly distinguished from unfinished work by the analyst. For example, if an analyst tries to connect two data stores with a data flow then this violation may be immediately presented to the analyst by the CASE tool. However, if an analyst has just entered a new process on a diagram it would be counter-productive for the CASE tool to immediately stop the analyst to indicate that the process has no inflows or outflows. The second example is a suspected rule violation and may be flagged automatically but it would be more appropriate to do so at the request of the analyst. However, the suspected violation could be automatically presented when the analyst attempts to save the diagram and/or exit the diagramming tool. The *enforcement* of the rule refers to the set of options available to the analyst once a rule violation has been detected by the CASE tool. The analyst may be forced to correct the violation before proceeding further (mandatory) or the analyst may be allowed to continue working and ignore the error (override). In summary, a rule will be considered to be implemented within the CASE tool in a restrictive fashion if the analyst is automatically presented with the rule violation while using an operator, or while terminating use of the operator, and is forced to correct the violation before proceeding.

3.3.2 CASE Tool Development Guidance

An alternative to restricting the structure and execution of the systems development process is to provide suggestions and information that serve to guide the analyst through the development process. CASE tool development guidance may be formally defined as follows:

The degree to which and the manner in which a CASE tool guides its users in structuring and executing the systems development process and constructing its resultant products, by assisting them in choosing and using its methods (adapted from Silver, 1990).[15]

A CASE tool may guide both the structure of the systems development process as well as the execution of the systems development process (see Figure 11). The structure of the systems development process may be guided by the CASE tool through the use of suggestions and information that allow the analyst to choose and order the development techniques. For example, after completing a data flow diagram the

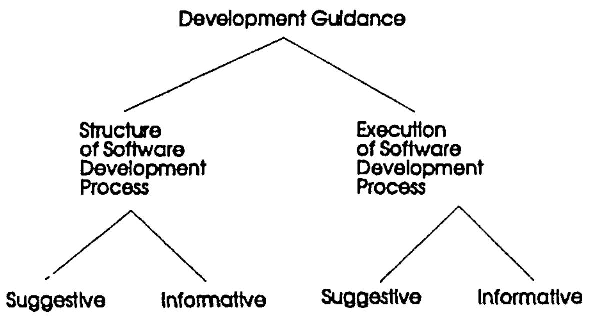

Development Guidance
Structure of Software Development Process
Execution of Software Development Process
Suggestive
Informative
Suggestive
Informative

Figure 11 CASE Tool Development Guidance (adapted from Silver, 1990)

[15] It should be noted that the type of guidance referred to in the definition is that which applies directly to the systems development process. A help screen that simply lists what happens when a particular key is pressed does not constitute development guidance.

CASE tool may suggest that the analyst begin constructing the primitive process specifications but offer the option of skipping directly to structure charts. In this instance, the analyst is not being restricted to follow a predetermined sequence of operations. Rather, the analyst is being presented with a logical choice of operators based upon the work completed to that point. An examination of the effects of guiding the structure of the systems development process is beyond the scope of this study.

Along with the structure of the systems development process a CASE tool may also guide the execution of the systems development process. This may be accomplished by providing the analyst with suggestions and information regarding the procedures of a particular systems development activity as well as the resultant product of that activity. Section 3.3.1 describes three properties of methodology rules embedded in a CASE tool: timing, invocation, and enforcement. These three properties will now be discussed as they pertain to providing guidance within the context of a data flow diagramming tool.

By definition, the word "guidance" implies the existence of an underlying choice, i.e., a decision must be made as to whether or not to accept the guidance. The guidance provided by the CASE tool to the analyst must be rigorous enough to highlight methodology violations when they occur yet be flexible enough so that the analyst can ignore the embedded advice. Two types of guidance can be made available to the analyst by the CASE tool: active guidance and passive guidance. Active guidance is informative and suggestive advice that is unsolicited, i.e., the CASE tool delivers the guidance to the analyst when the CASE tool detects a need for guidance (see Figure 12). Active guidance can be provided by the CASE tool while the analyst is using an operator (Level 1 Active Guidance) or it may be provided by the CASE tool when the analyst is finished using an operator (e.g., while saving and/or exiting) (Level 2 Active Guidance). The violation must be presented to the analyst in the form of an informative message or suggestion for correcting the violation. It is then left to the discretion of the analyst to determine whether or not to correct the violation. For example, if the analyst tries to delete an input data flow from a child diagram the CASE tool could immediately warn the analyst that the deletion may result in a methodological inconsistency. The analyst should then have the option of continuing or aborting the operation.

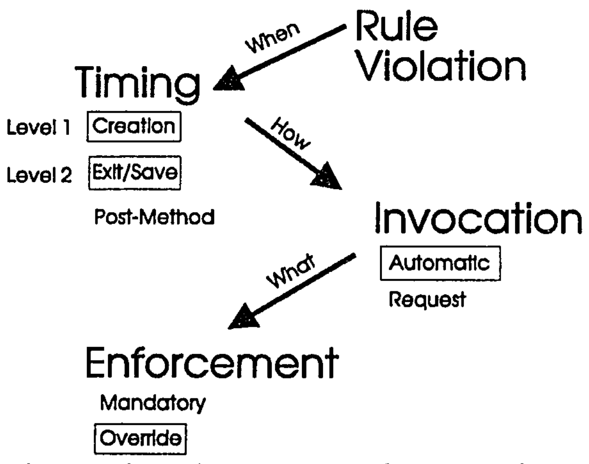


Figure 12 Active Guidance for Executing the Systems Development Process

A second type of guidance available from a CASE tool is passive guidance. Passive guidance is informative and suggestive advice that is solicited by the analyst from the

CASE tool (see Figure 13). For example, if the analyst is unsure if a parent and child data flow diagram are balanced he can request that the CASE tool check the balancing and report any inconsistencies. Passive guidance may be requested by the analyst from within the data flow diagramming tool (Level 1 Passive Guidance) or it may be implemented as a separate function outside of the diagramming tool (Level 2 Passive Guidance). For example, Visible Analyst Workbench allows the analyst to validate work in progress by requesting a consistency check from within the data flow diagramming tool. Excelerator, on the other hand, requires the analyst to save the diagram, exit the diagramming tool, and then execute a separate analysis function. As with active guidance, any violations must be presented to the analyst in the form of informative messages or suggestions for correcting the violations.

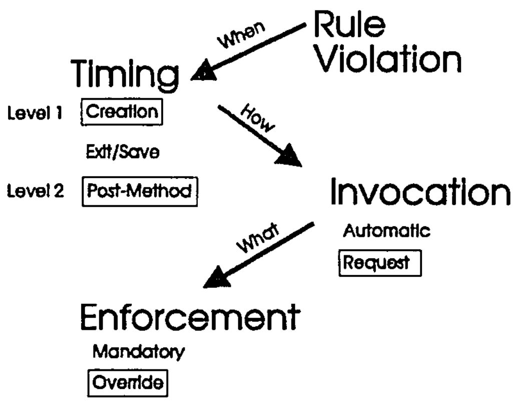


Figure 13 Passive Guidance for Executing the Systems Development Process

The amount of guidance a CASE tool provides the analyst is a function of the restrictiveness of the CASE tool, i.e., a CASE tool that is minimally restrictive has many opportunities to provide informative and suggestive guidance to the analyst. If a CASE tool handles all methodology rule violations in the restrictive manner described in Section 3.3.1 then there is no opportunity to suggest alternatives to working around the rule violation -- the analyst must fix the violation before proceeding. For example, if a context diagram is required to have only one process a CASE tool that enforces this rule in a restrictive manner would not allow more than one process to be placed on the diagram. On the other hand, if a CASE tool does not handle rule violations in a restrictive fashion, the opportunity presents itself for the CASE tool to offer suggestions about the nature of the violation. Referring back to the context diagram example, if the CASE tool does not enforce this particular rule in a restrictive fashion it may simply notify the user of the existence of the violation and offer the analyst advice as to the consequences of placing more than one process on a context diagram. The analyst is then free to consider the advice as he sees fit.

An alternative to embedding restrictiveness and guidance within a CASE tool is the complete lack of support for a methodology or a particular methodology rule. For example, a CASE tool may not contain any embedded checks of the internal consistency of a diagram. Rather, the analyst is left with the responsibility of making certain the diagram adheres to the rules of the chosen methodology. In the next section objectives favoring greater and lesser degrees of restrictiveness of the systems development process will be discussed.

3.3.3 Objectives of Restrictiveness

Too much or too little restrictiveness may prevent an analyst from using the CASE tool. A CASE tool that is too restrictive may exclude an experienced analyst from using a preferred method of systems development (Vessey et al., 1992). The end result may be that the analyst will decide not to use the CASE tool. On the other hand, too little restriction may overwhelm an analyst by offering more options than the analyst may be able to handle. The end result may be that the CASE tool is not properly utilized.

There are several objectives that favor a high degree of CASE tool restrictiveness. A restrictive CASE tool can be used by an organization to prescribe a particular development methodology for their analysts. Rowe (1993) reports that, among systems analysts, CASE users are required to conform to a specific methodology more often than non-CASE users. Rowe also reports that CASE users have their development standards more rigidly enforced than non-CASE users. Alternatively, a restrictive CASE tool may be used to proscribe systems development techniques that are counter to those mandated by the organization. A restrictive CASE tool may also be used to provide structure to the extremely complex task of systems development and promote structured learning of a development methodology (adapted from Silver, 1990).

There are also objectives favoring lesser degrees of restrictiveness which must be considered. By offering a less restrictive CASE tool, more options are available to support the development of different types of information systems for use in different functional areas of business. Palvia and Nosek (1990) indicate that the life cycle paradigm is preferred for transaction processing systems and management information systems, while the prototyping paradigm is preferred for decision support systems and executive information systems. A less restrictive CASE tool may also promote analyst creativity and foster exploratory learning during the systems development process (adapted from Silver, 1990). The following section examines the two CASE tools used in this study, Visible Analyst Workbench 3.1 and Excelerator 1.9, and compares how they enforce the structured analysis development methodology.

3.3.3.1 Comparison of Two CASE Tools

Table 3 contains the results of Vessey et al.'s (1992) evaluation of the data flow diagramming tool for two popular CASE tools, Intersolv's Excelerator Version 1.8 and Visible System's Visible Analyst Workbench (VAW) Version 3.0. The authors categorized Excelerator as one of the two most flexible CASE tools in their study and VAW as one of the two most restrictive CASE tools in their study. For the current study, more recent versions of the CASE tools were utilized: Excelerator 1.9 and Visible Analyst Workbench 3.1. Vessey et al. reported their findings based on 17 methodology rules for data flow diagrams. The current study examines 28 rules which encompass data flow diagrams, the data dictionary, and primitive process specifications. The remainder of this section describes how the 28 structured analysis methodology rules are implemented by each CASE tool. For each CASE tool, the 28 rules will be mapped to the methodology support spectrum introduced earlier in this section.

Table 3 Results of CASE Tool Evaluation (Vessey, Jarvenpaa, & Tractinsky, 1992)

	Total	When			How		What	
Tool	Checks	creation	technique	phase	auto	request	error	warning
VAW 3.0								
Process (1)[16]								
Hier. Con. (5)	3	3			2	1	3	
Int. Con. (11)	8	8			1	7	8	
Excelerator 1.8								
Process (1)	1	1			1			
Hier. Con. (5)	1			1		1	1	
Int. Con. (11)	5			5		5	5	

Adapted from Vessey et al. (1992).

Before beginning a new systems development project with Excelerator 1.9 the analyst has the option of specifying either the Yourdon or Gane & Sarson data flow diagram symbol set. The symbol set may be changed at any time during the project. Visible

[16] The number in parenthesis indicates the number of possible checks, i.e., there was one check for methodology process, there were five checks for hierarchical consistency between diagrams, and there were eleven checks for internal consistency within diagrams.

Analyst Workbench 3.1 offers many options to the analyst before beginning a systems development project. First, the analyst must indicate the level of analysis support provided by the CASE tool. Specifically, the analyst may indicate whether or not the methodology enforcement described above will be utilized for the project. The analyst may disable all methodology rules or choose a particular implementation of structured analysis, either Yourdon or Gane & Sarson. The only difference between the implementations, as far as VAW is concerned, is the symbol set. Regardless of the implementation chosen, the analyst may use diagramming symbols from either Yourdon or Gane & Sarson. However, if rules are "enabled" they will only apply to the symbols from the chosen set. Similarly, only symbols from the enabled set may be described in the data dictionary. For example, if the analyst chooses to enable Yourdon rules he may also use Gane & Sarson symbols in the data flow diagrams. The methodology rules and the data dictionary, however, will only apply to those diagram symbols that are from the Yourdon set. The analyst may also specify whether or not the CASE tool automatically prompts the analyst to label both data flows and symbols. The analyst also has the option of enabling the data dictionary for the project. The data dictionary may only be utilized if methodology rules are being used. These parameters may not be changed once a project has begun.

Process Rule

1. A parent process must be specified before a child process.

Visible Analyst Workbench 3.1 -- In VAW, a process does not need a parent process to exist. Data flow diagram hierarchy is represented to the analyst via a project tree. Nodes may be inserted anywhere within the tree and then linked to a child diagram if desired. The analysis report contains a listing of all diagrams (and, therefore, all processes) that do not have a parent. This report may be generated at the request of the analyst from within the data flow diagramming tool. However, it should be noted that the absence of a parent process does not necessarily imply that the parent did not exist at one time and that it was not created before its child.

Excelerator 1.9 -- In Excelerator, a process does not need a parent process to exist. Data flow diagrams may be created in any order desired and then linked together after they have been created. The analyst has no way of determining which processes do not have a parent as all reports indicate downward decomposition only.

Product Rules (Internal Consistency)

2. A data flow diagram must have at least one process.

Visible Analyst Workbench 3.1 -- With the exception of the context diagram, this rule is not implemented by VAW.

Excelerator 1.9 -- Violations of this rule are indicated to the analyst in the "Data Flow Diagram Verification Report." This report may be generated at the request of the analyst from outside the data flow diagramming tool.

3. A data flow diagram must have no more than seven processes.

Visible Analyst Workbench 3.1 -- This rule is not implemented by VAW.

Excelerator 1.9 -- This rule is not implemented by Excelerator.

4. A context diagram must exist.

Visible Analyst Workbench 3.1 -- When using the Yourdon methodology rules, VAW asks the analyst at the beginning of the project if a context diagram will be used. The analyst has the option of accepting or rejecting a context diagram.

Excelerator 1.9 -- While the Excelerator documentation does indicate that a context diagram is the top-level of a data flow diagram set, Excelerator makes no provision for assigning one of the diagrams to be a context diagram.

5. The context diagram must contain only one process.

Visible Analyst Workbench 3.1 -- Once a process has been placed on the context diagram, VAW disables the use of the process symbol for that diagram. VAW enforces the requirement of having a process on the context diagram by not allowing the diagram to be saved until a process is placed on the diagram.

Excelerator 1.9 -- Excelerator does not recognize the concept of a context diagram.

6. The context diagram must contain at least one input from an external entity and one output to an external entity.

Visible Analyst Workbench 3.1 -- This rule is not implemented by VAW.

Excelerator 1.9 -- Excelerator does not recognize the concept of a context diagram.

7. The context diagram process must be numbered zero (0).

Visible Analyst Workbench 3.1 -- The process is automatically numbered zero by VAW and may not be changed.

Excelerator 1.9 -- Excelerator does not recognize the concept of a context diagram.

8. A process must have at least one input data flow and one output data flow.

Visible Analyst Workbench 3.1 -- The analysis report contains a listing of processes that are either "input only" or "output only." This report may be generated at the request of the analyst from within the data flow diagramming tool.

Excelerator 1.9 -- Violations of this rule are indicated to the analyst in the "Data Flow Diagram Verification Report." This report may be generated at the request of the analyst from outside the data flow diagramming tool.

9. A process must be connected to at least one of the following: data store, process, external entity.

Visible Analyst Workbench 3.1 -- The analysis report contains a listing of processes that are freestanding. This report may be generated at the request of the analyst from within the data flow diagramming tool.

Excelerator 1.9 -- Freestanding or self-connected processes are indicated to the analyst in the "Data Flow Diagram Verification Report." This report may be generated at the request of the analyst from outside the data flow diagramming tool.

10. A process must be labeled.

Visible Analyst Workbench 3.1 -- When a process is created the user is automatically prompted to enter a label. However, the user may override this prompt if desired. The analysis report contains a listing of processes that are unlabeled. This report may be generated at the request of the analyst from within the data flow diagramming tool.

Excelerator 1.9 -- Labels may be assigned to processes from within the data flow diagramming tool or the data dictionary. The data dictionary is accessible from within the data flow diagramming tool. Once a label has been entered the analyst has the option of displaying or not displaying the label on the diagram. Violations of this rule are indicated to the analyst in the "Undescribed Graph Entities Report." This report may be generated at the request of the analyst from outside the data flow diagramming tool.

11. An external entity must appear for the first time on the context diagram.

Visible Analyst Workbench 3.1 -- An external entity may be drawn on any diagram and the data flow it receives or produces is considered a "net input (output)" data flow. If the entity is appearing for the first time in the diagram set (but not on the context diagram) the data flow will be indicated in the analysis report as being inconsistent with the data flows on the context diagram. This report may be generated at the request of the analyst from within the data flow diagramming tool.

Excelerator 1.9 -- Excelerator does not recognize the concept of a context diagram. An external entity may appear on any diagram. Data flows into or out of the external entity (net input or net output) are treated the same as any other data flow.

12. An external entity must be connected to a process.

Visible Analyst Workbench 3.1 -- The analysis report contains a listing of external entities that are freestanding or are not connected to a process. This report may be generated at the request of the analyst from within the data flow diagramming tool.

Excelerator 1.9 -- Freestanding or self-connected external entities are indicated to the analyst in the "Data Flow Diagram Verification Report." This report may be generated at the request of the analyst from outside the data flow diagramming tool. There is nothing implemented by Excelerator that would stop an analyst from connecting an external entity to a data store.

13. An external entity must be labeled.

Visible Analyst Workbench 3.1 -- When an external entity is created the user is automatically prompted to enter a label. However, the user may override this prompt if desired. The analysis report contains a listing of external entities that are unlabeled. This report may be generated at the request of the analyst from within the data flow diagramming tool.

Excelerator 1.9 -- Labels may be assigned to external entities from within the data flow diagramming tool or the data dictionary. The data dictionary is accessible from within the data flow diagramming tool. External entities also have an identification tag, which is used by the data dictionary. The analyst is prompted to enter an identifier before describing the external entity for the first time in the data dictionary. If a label already exists, the analyst has the option of using the label as the identifier. The analyst has the option of displaying or not displaying both the label and the identifier on the diagram. Violations of this rule are indicated to the analyst in the "Undescribed Graph Entities Report." This report may be generated at the request of the analyst from within the data flow diagramming tool.

14. A data flow must be an interface between a process and either a second process, a data store, or an external entity.

Visible Analyst Workbench 3.1 -- Data flows that are not connected at one end to a process are indicated in the analysis report. Data flows that are connected at only one end point are assumed to be input (output) from the parent diagram and, if they do not match with the parent diagram, will be indicated in the analysis report as inconsistent data flows. This report may be generated at the request of the analyst from within the data flow diagramming tool.

Excelerator 1.9 -- Data flows are defined by the analyst by selecting two objects to connect. Therefore, there cannot be any freestanding data flows. Data flowing between two data stores or two external entities is reported to the analyst in the "Data Flow Diagram Verification Report". This report may be generated at the request of the analyst from outside the data flow diagramming tool.

15. A data flow into (from) a data store must have a composition that is a subset of the data store's composition.

Visible Analyst Workbench 3.1 -- While the VAW dictionary allows the analyst to specify the composition of the both data flows and data stores, there is no check performed on the data flows and data stores to ensure a subset relationship exists.

Excelerator 1.9 -- Violations of this rule are indicated to the analyst in the "Data Flow Diagram Verification Report". This report may be generated at the request of the analyst from outside the data flow diagramming tool.

16. A data flow must be labeled.

Visible Analyst Workbench 3.1 -- When a data flow is created the user is automatically prompted to enter a label. However, the user may override this prompt if desired. The analysis report contains a listing of data flows that are unlabeled. This report may be generated at the request of the analyst from within the data flow diagramming tool.

Excelerator 1.9 -- Labels may be assigned to data flows from within the data flow diagramming tool or the data dictionary. The data dictionary is accessible from within the data flow diagramming tool. Data flows also have an identification tag, which is used by the data dictionary. The analyst is prompted to enter an identifier before describing the data flow for the first time in the data dictionary. If a label already exists, the analyst has the option of using the label as the identifier. The analyst has the option of displaying or not displaying both the label and the identifier on the diagram. Violations of this rule are indicated to the analyst in the "Undescribed Graph Entities Report." This report may be generated at the request of the analyst from outside the data flow diagramming tool.

17. A data store can only exist as an interface between two processes.

Visible Analyst Workbench 3.1 -- If a data store exists with only input (output) the data flow is presumed to be a net input (output) from the parent diagram. If the flows do not match with the parent diagram they will be indicated in the analysis report as inconsistent data flows. The analysis report also indicates freestanding data stores. This report may be generated at the request of the analyst from within the data flow diagramming tool.

Excelerator 1.9 -- In Excelerator a data store must be connected to at least one process. Violations of this rule are indicated to the analyst in the "Data Flow Diagram Verification" report. A freestanding data store will also be indicated in this report. This report may be generated at the request of the analyst from outside the data flow diagramming tool. There is nothing implementea by Excelerator that would indicate a data store that provides solely input or output and should, therefore, be represented as an external entity.

18. A data store must be labeled.

Visible Analyst Workbench 3.1 -- When a data store is created the user is automatically prompted to enter a label. However, the user may override this prompt if desired. The analysis report contains a listing of data stores that are unlabeled.

This report may be generated at the request of the analyst from within the data flow diagramming tool.

Excelerator 1.9 -- Labels may be assigned to data stores from within the data flow diagramming tool or the data dictionary. The data dictionary is accessible from within the data flow diagramming tool. Data stores also have an identification tag, which is used by the data dictionary. The analyst is prompted to enter an identifier before describing the data store for the first time in the data dictionary. If a label already exists, the analyst has the option of using the label as the identifier. The analyst has the option of displaying or not displaying both the label and the identifier on the diagram. Violations of this rule are indicated to the analyst in the "Undescribed Graph Entities Report." This report may be generated at the request of the analyst from outside the data flow diagramming tool.

Product Rules (Hierarchical Consistency)

19. A parent data flow diagram must exist unless it is a context diagram.

Visible Analyst Workbench 3.1 -- In VAW, a diagram may be created as a node anywhere within the data flow diagram hierarchy. However, a parent-child relationship does not have to exist between diagrams. The analysis report contains a listing of diagrams that do not have a parent. This report may be generated at the request of the analyst from within the data flow diagramming tool. Additionally, no diagram may be placed above a context diagram.

Excelerator 1.9 -- This rule is not enforced by the CASE tool. Data flow diagrams may be created and linked in any order the analyst desires. Alternatively, the analyst is not required by Excelerator to link the diagrams.

20. A process must decompose to either another data flow diagram or a primitive process specification.

Visible Analyst Workbench 3.1 -- This rule is not implemented by VAW. In order to determine if processes have been fully decomposed the analyst must browse the data dictionary and examine each process for its level of decomposition.

Excelerator 1.9 -- A "Graph Explosion Report", which lists the decomposition of objects on a specified list of data flow diagrams, may be generated at the request of the analyst from outside the data flow diagramming tool. All undefined process decompositions are indicated in this report.

21. A process must be numbered with respect to its parent.

Visible Analyst Workbench 3.1 -- A process is automatically numbered with respect to its parent when it is created. However, the analyst may later change the assigned numbering by using a text editing function within the data flow diagramming tool. All processes must have a number but they do not have to be numbered with respect to their parents.

Excelerator 1.9 -- When the analyst describes a process in the data dictionary for the first time, Excelerator requires the analyst to input a process identifier. Excelerator provides a default identifier which is numbered with respect to the parent process.

The analyst may use the default value or provide a new identifier. While the prompting for the identifier is automatic and may be overridden, the prompt only occurs as a result of attempting to put a process in the dictionary. If the process is never placed in the dictionary it will never be numbered. Violations of this rule are indicated to the analyst in the "Undescribed Graph Entities Report." This report may be generated at the request of the analyst from outside the data flow diagramming tool.

22. An input (output) data flow on a parent data flow diagram must appear on a child data flow diagram as input (output).

Visible Analyst Workbench 3.1 -- Violations of this rule are indicated to the analyst in the analysis report. This report may be generated at the request of the analyst from within the data flow diagramming tool. When a process is decomposed, all input and output data flows are carried down to the new diagram. There is nothing to stop the analyst from deleting these data flows.

Excelerator 1.9 -- Violations of this rule are indicated to the analyst in the "Level Balancing Report". This report may be generated at the request of the analyst from outside the data flow diagramming tool. When a process is decomposed, all input and output data flows are carried down to the new diagram. There is nothing to stop the analyst from deleting these data flows.

23. An input (output) data flow on a child data flow diagram must appear on a parent data flow diagram as input (output).

Visible Analyst Workbench 3.1 -- Violations of this rule are indicated to the analyst in the analysis report. This report may be generated at the request of the analyst from within the data flow diagramming tool. When a process is decomposed, all input and output data flows are carried down to the new diagram. There is nothing to stop the analyst from adding more input (output) data flows.

Excelerator 1.9 -- Violations of this rule are indicated to the analyst in the "Level Balancing". This report may be generated at the request of the analyst from outside the data flow diagramming tool. When a process is decomposed, all input and output data flows are carried down to the new diagram. There is nothing to stop the analyst from adding more input (output) data flows.

24. A set of input data flows on a child data flow diagram that were split from a data flow on a parent data flow diagram must match the parent data flow's composition.

Visible Analyst Workbench 3.1 -- Split data flows are supported by VAW. If any inputs that were split from a data flow on a parent diagram are not found this will be indicated in the analysis report. This report may be generated at the request of the analyst from outside the data flow diagramming tool.

Excelerator 1.9 -- Excelerator does not support split data flows.

25. A data flow must decompose to either a record definition or an element definition.

Visible Analyst Workbench 3.1 -- This rule is not implemented by VAW. The composition of a data flow can be indicated from within the data dictionary. However, there is no report available that indicates which flows have yet to be decomposed. In order to determine if all data flows have been decomposed the analyst must browse the data dictionary and examine each data flow's composition field.

Excelerator 1.9 -- A "Graph Explosion Report", which lists the decomposition of objects on a specified list of data flow diagrams, may be generated at the request of the analyst from outside the data flow diagramming tool. All undefined data flow decompositions are indicated in this report.

26. A data store must decompose to either a file definition or a record definition.

Visible Analyst Workbench 3.1 -- This rule is not implemented by VAW. The composition of a data store can be indicated from within the data dictionary. However, there is no report available that indicates which stores have yet to be decomposed. In order to determine if all data stores have been decomposed the analyst must browse the data dictionary and examine each data store's composition field.

Excelerator 1.9 -- A "Graph Explosion Report", which lists the decomposition of objects on a specified list of data flow diagrams, may be generated at the request of the analyst from outside the data flow diagramming tool. All undefined data store decompositions are indicated in this report.

27. All inputs and outputs of a primitive process specification must match those of the corresponding parent process on the data flow diagram.

Visible Analyst Workbench 3.1 -- A primitive process specification may be entered as an attribute of the corresponding process entry in the data dictionary. Listed immediately below the primitive process specification are the inputs and outputs to the process. However, there is no way of verifying if inputs and outputs from the parent process are present in the primitive process specification.

Excelerator 1.9 -- When a primitive process specification is created from a parent process the inputs and outputs are not carried down from the process to the primitive process specification. Violations of this rule are indicated to the analyst in the "Level Balancing" report. This report may be generated at the request of the analyst from outside the data flow diagramming tool.

28. A primitive process specification must be labeled with the same identifier as the corresponding primitive process on the data flow diagram.

Visible Analyst Workbench 3.1 -- A primitive process specification is an attribute of a process as is the label of the process. Therefore, the name of the process and the name of the primitive process specification are automatically the same.

Excelerator 1.9 -- When defining a decomposition path from a process to a primitive process specification the analyst may use any label desired.

The implementation of the methodology rules employed by Visible Analyst Workbench 3.1 and Excelerator 1.9 are summarized in Tables 4 and 5.

Table 4 Summary of Visible Analyst Workbench 3.1 Methodology Enforcement

	During Creation			During Exit/Save		Post	
	Automatic		On	Automatic		On	No
Rule #	Mandatory	Override	Request	Mandatory	Override	Request	Enforcement
1							▨
2							▨
3							▨
4		▨					
5	▨			▨			
6							▨
7	▨						
8			▨				
9			▨				
10		▨					
11			▨				
12			▨				
13		▨					
14			▨				
15							▨
16		▨					
17			▨				
18		▨					
19			▨				
20							▨
21		▨					
22			▨				
23			▨				
24			▨				
25							▨
26							▨
27							▨
28	▨						
	Level 1 Restriction	Level 1 Active Guidance	Level 1 Passive Guidance	Level 2 Restriction	Level 2 Active Guidance	Level 2 Passive Guidance	No Enforcement

Table 5 Summary of Excelerator 1.9 Methodology Enforcement

	During Creation			During Exit/Save		Post-Method	
	Automatic		On	Automatic		On	No
Rule #	Mandatory	Override	Request	Mandatory	Override	Request	Enforcement
1							▨
2						▨	
3							▨
4							▨
5							▨
6							▨
7							▨
8						▨	
9						▨	
10						▨	
11							▨
12*						▨	
13						▨	
14						▨	
15						▨	
16						▨	
17*						▨	
18						▨	
19							▨
20						▨	
21						▨	
22						▨	
23						▨	
24'							
25						▨	
26						▨	
27						▨	
28							▨
	Level 1 Restriction	Level 1 Active Guidance	Level 1 Passive Guidance	Level 2 Restriction	Level 2 Active Guidance	Level 2 Passive Guidance	No Enforcement

* Special cases of this violation may not be detected by Excelerator 1.9. ' Excelerator 1.9 does not support split data flows.

From Tables 4 and 5 it can be seen that the two CASE tools are quite different in their approach to enforcing the structured analysis methodology rules. In fact, the intersection of the rule bases of the two CASE tools contains only two rules (#3, #6) that are enforced in an identical fashion. It can also be noted from Tables 4 and 5 that it is inappropriate to apply a label such as "restrictive" or "guided" to either of these CASE tools. VAW implements only 10.7% (3 of 28) of the rules in a restrictive fashion, 60.7% (17 of 28) are implemented in a guided fashion (Level 1 Active and Level 1 Passive Guidance), and the remaining 28.6% (8 of 28) of the rules are not implemented. Excelerator implements 64.3% (18 of 28) of the rules in a guided fashion (Level 2 Passive Guidance) and the remaining 35.7% (10 of 28) of the rules are not implemented. While both CASE tools claim to support Yourdon structured analysis it is clear that their respective implementations of the methodology are quite different.

In this section a method has been described for classifying CASE methodology support at the level of the individual methodology rules. The following section describes how methodology consistency has been operationalized in previous studies and how the classification introduced in this section will be used to operationalize methodology consistency.

3.4 Research Variable: Methodology Consistency

The term "quality" as it applies to systems and software is too high-level to be meaningful or measured directly. By focusing on the user view of the system, key attributes of quality (quality factors) have been identified. Like quality itself, these factors are too high-level to have meaning and have been further decomposed into quality criteria. These quality criteria have been associated with a set of directly measurable attributes called quality metrics (Fenton, 1991).

One of the criteria identified as having an effect on several of the quality factors is consistency. "Consistency provides uniform design, code, and test techniques, as well as uniform system documentation. Standards for design techniques, coding, naming, and documentation all serve to improve consistency" (Arthur, 1985, p. 132). The consistent use of a design methodology improves the comprehension of the system by both the developers and the users by providing a common language for the duration of the project. Flexibility, i.e., the ability to make enhancements to the system, is improved when a consistent set of standards are applied to the systems development project. Maintainability, i.e., the ability to fix errors, will also be enhanced through the use of uniform development techniques. Finally, system reliability, i.e., the extent to which the system performs without failure, is improved by adhering to an agreed upon set of design standards (Arthur, 1985).

System specification (primarily data flow diagrams) quality has been assessed in various ways in previous empirical research. Baram et al. (1990) analyzed data flow diagrams for their correctness, completeness, balance, and readability. Correctness and balance were measured by counting errors within the data flow diagrams. Yellen (1990) analyzed data flow diagrams and the accompanying data dictionary for correctness, completeness, and communicability. Yellen defined correctness to be "conformity to rules of data flow diagramming construction" (p. 501). Correctness was measured by evaluating the diagrams against eight criteria.[17] A weight was assigned to each criteria in order to reflect the relative importance of the criteria. For each criteria, a score was assigned between zero and 100. A total score for correctness was obtained by summing the weighted scores for each criteria. Finally, Frolick et al. (1993) examined data flow diagrams for "overall system quality". The criteria used to evaluate quality were completeness, clarity, simplicity, technical accuracy, and naming accuracy. Each criteria was assigned an integer value between one and seven. A quality score was reached by taking the mean of the scores received for the five criteria. Scores for each of the quality criteria were not reported. In this study, consistency will be measured with respect to the 28 methodology rules previously defined. Specifically, consistency will be

[17] The eight criteria roughly correspond to nine of the structured analysis methodology rules listed in Table 2.

operationalized by counting the number of violations of each rule per system specification.

3.5 Research Hypotheses

The process of top-down design dictates that the system components are continuously specified in increasing levels of detail. A methodology rule violation that is not immediately presented to the analyst may propagate through the system specification, leading to errors that may not be detected until later phases of the life cycle. Therefore, it is important for an error to be detected by the CASE tool and presented to the analyst as soon as it is feasible to do so.

Hypothesis 1: For a given methodology rule, the number of violations encountered in the system specification will increase as the rule implementation mechanism moves from left to right on the methodology support spectrum.

The corresponding null hypothesis is given as

Hypothesis 1_0: There will be no difference in the number of violations of a given methodology rule between two CASE tools that utilize the same mechanism (relative to the methodology support spectrum) to support the rule.

Specification consistency has been measured and reported in two of the three cited studies of specification quality. In both instances, specifications produced with the aid of a CASE tool, and its accompanying methodology support, were found to be more methodologically consistent than specifications produced without the aid of a CASE tool, i.e., manually.

Hypothesis 1a: A methodology rule violation will be encountered less frequently in the system specification if the rule is supported by a CASE tool than if it is not supported by a CASE tool.

In summary, it has been proposed that a comparison of methodology support is inappropriate if done at the level of the CASE tool. This is due to the fact that, within a particular CASE tool, the level of support varies from rule to rule. Instead, methodology support will be examined at the level of the individual rules. It is anticipated that a rule implemented in a more restrictive manner will be violated less frequently than a rule implemented in a less restrictive manner.

3.6 Summary

This chapter presented the overarching research model, the research concepts, and hypotheses. A model was introduced that described the relationship between the independent and dependent variables in the systems development process. The dependent measure was defined and hypotheses were presented which predict the effects of methodology support on system specification quality. Chapter 4 presents the research methodology for this study.

CHAPTER 4

METHODOLOGY

4.1 Introduction

This chapter presents the research methodology used in this study. First, the independent and dependent variables will be defined. The second section describes the research design employed in this study. Subsequent sections present the subjects, task, setting, and procedures.

4.2 Variables

The independent, dependent, and control variables in this study are all derived from the systems development research model presented earlier and will be explained in this section.

4.2.1 Independent Variable

The independent variable in this study is the level of methodology support provided by the CASE tool for a particular rule. Support for a methodology rule is an inherent characteristic of the CASE tool and, therefore, can not be directly manipulated. The

28 methodology rules are listed in Table 2 and the support for the rules provided by the CASE tools is described in Section 3.3.3.1.

4.2.2 Control Variables

In this study the systems requirements were controlled by requiring the project teams to use an identical set of requirements. Personnel experience was controlled by offering subjects a common training experience in the development methodology. A survey of subjects revealed a common background in information systems-related experience. Project teams were required to use Yourdon structured analysis to develop their system specification.

4.2.3 Dependent Variable

The dependent variable, methodology correctness, can be defined as the degree to which the system specification adheres to the rules of the chosen methodology. For each project, a complete set of analysis reports was generated and errors were noted from the reports. For those rules that were not implemented by the CASE tools, the system specification was manually verified. Two reviewers each evaluated half of the specifications. Errors attributed to not following the methodology were noted by the reviewers. A third reviewer verified the work of the first two evaluators and tabulated the errors.

4.3 Experimental Design

This research was conducted using a replicated project study. Replicated project studies examine objects across a set of teams and a single project (Basili, Selby, & Hutchens, 1986). Figure 14 illustrates the study. Sixteen four-person project teams analyzed a hotel information system.

Eight of the teams used Excelerator 1.9 while the remaining eight teams used Visible Analyst Workbench 3.1.

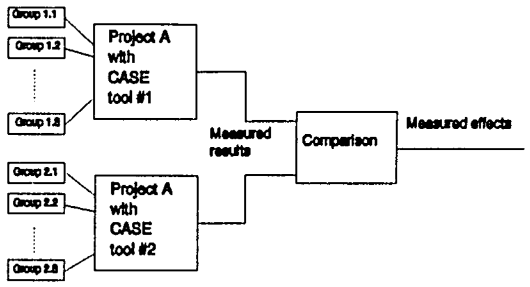


Figure 14 Research Design -- Replicated Project Study

4.4 Subjects

Moher and Schneider (1982) list seven subject characteristics that are of interest to the software engineering experimenter. These qualities are divided into two classes: those that are independent of systems development experience, and those that relate directly to systems development. Those characteristics that are independent of systems development experience include physical characteristics, intelligence, and level of education. Those characteristics of systems development experience include aptitude, skill, experience, and level of training.

Subjects were upper-division students (second-semester juniors or first-semester seniors) majoring in Management Information Systems and registered in an introductory systems analysis and design course. Prior to enrolling in this course, subjects have completed course work in business programming, data structures, and telecommunications. Additionally, subjects were required to be concurrently registered in a database design course.

A stratified random sample was originally planned in order to determine if differences existed in subjects' preparation coming into the systems analysis and design course. An extensive survey was given to the subjects (see Appendix A) requesting information regarding previous experience in the field of systems analysis, use of CASE tools, academic background, and so on. Results of this survey indicated that none of the subjects had any prior knowledge, through course work or experience in the professional world, with systems analysis and design or CASE tools. In light of this, a random sampling procedure was employed to assign subjects to groups. In order to avoid group attrition, the group assignments were not made until after the "midterm" examination and university "free drop" date.

After the groups were assigned, but before the project was assigned, groups were required to participate in group dynamics exercises in order to familiarize themselves with each other's personal qualities and academic strengths and weaknesses. As incentive to perform well on the project, 25 percent of the subjects' semester course grade was based upon the project deliverables. In order to discourage free-riding during the course of the project, subjects were told they would be submitting a post-project evaluation (read only by the instructor) of their group members' individual contributions to the project. This evaluation would be factored into the final project grade.

4.5 Experimental Task

Before receiving the requirements specification, each group was required to visit a local hotel (each group chose a different hotel) in order to familiarize themselves with the application domain. A 1985 study by Adelson and Soloway indicates that analysts with experience in the task domain are better equipped to creatively integrate material. A set of questions regarding hotel information systems was provided to each group. Appointments were set up with the hotels, on-site interviews were conducted with information systems managers and users, and the results were presented in a report.

After presenting the findings from the hotel interviews each of the project teams received an identical requirements specification for a proposed hotel information system (see Appendix B). Brooks (1980) states that an experimental task used in software engineering research must be of an appropriate level of difficulty. Therefore, the hotel information system specification was derived both from a textbook (not available to the subjects) case study (Whitten, Bentley, & Barlow, 1989) suitable for an academic project and an IBM case study (International Business Machines, 1990) used to train employees in Joint Application Design (JAD) sessions. Required deliverables and due dates were clearly stated in the requirements specification. Required deliverables included a complete set of data flow diagrams, primitive process specifications, structure charts, and a data dictionary.

4.6 Experimental Setting

Moher and Schneider (1982) provide several categories of environmental characteristics that must be considered when performing experimental research in software engineering. These categories include characteristics of the training, aid and reference materials, and the physical environment. The rest of this section describes the experimental setting used in this study.

4.6.1 Training

The literature on CASE tool adoption suggests that users should be familiar with the development methodology before using the CASE tool (Alavi, 1993; Kemerer, 1992; Loy, 1993). Before the project and group assignments were made, all subjects received approximately 2 months (1 hour per day, 3 times per week) of instruction in information systems development, with a heavy emphasis on information systems analysis and, in particular, Yourdon structured analysis and design. To accompany their instruction in structured analysis, subjects were required to complete pencil-and-paper exercises in the Yourdon method.

Before the project groups were assigned to a CASE tool, subjects received background material regarding CASE tools and how CASE is used to support information systems development. This material was in the form of lecture notes dealing with the concept of CASE as well as case studies of organizations that have successfully used CASE as a systems development aid. As soon as the CASE tools were assigned, subjects received a demonstration of their assigned tool. Omar (1992) shows that a demonstration of CASE tools has a positive impact on learning and understanding information systems and CASE tools. The demonstration of the tool consisted of preliminary details such as how to login and logoff the tool, followed by a detailed look at the functional areas of the tool that would be used to support the development of the system specification, including the graphics tool (for drawing data flow diagrams and structure charts), the analysis facility (for checking the completeness and correctness of the specification), and the data dictionary. In order to eliminate the steep learning curve frequently associated with CASE tools (Burkhard, 1989; Jankowski & Norman, 1992; Slusky, 1989), subjects were required to complete previously assigned pencil-and-paper exercises with the use of their CASE tool. To assure that subjects were adequately exposed to their group's CASE tool, the replication of the pencil-and-paper exercises was made an individual assignment.

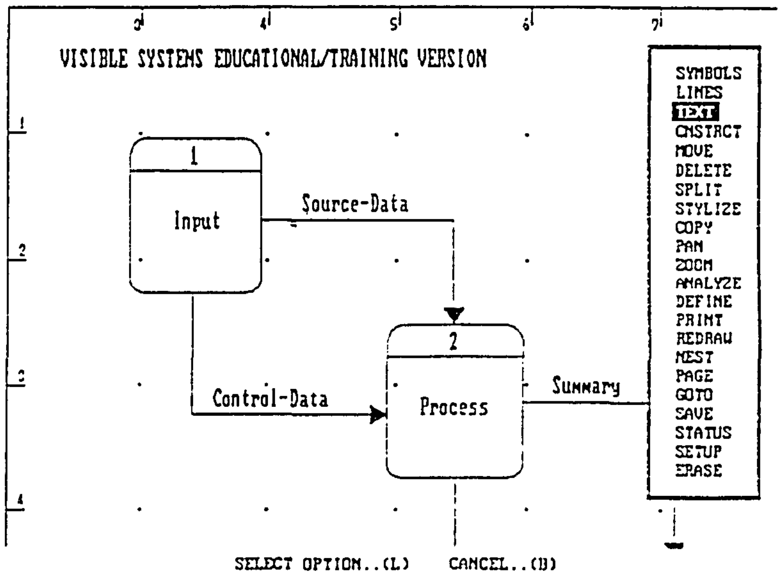
4.6.2 Reference Materials

To assist the groups in the completion of the project, several aids and reference materials were made available. After seeing a demonstration of their assigned CASE tool, groups were provided with a third-party manual and tutorial for their product (Schmidt, 1992; Wenig, 1991). Subjects also had access to vendor-supplied documentation and tutorials. To provide additional support, a "CASE tool expert" was available to the students for approximately fifteen hours per week.

4.6.3 Software

Eight of the sixteen groups were assigned to Visible Analyst Workbench 3.1 while the remaining eight groups were assigned to Excelerator 1.9. Before the projects were assigned to the groups the CASE tools were initialized by the "CASE tool expert". Visible Analyst Workbench was initialized to support the Yourdon structured analysis rules, the data dictionary, and automatic labeling of symbols and flows. Excelerator was initialized with the Yourdon data flow diagram symbol set. Both CASE tools support the analysis and design phases of the life cycle and the accompanying techniques necessary to complete a system specification. Each tool uses a graphical user interface that relies extensively on a mouse for input (see Figure 15).

Differences between the CASE tools as they relate to this study are described in Section 3.3.3.1.



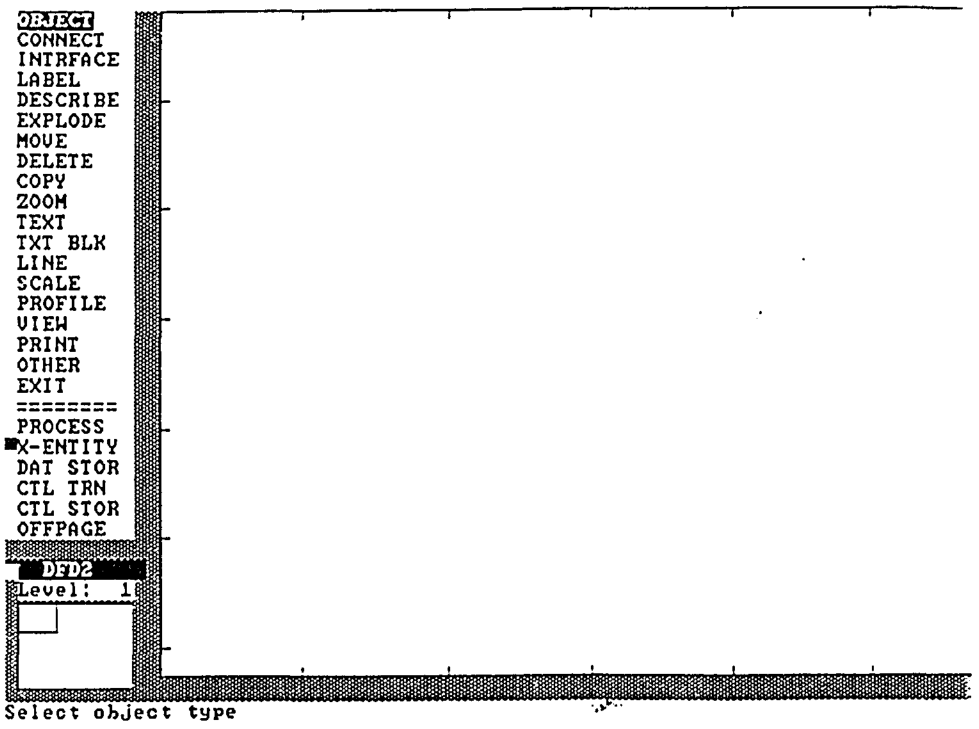


Figure 15 User Interface for VAW 3.1 (top) and Excelerator 1.9 (bottom)

4.6.4 Hardware

CASE tool usage occurred in a microcomputer laboratory dedicated to systems analysis and design courses. Each group was assigned a particular computer for the duration of the project. Computers were AT&T 6312 WGS workstations with 1Mb of RAM, a 68Mb hard disk, EGA monitor, and mouse. Each computer was connected to an IBM Graphics compatible Okidata line printer. Both CASE tools were fully compatible with the chosen hardware platform. Groups were assigned times during the week in which they could use the computer. Each group was given approximately 30 hours of CASE tool access time per week. Lab access time was between the hours of 9 a.m. and 8 p.m., Sunday through Saturday. Lab time was scheduled in multiple hour blocks on random days with the schedule changing each week to assure equal access for all groups.

4.7 Experimental Procedures

After becoming familiar with the application domain and the CASE tools (described above), the project was assigned. All deliverables, due dates, and grading criteria were discussed with the groups. Project deliverables were graded on the following criteria: 1) Were the system requirements satisfied? 2) Was the CASE tool output presentable? and 3) Were the project deliverables logically organized for submission to a hypothetical programmer? (It should be noted that, while methodology adherence was a stated grading criteria, in order to avoid penalizing the users of a particular

CASE tool, methodology adherence was not used as a grading criteria for the projects. However, during their methodology training, the subjects were versed in the importance of methodologically correct specifications. Further, groups received training in how to properly check their specifications for methodology correctness.).

As part of the project deliverables, and to assure that adequate progress was being made by each group, the groups were required to submit weekly logs describing their CASE tool usage. Specifically, the groups were asked to discuss the functional areas of the system they had worked with as well as the CASE tool components used to assist in the analysis of the functional area. Besides describing their work, the groups were also required to record the amount of time spent in each functional area of the CASE tool (graphics, analysis, data dictionary, etc.) each time they used the CASE tool. In order to verify the usage logs, the groups were required to submit their project data dictionary (on a computer diskette) along with the usage logs.

At the end of the semester, each group was required to submit a project notebook. This notebook contained an executive summary of the hotel information system as well as a complete set of analysis and design specifications. An oral presentation of the system was also required. After all deliverables had been received, each subject was required to fill out a subjective questionnaire regarding their CASE tool. Elapsed time between the assignment of the project and the presentation of the deliverables was approximately two months.

4.8 Summary

This chapter described the research methodology employed for this study. Independent variables, and how they were controlled and manipulated, were introduced first, followed by the dependent variable and how it was measured. The research design, subjects, task, setting, and procedures were also detailed. Chapter 5 presents the results of the field experiment.

CHAPTER 5

RESULTS

5.1 Introduction

This chapter presents the results of the statistical analysis of the data collected in the field experiment described in Chapter 4. The statistical method used to analyze the project data was a one-way analysis of variance (ANOVA) performed in SPSS for Windows (Base System, Release 6.0). The following section presents summary statistics for the projects. Next, the results of the dependent measure (methodology correctness) described in Section 4.2.3 will be presented for each methodology rule identified in Table 2.

5.2 Project Summary

Before proceeding with the formal hypothesis tests, ANOVA tests were run on various sizing characteristics of the projects. While each team received the same requirements specification for the hotel information system, it was necessary to determine if the usage of one or the other of the CASE tools might confound the results by assisting in the creation of a set of specifications that significantly differed in size from that created with the other CASE tool. Boehm (1976) states that the number of specification errors is a function of the specification size. Thus, the following null hypothesis was tested:

For a given attribute of specification size, the count of a particular attribute will not be dependent upon the CASE tool used in creating the system specification.

Table 6 displays a summary of the sizing data for the projects. For each of the project size attributes (number of diagrams, maximum diagram decomposition depth, number of external entities, number of processes, number of files, number of data flows, number of data elements, number of primitive processes, number of input/output data flows to primitive processes, and number of split data flows), we fail to reject the null hypothesis at the level of 5%

Table 6 Summary of Project Size Attributes

Attribute	Mean (VAW/Excelerator)	Standard Deviation (VAW/Excelerator)	F Ratio	F Probability
Diagrams	7.9 / 8.1	.35 / 1.96	.13	.73
Depth	3.1 / 3.3	.35 / .46	.39	.55
External Entities	3.1 / 3.0	.35 / .00	1.00	.33
Processes	26.8 / 27.1	2.96 / 7.83	.02	.90
Files	6.5 / 6.1	.53 / .35	2.74	.12
Data Flows	53.6 / 59.4	5.73 / 7.27	3.09	.10
Data Elements	67.1 / 77.4	16.39 / 18.28	1.39	.26
Primitive Processes	20.0 / 20.0	2.73 / 7.23	.00	1.00
Primitive Process I/O	78.5 / 71.6	8.49 / 16.28	1.12	.31
Split Data Flows	.5 / .0	1.41 / .00	1.00	.33

5.3 Methodology Correctness

In this section the results of the ANOVA tests will be presented for each methodology rule enumerated in Chapter 3. The rule enforcement mechanisms for the two CASE tools were described in Section 3.3.3.1 and summarized in Tables 4 and 5. For all rules for which violations were observed, the mean number of violations for VAW groups and Excelerator groups will be presented along with the corresponding F ratios and F probabilities. The results are summarized in Table 7.

Process Rule

1. A parent process must be specified before a child process. The null hypothesis (no difference in the number of violations of this rule between the two CASE tools) was supported. Among all project groups there were no violations of this rule.

Product Rules (Internal Consistency)

2. A data flow diagram must have at least one process. The hypothesis that projects using VAW would have more violations of this rule than projects using Excelerator was not supported. Among all of the project groups there were no violations of this rule.

3. A data flow diagram must have no more than seven processes. The null hypothesis (no difference in the number of violations between the two CASE tools) was supported (VAW = 0.0, Excelerator = .125, F = 1.0, p = .33).

4. A context diagram must exist. The hypothesis that projects using VAW would have fewer violations of this rule than projects using Excelerator was not supported. Among all project groups there were no violations of this rule.

5. The context diagram must contain only one process. The hypothesis that projects using VAW would have fewer violations of this rule than projects using Excelerator was not supported. Among all project groups there were no violations of this rule.

6. The context diagram must contain at least one input from an external entity and one output to an external entity. The null hypothesis (no difference in the number of violations of this rule between the two CASE tools) was supported. Among all project groups there were no violations of this rule.

7. The context diagram process must be numbered zero (0). The hypothesis that projects using VAW would have fewer violations of this rule than projects using Excelerator was supported (VAW = 0.0, Excelerator = .75, F = 21.0, p = .0004).

8. A process must have at least one input data flow and one output data flow. Although the number of violations observed in the specifications created with VAW was, as predicted, less than the number of violations observed in the specifications created with Excelerator, the hypothesis that projects using VAW would have fewer violations of this rule than projects using Excelerator was not supported (VAW = .125, Excelerator = ..625, F = 2.95, p = .11) at a statistically significant level (5%).

9. A process must be connected to at least one of the following: data store, process, external entity. The hypothesis that projects using VAW would have fewer violations of this rule than projects using Excelerator was not supported. Among all project groups there were no violations of this rule.

10. A process must be labeled. The hypothesis that projects using VAW would have fewer violations of this rule than projects using Excelerator was not supported. Among all project groups there were no violations of this rule.

11. An external entity must appear for the first time on the context diagram. The hypothesis that projects using VAW would have fewer violations of this rule than projects using Excelerator was not supported. Among all project groups there were no violations of this rule.

12. An external entity must be connected to a process. The hypothesis that projects using VAW would have fewer violations of this rule than projects using Excelerator was not supported. Among all project groups there were no violations of this rule.

13. An external entity must be labeled. The hypothesis that projects using VAW would have fewer violations of this rule than projects using Excelerator was not supported. Among all project groups there were no violations of this rule.

14. A data flow must be an interface between a process and either a second process, a data store, or an external entity. The hypothesis that projects using VAW would have fewer violations of this rule than projects using Excelerator was not supported. Among all project groups there were no violations of this rule.

15. A data flow into (from) a data store must have a composition that is a subset of the data store's composition. The hypothesis that projects using Excelerator would have fewer violations of this rule than projects using VAW was not supported (VAW = 5.5, Excelerator = 29.75, F = 13.74, p = .002).

16. A data flow must be labeled. The hypothesis that projects using VAW would have fewer violations of this rule than projects using Excelerator was not supported. Among all project groups there were no violations of this rule.

17. A data store can only exist as an interface between two processes. The hypothesis that projects using VAW would have fewer violations of this rule than projects using Excelerator was not supported (VAW = 1.5, Excelerator = .375, F = 1.37, p = .26).

18. A data store must be labeled. The hypothesis that projects using VAW would have fewer violations of this rule than projects using Excelerator was not supported. Among all project groups there were no violations of this rule.

Product Rules (Hierarchical Consistency)

19. A parent data flow diagram must exist unless it is a context diagram. The hypothesis that projects using VAW would have fewer violations of this rule than projects using Excelerator was supported (VAW = 0.0, Excelerator = 1.13, F = 6.52, p = .02).

20. A process must decompose to either another data flow diagram or a primitive process specification. Although the number of violations observed in the specifications created with Excelerator was, as predicted, less than the number of violations observed in the specifications created with VAW, the hypothesis that projects using Excelerator would have fewer violations of this rule than projects using VAW was not supported (VAW = .75, Excelerator = .13, F = .98, p = .34) at a statistically significant level (5%).

21. A process must be numbered with respect to its parent. Although the number of violations observed in the specifications created with VAW was, as predicted, less than the number of violations observed in the specifications created with Excelerator, the hypothesis that projects using VAW would have fewer violations of this rule than projects using Excelerator was not supported (VAW = 0.0, Excelerator = .5, F = 1.0, p = .33) at a statistically significant level (5%).

22. An input (output) data flow on a parent data flow diagram must appear on a child data flow diagram as input (output). The hypothesis that projects using VAW would have fewer violations of this rule than projects using Excelerator was supported (VAW = .75, Excelerator = 10.375, F = 7.15, p = .02).

23. An input (output) data flow on a child data flow diagram must appear on a parent data flow diagram as input (output). Although the number of violations observed in the specifications created with VAW was, as predicted, less than the number of violations observed in the specifications created with Excelerator, the hypothesis that projects using VAW would have fewer violations of this rule than projects using Excelerator was not supported (VAW = 3.13, Excelerator = 12.25, F = 2.66, p = .13) at a statistically significant level (5%).

24. A set of input data flows on a child data flow diagram that were split from a data flow on a parent data flow diagram must match the parent data flow's composition. The hypothesis that projects using VAW would have fewer violations of this rule than projects using Excelerator was not supported. Among all project groups there were no violations of this rule.

25. A data flow must decompose to either a record definition or an element definition. The hypothesis that projects using Excelerator would have fewer violations of this rule than projects using VAW was not supported (VAW = 10.75, Excelerator = 20.38, F = 1.24, p = .29).

26. A data store must decompose to either a file definition or a record definition. The hypothesis that projects using Excelerator would have fewer violations of this rule than projects using VAW was not supported. Among all project groups there were no violations of this rule.

27. All inputs and outputs of a primitive process specification must match those of the corresponding parent process on the data flow diagram. Although the number of violations observed in the specifications created with Excelerator was, as predicted, less than the number of violations observed in the specifications created with VAW, the hypothesis that projects using Excelerator would have fewer violations of this rule than projects using VAW was not supported (VAW = 16.5, Excelerator = 11.75, F = 2.17, p = .16) at a statistically significant level (5%).

28. A primitive process specification must be labeled with the same identifier as the corresponding primitive process on the data flow diagram. Although the number of violations observed in the specifications created with VAW was, as predicted, less than the number of violations observed in the specifications created with Excelerator, the hypothesis that projects using VAW would have fewer violations of this rule than projects using Excelerator was not supported (VAW = 0.0, Excelerator = 7.88, F = 3.3, p = .09) at a statistically significant level (5%).

Table 7 Summary of Comparison of Rule Violations Between CASE Tools

Rule #	Predicted # of Violations	Actual (mean) # of Violations		Significance
		VAW	Excelerator	$\alpha = .05$
1	VAW = Excel.	0.00	0.00	
2	VAW > Excel.	0.00	0.00	
3	VAW = Excel.	0.00	0.13	.3343
4	VAW < Excel.	0.00	0.00	
5	VAW < Excel.	0.00	0.00	
6	VAW = Excel.	0.00	0.00	
7	VAW < Excel.	0.00	0.75	.0004
8	VAW < Excel.	0.13	0.63	.1080
9	VAW < Excel.	0.00	0.00	
10	VAW < Excel.	0.00	0.00	
11	VAW < Excel.	0.00	0.00	
12	VAW < Excel.	0.00	0.00	
13	VAW < Excel.	0.00	0.00	
14	VAW < Excel.	0.00	0.00	
15	VAW > Excel.	5.50	29.75	.0023
16	VAW < Excel.	0.00	0.00	
17	VAW < Excel.	1.50	0.38	.2620
18	VAW < Excel.	0.00	0.00	
19	VAW < Excel.	0.00	1.13	.0230
20	VAW > Excel.	0.75	0.13	.3396
21	VAW < Excel.	0.00	0.50	.3343
22	VAW < Excel.	0.75	10.38	.0181
23	VAW < Excel.	3.13	12.25	.1251
24	VAW < Excel.	0.00	0.00	
25	VAW > Excel.	10.75	20.38	.2850
26	VAW > Excel.	0.00	0.00	
27	VAW > Excel.	16.50	11.75	.1633
28	VAW < Excel	0.00	7.88	.0905

5.4 Summary

This results of the experimental study were presented in this chapter. The following chapter discusses the implications of the results as they pertain to the systems development process, systems analysts and CASE tool design.

CHAPTER 6

DISCUSSION OF RESULTS

6.1 Introduction

The previous five chapters presented the groundwork for this study. This chapter presents the principal research findings based upon the results presented in Chapter 5 and discusses the implications of these findings with respect to the systems development process, systems analysts, and CASE tool design. Finally, the limitations of this research will be presented.

6.2 Principal Research Findings

In this section the results presented in Chapter 5 will be examined and discussed in greater detail. This discussion will be organized by categorizing the methodology rules presented in Table 2.

6.2.1 Top-Down Design (Process)

Within the set of methodology rules that apply to data flow diagrams, only one rule (#1) applies to the top-down process of constructing the diagram set. While neither CASE tool provides a mechanism for the enforcement of top-down design, there were no violations of this rule by any of the project teams. Violations were determined by examining the date/time stamp on the individual diagrams. This result may be explained by the fact that the subjects were all information systems students who had recently learned structured analysis and had not been exposed to any other systems development methodologies. For more experienced analysts familiar with other development techniques, such as bottom-up design, different results may have been observed (Adelson & Soloway, 1985).

6.2.2 Data Flow Diagram (Internal Consistency)

Two rules (#2, #3) exist that apply to the number of processes on any data flow diagram. There were no violations of either of these rules. While VAW does not check for the existence of at least one process on a diagram (Excelerator, on the other hand, does), it is a trivial task for an analyst to look at a one page diagram and determine that at least one process exists. Similarly, while neither CASE tool enforces the rule (#3) that, to enhance readability, no diagram should have more than seven processes, there was only one project team that violated this rule (and only on one diagram). Again, because the scope of effect of this rule is confined to a single diagram, it is trivial for an analyst to count the number of process symbols on a diagram in order to determine if the rule has been violated.

6.2.3 Context Diagram (Internal Consistency)

Structured analysis dictates that a set of data flow diagrams is to be developed using the principles of top-down design, with the initial diagram serving to define the system's environment. This diagram, called the context diagram, has several rules (#4, #5, #6, #7) applied to it that do not apply to any of the other data flow diagrams in a set. In all instances there is a maximum of one violation per rule per project. This is due to the fact that there is only one context diagram per diagram set and all of the rules that apply to the context diagram are binary. For three of the four rules (#4, #5, #6) there were no violations with either CASE tool. In all three cases, adherence to the rule is easily confirmed by a visual inspection of the context diagram to verify the existence of only one process as well as input and output connected to the process.

It would appear as if adherence to the fourth context diagram rule (#7), requiring the context process to be numbered as zero (0), could also be easily verified through a visual inspection of the context diagram. In spite of this, seven of the eight groups using Excelerator did not number the context process as zero. Because VAW automatically numbers all processes there were no violations of this rule from the VAW teams. At first glance it would appear as if the choice of zero is purely arbitrary and that any number would suffice for the context process. However, the numbering of all subsequent processes is based upon using zero as a basis. Failure to adhere to a consistent numbering scheme will inhibit the understanding of the hierarchical relationship between processes. This will be discussed further in Section 6.2.9.

6.2.4 Process (Internal Consistency)

Two of the internal consistency rules affecting processes (#9, #10) were not violated by any of the project groups. In both cases (freestanding processes and process labels) it is easy to visually inspect a data flow diagram to determine if violations of these errors exist. It was to be expected that VAW groups would have no violations of the rule requiring processes to be labeled, as VAW automatically prompts the analyst to enter a label as soon as a process symbol is placed on a diagram.

While it is a simple task to visually inspect a diagram and determine if a process is freestanding, as the diagram becomes more complex and the number of data flows grows it may become more difficult to distinguish between input flows and output flows. As predicted, the project groups using VAW had fewer violations of the rule requiring that each process have at least one input data flow and one output data flow (#8). However, the difference between the tools was not quite enough to be statistically significant. It is important to note that a violation of this error will be propagated downward via the leveling process to the primitive process specifications.

The result will be indeterminate ("black hole") or impossible ("miracle") primitive process specifications.

6.2.5 External Entity (Internal Consistency)

For all three rules involving external entities (#11, #12, #13) it was hypothesized that the VAW groups would have fewer errors than the Excelerator groups. In fact, there were no violations of any of the three errors by any of the project groups. As with the process rules described above, adherence to two of the external entity rules (freestanding external entities and external entity labels) is easily verified by a visual inspection of the diagram. The third rule deals with the role an external entity plays with respect to the context diagram. An external entity serves as a provider of input to the system or a receptor of output from the system. In either case, an external entity is not part of the system and is, therefore, only shown on the context diagram. As with the other rules that apply to external entities, adherence to this rule may also be easily verified by visually inspecting the data flow diagrams. Verification of these rules is made even easier due to the fact that external entities are typically confined to the context diagram.

6.2.6 Data Flow (Internal Consistency)

Two of the rules applying to data flows (#14, #16), as they relate to the internal consistency of data flow diagrams, were not violated by any of the project groups. In both cases it was predicted that VAW groups would have fewer violations than Excelerator groups. One of the two rules with no violations (data flow labels) is easily checked through a visual inspection of the data flow diagrams. The second rule applying to data flows for which no violations were observed is the requirement that a data flow serve as an interface between a process and some other object (process, external entity, data store). Neither CASE tool allows a data flow to be drawn freestanding and both CASE tools assume that a data flow connected to only one object has been brought down from the data flow diagram sitting above it in the leveling hierarchy (in which case rules #22 or #23 from Table 2 apply). This leaves a simple visual inspection of the data flow diagram to reveal instances when a data flow has been used to connect external entities or data stores to each other.

An examination of the third rule that applies to data flows (#15) reveals a result that is quite different from what was expected. A data flow that flows into (or from) a data store must be a subset of the data store's composition. While VAW does not support this rule and Excelerator does (Level 2 Passive Guidance), the VAW groups were found to have fewer violations of this rule than the Excelerator groups ($p = .0023$). A closer inspection of the guidance offered by Excelerator reveals that, rather than listing those data flows (and accompanying data elements) that are not subsets of the data store they flow into (or from), only the individual data elements that are not part of the data store's composition are listed. Due to the absence of the associated data flows, the analyst is required to browse through the data dictionary to determine the composition of all data flows entering the affected data store in order to determine which flows are in violation of the methodology rule. This problem is further compounded by the fact that many data flows have data elements in common. In this instance, the presence of guidance has hindered, rather than improved, the analysis of the data flow diagrams. It is also important to note that, due to the fact that a visual inspection of a data flow diagram will not reveal violations of this rule, both sets of groups encountered problems with this methodology rule.

6.2.7 Data Store (Internal Consistency)

There are two internal consistency rules that apply to data stores (#17, #18). One of the rules (data store labels) was not violated by any of the project groups. As with other data flow diagram objects, it is a simple matter to visually inspect a diagram for violations of this rule. The second data store rule was violated by both sets of groups (the difference was not statistically significant). A data store must serve as an interface between at least two processes, i.e., if a data store receives only net input or net output it should be modeled as an external entity rather than a data store. Because more than one process on a diagram is involved, a visual inspection of a particular diagram will not reveal violations of this rule. In order to find a violation of this rule, all diagrams within a particular level must be inspected. This inspection problem becomes more difficult as both the number of processes and the number of levels increases.

6.2.8 Summary of Internal Consistency Rule Violations

Seventeen rules have been identified as applying to the internal consistency of a data flow diagram. Twelve of these rules were not violated by any of the project teams. Adherence to all twelve of these rules is easily determined by visually inspecting a data flow diagram. For only two of the five rules for which violations were observed were the differences between the CASE tools statistically significant. In the case of the two rules that were violated most frequently, a visual inspection of a single data flow diagram is not sufficient to determine if the methodology rule has been broken. For these rules (#15, #17) it may be appropriate to provide a more rigorous degree of enforcement than either passive guidance or relying on the analyst to find the mistake himself.

6.2.9 Data Flow Diagram (Hierarchical Consistency)

In order for data flow diagram changes to be able to propagate to lower diagram levels, it is important for the CASE tool to be aware of the hierarchical relationship that exists between data flow diagrams in a set (#19). Without this awareness, any changes made to a diagram on level N must be manually implemented on levels N+1, N+2, etc. As was previously discussed in Section 2.3.4, the tedious nature of manually updating data flow diagrams was the cause of many organizations abandoning structured analysis.

In the analysis report provided by Visible Analyst Workbench, the analyst is made aware of any diagrams that do not have a parent (with the exception of the context diagram). Groups using VAW were observed to have no violations of this rule. On the other hand, Excelerator does not check for the existence of a parent diagram, i.e., diagrams may be created in any order desired and/or parent diagrams may be deleted at any time. All but two of the Excelerator groups had at least one diagram that was not connected to its parent. Any changes (addition/deletion of a data flow, external entity, etc.) made to the "logical" parent diagram of a child diagram will not appear on the child diagram because the CASE tool does not recognize the "physical" relationship between the diagrams. The most likely consequence of this violation is a failure to propagate data flows from a parent diagram to a child diagram. This will be examined in Section 6.2.11.

6.2.10 Process (Hierarchical Consistency)

To ensure the completeness of a set of data flow diagrams it is necessary that all processes be decomposed to a primitive process specification (#20). In other words, the child of every process on a data flow diagram must be either another data flow diagram or a primitive process specification. VAW does not directly support this rule, leaving the analyst to browse through the data dictionary to verify that all processes have been completely decomposed. However, as was discussed in Section 6.2.9, VAW does report when a diagram does not have a parent (the corollary to a process not having a child). Excelerator provides the analyst with decomposition information in its data flow diagram "Graph Explosion" report (obtained via Level 2 Passive Guidance). As expected, the Excelerator groups had fewer violations than the VAW groups; however, the difference was not statistically significant.

The second rule affecting the hierarchical consistency of processes (#21) concerns the method by which processes are numbered. At a maximum of seven processes per diagram, a system with four levels of data flow diagrams (not including primitive process specifications) will contain 58 diagrams and 400 processes. In order to keep the process hierarchy organized, and to quickly associate any process with its data flow diagram, it is necessary to adopt a numbering convention that relates a process to its parent diagram. Use of a hierarchical numbering convention would prove especially helpful when using the data flow diagrams as a communications tool in an environment, such as a JAD session or structured walk-through, with a diverse group of people associated with the systems development project (e.g., analysts, users, management).

Both of the CASE tools used in this study automatically number a process with respect to its parent; however, two distinct approaches are used. VAW numbers a process for the analyst when the process is placed on a diagram. The process number may be changed by the analyst after the process has been created (Level 1 Active Guidance). Excelerator also numbers a process for the analyst but not until the process is described in the data dictionary. At the time the process is described, the analyst has the option of changing the numbering. However, if a process is not placed in the data dictionary it will not be numbered. The analyst is not made aware of this until a data flow diagram analysis report is generated via Level 2 Passive Guidance. As expected, the groups using VAW had fewer violations of this rule than the groups using Excelerator; however, the difference was not statistically significant.

6.2.11 Data Flow (Hierarchical Consistency)

In order to ensure the hierarchical consistency of a set of data flow diagrams, it is crucial that all diagrams be level balanced, i.e., any data flow that appears as input (output) to a parent process must also appear as input (output) on the child diagram (#22). Similarly, a data flow that appears as input (output) on a child data flow diagram must appear as input (output) to the parent process (#23). By not propagating all data flows down through the diagram levels, the inputs and outputs to the primitive process specifications will be incomplete, resulting in incorrect algorithms. This will be discussed further in Section 6.2.13. As expected, groups using VAW (Level 1 Passive Guidance) had far fewer violations of these errors than groups using Excelerator (Level 2 Passive Guidance).

A third rule that serves to maintain the hierarchical consistency of the data flows (#24) applies to flows on a child diagram that have been split from the parent. Split data flows allow the analyst to package data into its logical form until the level of decomposition necessitates the use of individual data elements. VAW supports the use of split data flows while Excelerator does not. Only one of the sixteen project groups utilized split data flows; therefore, no specific conclusions can be made as to the usefulness of ensuring the integrity of the composition of a split data flow. However, the results presented in the previous paragraph indicate that the presence of CASE tool support for verifying data flow balance is useful for ensuring consistent specifications. Because split flows must be balanced between levels, it can be assumed that support for this rule would be of use in helping to maintain consistent specifications.

A final rule applying to data flows concerns the level of abstraction represented by a data flow (#25). A data flow is intended to represent a "packet" of data as it moves through the system. In order to prepare for the design of an information system it is necessary to decompose all data flows into their component records and/or elements. Failure to do this can result in an incomplete set of inputs and outputs to the primitive process specifications as well as an incomplete data dictionary. Although VAW does not provide support for the enforcement of this rule, while Excelerator uses Level 2 Passive Guidance, the groups using VAW had fewer violations of this error than did the Excelerator groups. This result may be explained by the contents of the "Graph Explosion" report used by Excelerator to indicate violations of this error. Rather than simply listing the data flows that have yet to be decomposed, the report lists the decomposition of every object (data flow diagram, process, external entity, data store, and data flow) for an entire diagram level (or the entire project). For even a small project this results in a lengthy report that is not easy to read.

6.2.12 Data Store (Hierarchical Consistency)

As with a data flow, a data store is an abstract item. While data flows represent data in motion, data stores represent data at rest. Underlying the abstraction shown on a data flow diagram is a data structure consisting of records and elements which must be specified in order for the functional specifications to be considered complete and ready to be sent to design. Further, without specifying the structure of the data store, the CASE tool is unable to verify that data flowing into or out of a data store is a subset of the data store. Excelerator uses Level 2 Passive Guidance to report violations of this rule while VAW does not enforce this rule in any fashion. However, there were no violations of this rule from either set of groups. This may be due to the fact that, although the Excelerator groups had to manually scan a report and the VAW groups had to manually scan the data dictionary, the number of data stores in the hotel information system was a manageable number for all of the groups (either six or seven data stores). This small number of data stores (both absolutely and relative to the number of data flows) made the verification process a simple one.

6.2.13 Primitive Process Specification (Hierarchical Consistency)

The leveling of a process is considered to be complete when the process has reached its primitive point. A process is generally considered to be a primitive process when it can be expressed in one page of pseudocode or some other algorithmic specification language. As with the leveling of any non-primitive process, the primitive process specification's parent process data flows must be carried down to the primitive process (#27). Without these data flows, the process will be incorrectly defined. Further, no other inputs and outputs may be introduced at the primitive process level. Excelerator uses Level 2 Passive Guidance to assist the analyst in "balancing" a primitive process with its parent process while VAW does not support this rule. As expected, groups using Excelerator had fewer violations of this rule than did groups using VAW; however, the results were not statistically significant.

It is interesting to note that the Excelerator groups were observed to have more violations of the data flow balancing rules discussed in Section 6.2.11 than the VAW groups. While the number of violations of rules 22, 23, and 27 is consistent for the

Excelerator groups (10.38, 12.25, and 11.75 respectively), the number of violations of rules 22, 23, and 27 showed a significant increase for the VAW groups (.75, 3.13, 16.50). As the number of levels in a data flow diagram set increase the difference between the number of processes on level *n* (and data flows into and out of the processes) and level *n+1* increases in an amount proportional to x^{n+1} where *x* is the number of processes on level *n*. These results are consistent with what has been presented in the preceding sections, i.e., as the size of the domain to be verified increases (in this case the number of processes) the opportunity to break a methodology rule also increases, thus necessitating rule verification assistance from the CASE tool.

The final rule applying to primitive process specifications (#28) serves to ensure a consistent naming convention for the primitive processes relative to their parent process. As was discussed in Section 6.2.10, it is important to maintain a naming convention that respects the hierarchical ordering of the process. By naming a primitive process specification for a parent process it becomes easier to relate a particular primitive process to its companion data flow diagram, as might be done in a JAD session or structured walk-through. VAW maintains the primitive process specification along with its parent process, thus forcing the names of the two to be identical. Excelerator, on the other hand, allows the analyst to apply any name to a primitive process. As expected, groups using VAW had no violations of this rule, while groups using Excelerator averaged nearly eight violations per group. This result fell just short of being statistically significant.

6.2.14 Summary of Hierarchical Consistency Rule Violations

Ten rules have been identified as applying to the hierarchical consistency of a set of data flow diagrams. Violations of eight of these rules were observed for at least one set of groups. Of the two rules for which no violations were observed, adherence to one of them (#26, data store decomposition) is easy to verify by browsing through the data dictionary. The second rule (#24, split data flow composition) was not violated because fifteen of the sixteen project groups did not use any split flows. For two of the eight rules for which violations were observed, the differences between the sets of groups were found to be statistically significant at the 95% level. For seven of the eight rules, however, the differences between the sets of groups was observed to be in the predicted direction.

Unlike the internal consistency rules summarized in Section 6.2.8, which are applied to only one diagram at a time, the hierarchical consistency rules cannot be verified by visual inspection of a single data flow diagram. As the system becomes more complex, the number of "links" between the diagram levels grows exponentially. This may explain the inconclusive results obtained in previous studies that examined the impact of CASE on the construction of data flow diagrams. These studies (summarized in Section 2.4.2.2) all had in common the fact that the experimental task was small, typically a context diagram and a level zero diagram. The small systems used in these studies contained only a small amount of decomposition and had no primitive process specifications to construct. The results presented in Section 6.2 indicate that as a project grows in size, the number of methodology violations also increases, thus necessitating methodology support from the CASE tool.

6.2.15 Summary of Methodology Rule Violations

Besides the results presented in the preceding sections, some other interesting observations can be made about CASE methodology enforcement. VAW implements nine of its methodology rules (#4, #5, #7, #10, #13, #16, #18, #21, #28) using either Level 1 Restriction or Level 1 Active Guidance. Seven of these nine rules apply to the internal consistency of a diagram, which is typically easy to verify through visual inspection. None of the project groups using VAW were observed to have violated any of these rules. Excelerator uses either Level 2 Passive Guidance or no enforcement for these same nine rules. Three of these nine rules (#7, #21, #28) were observed to have been violated by the Excelerator groups with the difference between the sets of groups being substantial for two of the rules (#7, #28).

Another interesting comparison may be made by looking at rule enforcement mechanisms that are adjacent on the methodology enforcement spectrum (see Figure

16). There are six instances of rules where Excelerator uses Level 2 Passive Guidance and VAW uses no enforcement (#2, #15, #20, #25, #26, #27). Differences between the sets of groups were observed for four of these rules (#15, #20, #25, #26) with only one of them being significant (in the opposite direction of what was predicted). This indicates that Level 2 Passive Guidance provides an insignificant amount of improvement in the quality of the functional specification beyond what the analyst can deliver by performing visual inspections of the data flow diagrams. A second set of rules (#8, #9, #12, #14, #17, #22, #23) are enforced by VAW with Level 1 Passive Guidance and by Excelerator with Level 2 Passive Guidance. Differences between the sets of groups were observed for four of these rules (#8, #17, #22, #23) with one of the differences being significant and two of the differences being close to significant. This indicates that being able to obtain guidance while working on a data flow diagram may be more helpful than being forced to exit the diagramming tool before obtaining guidance.

6.3 Implications for Systems Development

The systems development waterfall life cycle is an orderly model for managing the systems development activities, with the output of each phase of the model being used as the input of the succeeding phase. The structured techniques are a set of methodologies used to implement each of the phases of the life cycle. Recall from the discussion in Section 2.3 that the output of structured analysis, the functional specification, becomes the input to structured design. While the functional specification serves as a documented model of the proposed system, in the interest of clarity, many details, such as iterations, decision paths, and error control, are purposely omitted. Therefore, the functional specification is insufficient to act as the sole guideline for the coding of the system.

One of the activities of the design phase, which can be implemented via structured design, is the construction of the program structure through a transformation of the functional specification. This transformation, known as either transform analysis or transaction analysis, results in the creation of a set of structure charts. "A structure chart is a graphical representation of the program modules and their relationship to each other within a hierarchical structure" (Wu & Wu, 1994, p. 578).

The derivation of a set of program structure charts from a set of data flow diagrams is based on the principle that the structure of any system can be represented by three steps: input, process, and output. Transform and transaction analysis serve to identify, from the data flow diagrams, the system's input stream, central transform (or transaction center), and output stream. By utilizing the levels of the data flow diagrams, the structure charts are constructed in a top-down, hierarchical fashion, with the structure chart modules corresponding to data flow diagram processes and the structure chart data couples corresponding to data flow diagram data flows. Once the structure charts are complete, the modules are defined by expanding the primitive process specifications to include controls, error flags, and other input and output processing details.

Many metrics have been identified for use in evaluating the quality of structure charts. These metrics can be grouped into five categories: coupling, cohesion, complexity, modularity, and size (Troy & Zweben, 1981). Of these, the metrics most directly related to data flow diagrams are the coupling metrics. Coupling measures the strength of association established by the interconnections from one module to another. The degree of coupling is dependent upon the number and type of couples between modules (Yourdon & Constantine, 1979). Because the structure charts are derived directly from the data flow diagrams, it is crucial that all of the data flows be properly displayed on the data flow diagrams. In particular, when a process is decomposed its input and output data flows must be propagated to the next diagram level. A failure to correctly propagate the data flows through the data flow diagrams will result in incorrectly specified structure charts. Similarly, in order for the structure chart modules to be correctly specified, input and output data flows to the primitive processes must be propagated to the primitive process specifications. From Table 7 it is seen that five of the 28 methodology rules (#15, #22, #23, #25, #27) were violated much more frequently than any of the other rules. These five rules all impact the data flows (or their composition) and their inter-diagram relationships with processes and primitive process specifications. Of these five rules, the three violated most frequently (#15, #25, #27) were not enforced by Visible Analyst Workbench and were enforced with Level 2 Passive Guidance by Excelerator. The remaining two rules (#22, #23) were violated extensively by the Excelerator groups (supported with Level 2 Passive Guidance) but had very few violations among the VAW groups (supported by Level 1 Passive Guidance). Clearly, in order to ensure the consistency and correctness of the structure charts derived from the functional specification, it is important to provide support for the methodology rules that apply to the leveling of the data flow diagrams.

6.4 Implications for Systems Analysts

While the embedding of systems development methodology rules in CASE tools has been shown in this study to be an effective means of enforcing a development methodology, many advocates of automated development tools argue that restriction and/or guidance of the analyst must not be implemented at the expense of the analyst's ability to be creative. Crow (1990) argues that the decision to implement methodology support should take a back seat to creativity: "Creativity cannot be stifled" (p. 14). Page-Jones (1992) contends that many CASE tools are "draconian in their degree of methodology enforcement" (p. 38). Davis (1982) acknowledges that there exist situations when "detailed structure" may be necessary. However, Davis cautions that there are also situations when detailed structure may be "inhibiting and frustrating" (p. 12). Adelson and Soloway (1985) state that the level of support provided for an analyst by a tool should reflect the experience of the analyst with the problem domain and the design technique. Nunamaker, Dennis, Valacich, Vogel, and George (1993) advocate tools with a balance between restriction and flexibility, and indicate that too much restriction can "constrain creativity and exploration" (p. 135). Vessey et al. (1992) address the impact of CASE tool restriction and guidance on analyst creativity by advocating a development environment that adapts to the experience of the analyst. The first empirical work in this area is being conducted by Day (1993). The study, currently in the data collection phase, attempts to determine how analysts respond to CASE tool-imposed constraints during the systems development process. Until results from Day's study are reported, the idea that CASE stifles analyst creativity will merely be a "rule of thumb" that is subject to debate. However, as will be shown in the following section, the "draconian" image of CASE tool methodology enforcement proffered by Page-Jones (1992) is more a mistaken perception than a reality of CASE tool design.

While the subjects' personal perceptions of, and satisfaction with, their assigned CASE tool was not a variable in this study, all subjects were required to complete a post-project questionnaire regarding the use of their assigned CASE tool. Comments in response to the question about positive features of their assigned CASE tool included: "Excelerator made level balancing much easier than it would have been if done manually.", "Visible Analyst helped us follow the Yourdon methodology.", "The standards imposed by the CASE tool improved the quality of our project.", "Visible Analyst's automatic prompting for diagram object labels and automatic numbering of processes helped keep our project consistent.", and "VAW's graph analysis tool was extremely helpful to our group." While the students all had negative comments about their assigned CASE tools, none of the students indicated feeling constrained by the methodology support provided by their respective tools. Because the subjects in this study were all "novice" analysts, data collection from a more diverse pool of systems analysts is needed before substantive conclusions can be made about the effects of CASE methodology support on analyst creativity.

6.5 Feasibility of CASE Support for Structured Analysis

While all CASE tool vendors claim their tools support a particular methodology (or methodologies), and all of the vendors preach the importance of following a methodology for consistency and to aid in communication with users, the philosophy of methodology support differs quite substantially between CASE products. Some tools, such as BriefCASE and Excelerator, specifically indicate a flexible philosophy. In BriefCASE's documentation it states, "Rules may be bent or broken to enhance the objectives of clear communication" (Crow, 1990, p. 14). Excelerator's documentation states, "Excelerator doesn't require you to adhere to diagramming rules .... it does not require that you follow any or all of the rules it checks for"

(Intersolv, 1989, pp. 3-5, 6-23). On the other hand, some tools, such as Visible Analyst Workbench, indicate, in stronger terms, the importance of methodologically correct specifications and the role of the CASE tool in ensuring their correctness: "... the tool insures the integrity of the system in terms of both consistency and completeness .... by automating this ... the tool helps achieve the drastic shortening of analysis and design ... that is so strongly desired in software engineering" (Visible Systems, 1989, p. 24). Other tools, such as Cadre Technologies's Teamwork/PCSA, take the middle ground: "Teamwork/PCSA allows you to perform checks on your project" (Cadre Technologies, 1988, p. 2-20).

By adopting a particular systems development methodology and mandating its use, an organization is expecting the products of the systems development activities to conform to the chosen methodology. To achieve the objective of methodology prescription an organization, in theory, should be able to purchase a CASE tool that supports the chosen methodology. However, as has been discussed throughout this dissertation, the concept of methodology support can be handled differently from one CASE tool to another. Further complicating the issue is the fact that some methodology rules cannot, for practical reasons, be implemented in a restrictive fashion.

A discussion of the feasibility of automating methodology rules must begin with the conflict between an actual methodology violation and work in progress. The CASE tool should not interrupt the analyst's work to report a violation if the suspected violation may be a symptom of unfinished work (recall Page-Jones's (1992) "draconian" image of CASE tool methodology support). To account for this, a rule violation that may be the result of unfinished work should be handled in one of two ways: 1) The violation may be automatically presented to the analyst only when the analyst saves the diagram or exits the diagramming tool (Level 2 Active Guidance), or 2) The violation is only presented to the analyst upon the request of the analyst (Level 1 or Level 2 Passive Guidance). In either of the above methods, the analyst is not bothered with an erroneous interruption while drawing the diagram. While notification of violations via active guidance would have the probable effect of identifying possible inconsistencies earlier than would passive guidance, active guidance might also prove to be a nuisance to the analyst by forcing the analyst, every time a diagram is saved, to read through a list of violations that can be attributed to unfinished work. The remainder of this section examines the feasibility and practicality of automating the rules of structured analysis previously presented in Table 2. The feasibility of automating structured analysis methodology rules will be reported with respect to the scale shown in Figure 16. The scale displays the previously discussed methods of notifying an analyst of a rule violation, arranged in order of strong to weak enforcement rigidity.[18]

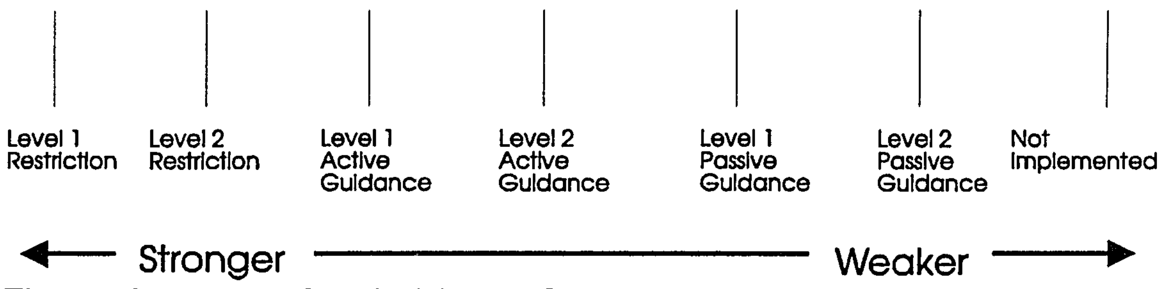


Figure 16 Spectrum of Methodology Enforcement

Process Rule

1. A parent process must be specified before a child process. This is the only methodology rule that enforces the process of top-down design. This rule may be restrictively (Level 1) enforced by a CASE tool by not allowing processes to be linked post-hoc, i.e., the only way new processes (with the exception of the context diagram process) may be created is through the decomposition and subsequent refinement of a parent process.

[18] For the methodology rules described in this section, implementation feasibility can propagate to the right (with respect to Figure 16), i.e., a rule that is listed as being able to be implemented as Level 2 Restriction can also be implemented as active guidance or passive guidance. However, a rule that is listed as being able to be implemented as active guidance cannot be implemented as either Level 1 Restriction or Level 2 Restriction. It has also been assumed that an error that could be attributed to unfinished work is to be treated as unfinished work. Therefore, when presenting the feasibility summary in Table 8, Level 1 Passive Guidance has been chosen over Level 2 Active Guidance.

Product Rules (Internal Consistency)

2. A data flow diagram must have at least one process. This rule cannot be enforced in a restrictive fashion because doing so will not take into account unfinished work. For example, a diagram in progress may contain data flows and data stores but not processes. Active guidance (Level 2) can be provided to the analyst while saving the diagram or exiting the diagramming tool. Passive guidance (Level 1 or Level 2) can be provided if the analyst does not wish to be distracted with automatic internal consistency checks on work in progress.

3. A data flow diagram must have no more than seven processes. This rule can be restrictively (Level 1) enforced by a CASE tool by not allowing the analyst to access a new process symbol if seven processes already exist on the data flow diagram.

4. A context diagram must exist. This rule can be restrictively (Level 1) enforced by a CASE tool by defining the first data flow diagram to be the context diagram and applying all other context diagram rules to this diagram.

5. The context diagram must contain only one process. This rule can be restrictively (Level 1) enforced by a CASE tool by not allowing the analyst to access a new process symbol if a process already exists on the diagram. However, the requirement that the context diagram must have a process cannot be restrictively enforced in order to account for unfinished work. Active guidance (Level 2) can be provided to the analyst while saving the diagram or exiting the diagramming tool.

Passive guidance (Level 1 or Level 2) can be provided if the analyst does not wish to be distracted with automatic internal consistency checks on work in progress.

6. The context diagram must contain at least one input from an external entity and one output to an external entity. This rule cannot be enforced in a restrictive fashion because doing so will not take into account unfinished work. For example, an unfinished context diagram might contain input from an external entity but no output to an external entity. Active guidance (Level 2) can be provided to the analyst when saving the diagram or exiting the diagramming tool. Passive guidance (Level 1 or Level 2) can be provided if the analyst does not wish to be distracted with automatic internal consistency checks on work in progress.

7. The context diagram process must be numbered zero (0). This rule can be restrictively (Level 1) enforced by a CASE tool by automatically numbering the context process for the analyst when the process is created and not allowing the analyst to change the numbering.

8. A process must have at least one input data flow and one output data flow. This rule cannot be enforced in a restrictive fashion because doing so will not take into account unfinished work. For example, an unfinished diagram might contain a process with an output data flow but no input data flow. Active guidance (Level 2) can be provided to the analyst when saving the diagram or exiting the diagramming tool. Passive guidance (Level 1 or Level 2) can be provided if the analyst does not wish to be distracted with automatic internal consistency checks on work in progress.

9. A process must be connected to at least one of the following: data store, process, external entity. This rule cannot be enforced in a restrictive fashion because doing so will not take into account unfinished work. For example, immediately after a process is drawn it is free-standing. Active guidance (Level 2) can be provided to the analyst when the diagram is saved or the diagramming tool is exited. Passive guidance (Level 1 or Level 2) can be provided if the analyst does not wish to be distracted with automatic internal consistency checks on work in progress.

10. A process must be labeled. This rule can be restrictively (Level 1) enforced by a CASE tool by automatically prompting the analyst to enter a label when the process is created and requiring the analyst to enter a label at the prompt.

11. An external entity must appear for the first time on the context diagram. This rule can be restrictively (Level 1) enforced by a CASE tool by verifying the name of any external entity placed below the context diagram with a list of names of those external entities appearing on the context diagram. If the external entity is appearing for the first time in the diagram set (but not on the context diagram) the CASE tool can disallow the placement of the external entity.

12. An external entity must be connected to a process. This rule has two possible scenarios, each of which requires a different enforcement mechanism. In the first scenario the external entity is free-standing and, therefore, in violation of the methodology rule. However, this may be attributed to work in progress rather than an error by the analyst. Active (Level 2) or passive (Level 1 or Level 2) guidance can be used to detect a free-standing external entity. In the second scenario, the analyst attempts to connect the external entity to a data store or another external entity. A CASE tool may prohibit this type of connection from being made (Level 1 Restriction).

13. An external entity must be labeled. This rule can be restrictively (Level 1) enforced by a CASE tool by automatically prompting the analyst to enter a label when the external entity is created and requiring the analyst to enter a label at the prompt.

14. A data flow must be an interface between a process and either a second process, a data store, or an external entity. This rule may be enforced in a restrictive (Level 1) manner by not allowing a data flow to be drawn as a free-standing object. Instead, a data flow can only be created by indicating the two existing objects that the flow is connecting. If one of the objects is not a process the CASE tool can prevent the data flow from being created.

15. A data flow into (from) a data store must have a composition that is a subset of the data store's composition. This rule cannot be enforced in a restrictive fashion because doing so will not take into account unfinished work. For example, the analyst can choose not to explicitly define, via the data dictionary, the composition of the data stores and/or data flows until after the diagram has been drawn. In this case, active guidance (Level 2) or passive guidance (Level 1 or Level 2) can be used to indicate any potential inconsistencies or the existence of an undefined data flow/store. Even if the CASE tool required the analyst to immediately enter the data dictionary after creating a data store or a data flow, the analyst must still be allowed to leave the composition definition unfinished.

16. A data flow must be labeled. This rule can be restrictively (Level 1) enforced by a CASE tool by automatically prompting the analyst to enter a label when the data flow is created and requiring the analyst to enter a label at the prompt.

17. A data store can only exist as an interface between two processes. This rule has two possible scenarios, each of which requires a different enforcement mechanism. In the first scenario the data store is free-standing or connected to only one process (and is not connected to the parent process) and, therefore, in violation of the methodology rule. However, this may be attributed to work in progress rather than an error by the analyst. Active (Level 2) or passive (Level 1 or Level 2) guidance can be used to detect this situation. In the second scenario, the analyst attempts to connect a data store to anything but a process. The CASE tool may prohibit this type of connection from being made (Level 1 Restriction).

18. A data store must be labeled. This rule can be restrictively (Level 1) enforced by a CASE tool by automatically prompting the analyst to enter a label when the data store is created and requiring the analyst to enter a label at the prompt.

Product Rules (Hierarchical Consistency)

19. A parent data flow diagram must exist unless it is a context diagram. This rule can be restrictively (Level 1) enforced by a CASE tool by only allowing a new diagram to be created (except for the context diagram) from a process decomposition resulting in a new (child) diagram level. Further, Level 1 restriction can be employed to prevent a diagram from being deleted if it has any child diagrams.

20. A process must decompose to either another data flow diagram or a primitive process specification. This rule cannot be enforced in a restrictive fashion because doing so will not take into account any unfinished leveling. Level 1 Active Guidance can be provided to the analyst by giving the analyst the option of creating a primitive process specification after creating the process. Level 2 Active Guidance regarding the completeness of the set of data flow diagrams can be provided to the analyst when saving a diagram or exiting the diagramming tool. Passive guidance (Level 1 or Level 2) can be provided if the analyst does not wish to be distracted with automatic completeness checks on work in progress.

21. A process must be numbered with respect to its parent. This rule can be restrictively (Level 1) enforced by a CASE tool by automatically numbering all processes when they are created and not allowing the analyst to change the numbering.

22. An input (output) data flow on a parent data flow diagram must appear on a child data flow diagram as input (output). This rule can be restrictively (Level 1) enforced by a CASE tool by automatically carrying down all input and output data flows from a parent process to a child diagram when moving between diagram levels and not allowing the net input and net output data flows to be deleted from the child diagram.

23. An input (output) data flow on a child data flow diagram must appear on a parent data flow diagram as input (output). This rule can be restrictively (Level 1) enforced by a CASE tool by not allowing insertions of net input and net output data flows on a child diagram.

24. A set of input data flows on a child data flow diagram that were split from a data flow on a parent data flow diagram must match the parent data flow's composition. This rule can be restrictively (Level 1) enforced by a CASE tool by automatically carrying down input data flows from a parent process to a child diagram when moving between diagram levels. Further, the CASE tool should not allow any of the sub-flows to be deleted from the child diagram nor may any sub-flows be added to the child diagram.

25. A data flow must decompose to either a record definition or an element definition. This rule cannot be enforced in a restrictive fashion because doing so will not take into account any unfinished work. Level 1 Active Guidance can be provided to the analyst by giving the analyst the option to enter the definition in the data dictionary after creating the data flow. Level 2 Active Guidance regarding the completeness of the set of data flow diagrams can be provided to the analyst when saving a diagram or exiting the diagramming tool. Passive guidance (Level 1 or

Level 2) can be provided if the analyst does not wish to be distracted with automatic completeness checks on work in progress.

26. A data store must decompose to either a file definition or a record definition. This rule cannot be enforced in a restrictive fashion because doing so will not take into account any unfinished work. Level 1 Active Guidance can be provided to the analyst by giving the analyst the option to enter the definition in the data dictionary after creating the data store. Level 2 Active Guidance regarding the completeness of the set of data flow diagrams can be provided to the analyst when saving a diagram or exiting the diagramming tool. Passive guidance (Level 1 or Level 2) can be provided if the analyst does not wish to be distracted with automatic completeness checks on work in progress.

27. All inputs and outputs of a primitive process specification must match those of the corresponding parent process on the data flow diagram. This rule can be restrictively (Level 1) enforced by a CASE tool by automatically carrying down input and output data flows from the parent process to the primitive process specification upon creation of the primitive process specification. Further, the CASE tool should not allow any of the inputs or outputs to be deleted from the primitive process specification nor may any inputs or outputs be added to the primitive process specification. To ensure that the inputs and outputs are actually used by the primitive process specification would require the CASE tool to parse the specification in order to find the inputs and outputs. In order to account for a specification "in progress", this procedure should only be done when exiting or saving the process specification (Level 2 Active Guidance) or at the request of the analyst (Level 1 or Level 2 Passive Guidance).

28. A primitive process specification must be labeled with the same identifier as the corresponding primitive process on the data flow diagram. This rule can be restrictively (Level 1) enforced by a CASE tool by automatically labeling the primitive process specification with the corresponding process label and not allowing the analyst to change the label.

Table 8 summarizes the methodology enforcement feasibility discussion presented in this section. From Table 8 it is seen that categorizing a CASE tool as "restrictive" is inappropriate. Ten of the twenty-eight methodology rules cannot be feasibly implemented in a restrictive fashion. It should also be noted that some form of methodology enforcement is possible for all of the 28 rules investigated in this study.

Table 8 Summary of CASE Tool Methodology Enforcement Feasibility

	Upon Creation			Upon Exit/Save		Post-Method	
	Automatic		On	Automatic		On	Not
Rule #	Mandatory	Override	Request	Mandatory	Override	Request	Feasible
1	▨						
2			▨				
3	▨						
4	▨						
5	▨		▨				
6			▨				
7	▨						
8			▨				
9			▨				
10	▨						
11	▨						
12	▨		▨				
13	▨						
14	▨						
15			▨				
16	▨						
17	▨		▨				
18	▨						
19	▨						
20		▨	▨				
21	▨						
22	▨						
23	▨						
24	▨						
25		▨	▨				
26		▨	▨				
27	▨						
28	▨						
	Level 1 Restriction	Level 1 Active Guidance	Level 1 Passive Guidance	Level 2 Restriction	Level 2 Active Guidance	Level 2 Passive Guidance	Not Feasible

6.6 Summary

This chapter discussed the implications of the research findings as they pertain to the systems development process, systems analyst creativity, and CASE tool design. The results indicate that rules applying to the hierarchical consistency of diagrams are violated more frequently than are rules applying to the internal consistency of diagrams. Adherence to the hierarchical consistency rules are crucial to the successful construction of structure charts in the design phase of the systems development life cycle. However, it is important not to let the desired outcome of methodologically correct specifications override the systems analyst's ability to be creative. Further, there exist many structured analysis methodology rules that are not feasible to enforce in a restrictive fashion, necessitating the systems analyst to take an active role in ensuring the specifications adhere to the rules of the methodology.

The following chapter summarizes this dissertation, describes limitations of this study, and suggests future research opportunities that could extend or refine this study.

CHAPTER 7

CONCLUSION

7.1 Introduction

This chapter serves to summarize the contributions of this research. Limitations of the experimental study and directions for future research will also be presented.

7.2 Contributions

Despite the hype surrounding CASE, and its claims to increase systems analyst productivity and information system quality, there is little in the way of empirical evidence to indicate CASE makes a substantial impact on the systems development process (Everest & Alanis, 1992; Kemerer, 1989). Previous experimental studies comparing the quality and consistency of functional specifications created with the aid of CASE tools versus those created manually have been, at best, inconclusive (Baram, Steinberg, & Nosek, 1990; Yellen, 1990; Frolick, Wilkes, & Rainer, 1993). In all three of the above studies, the experimental task was small and the subjects were under a short time constraint. Any potential positive impact of the CASE tool was overwhelmed by the subjects' frustrations with trying to learn the tool and complete the experimental task in the short time period allotted. For this dissertation, the subjects performed a complex task requiring the delivery of a complete functional specification. Further, in order to allow the subjects to do a thorough job in completing the specifications, the task was put into the context of a two-month term project. Before beginning the task, all subjects received extensive training in the use of both the methodology and the CASE tool.

Some of the results of this study echo those reported in previous experiments with CASE. When the unit of analysis is a single data flow diagram, the level of methodology support provided by the CASE tool does not appear to influence adherence to the systems development methodology. Due to the modular nature of the functional specification, a single data flow diagram represents only a small portion of the system being studied. Treated as an island, a single diagram is manageable enough to verify through a visual inspection. However, when the unit of analysis shifts to the entire project, and intra- and inter-level relationships are investigated, the number of methodology violations increases. Further, for many of the hierarchical consistency rules, the number of rule violations is dependent upon the level of methodology support provided by the CASE tool. The results indicate that CASE tools, in order to support and encourage analyst creativity, can "loosen up" the level of rule enforcement provided while drawing any particular data flow diagram, yet "tighten up" the enforcement of the rules involving the relationships between the diagrams.

The results presented in this study indicate that CASE tool support for a particular systems development methodology is not a binary question. Because support for a particular methodology has been identified in the literature as being an important issue to consider when choosing a CASE tool (Amundsen & Christoffersen, 1987; Baram & Steinberg, 1989; Boström, 1988; Burkhard, 1989; Everest & Alanis, 1992; Linos, 1992; McClure, 1989b; Rozman, Györkös, & Rizman, 1992; Shafer & Shafer, 1993; Subramanian & Gershon, 1991; Zucconi, 1989), CASE tool selection criteria may be further refined by comparing the level of methodology support offered by a particular CASE tool with the maximum feasible level of CASE tool methodology support. Although this study only looked at Yourdon structured analysis, a similar exercise could be performed with any methodology supporting structured analysis or the systems development life cycle.

7.3 Research Limitations

As with any experimental study, questions of external validity must be raised. All members of the project groups were undergraduate students in Management Information Systems. There has been significant debate in the literature over the appropriateness of using students as surrogates for professional systems developers (Moher and Schneider, 1982). In the professional world it is rare to find teams consisting only of novices. Professional systems development teams usually have the advantage of being able to work together for at least 40 hours per week. In order to maximize the efficiency and effectiveness of system development teams in the professional world, team assignments are generally made based upon the need for a set of particular skills rather than the random assignment method employed in this study (Robillard, 1989). The experimental task chosen for this study, while modeled after an actual development project in the professional world, had to be modified in order that it could be reasonably completed within two months.

Along with questions about external validity there are some internal validity issues that also need to be recognized. There were only eight project groups assigned to each CASE tool. While this allows project replication that is not possible in the professional world (Kemerer, 1989), the small sample size makes it difficult to obtain statistically significant results. Further, the small sample size allowed for the investigation of only two (of approximately 26) CASE tools that claim to support structured analysis. While all subjects were management information systems majors, and project team assignments were made after the deadline to drop the systems analysis and design course, there is no way to assure that all subjects were equally motivated to perform the experimental task.

7.4 Future Research

There are several future research opportunities that could extend or refine this study. These will be described in the following subsections.

7.4.1 Alternative User Interfaces

Both of the CASE tools used in this study, Excelerator 1.9 and Visible Analyst Workbench 3.1, were DOS-based. Since the completion of the study, both vendors (Intersolv and Visible Systems Corporation) have released Windows versions of their products. These new products provide the opportunity to replicate the study with an alternative, and increasingly more preferred, user interface. For both tools, the size of the methodology rule base as well as the level of enforcement for each rule is virtually unchanged between the DOS and Windows versions. However, the direct-manipulation interface employed by Windows allows the analyst to view multiple diagrams and/or methodology rule violations on the same screen. For example, Excelerator 1.9's Level 2 Passive Guidance requires the analyst to save the data flow diagram, exit the data flow diagramming tool, exit the graphics module, enter the analysis module, and choose the desired analysis report. Due to the inherent limitations of DOS, the analyst may not perform any of these tasks side-by-side. By using Excelerator 1.9 for Windows the analyst may visually inspect the hierarchical relationship between a parent and child data flow diagram or view analysis results in a window adjacent to a data flow diagram. The use of a direct-manipulation interface may also promote analyst creativity and, thus, restore any perceived ability to be creative that has been lost through the enforcement of the development methodology. For example, Roberts (1980) found that a direct-manipulation interface can greatly simplify a user's tasks compared to other forms of user interface. A direct-manipulation interface can provide the user with a strong feeling of mastery and competence. Users also feel more confident and are more willing to explore sophisticated aspects of a system when using a direct-manipulation interface (Hutchins, Hollan, & Norman, 1986).

7.4.2 CASE Methodology Support as a Pedagogical Instrument

The CASE adoption literature indicates that one of the critical success factors for CASE is to provide methodology training before CASE tool training (Alavi, 1993; Kemerer, 1992; Rozman, Györkös, & Rizman, 1992). Similar conclusions have been drawn regarding the use of CASE in academia (Jankowski & Norman, 1992). However, besides serving as a tool to assist an analyst in the construction of system specifications, a CASE tool, through its methodology support mechanisms, can also serve as a pedagogical instrument through which a novice analyst might be trained or an experienced analyst might learn a new systems development methodology. Research into computer-assisted learning shows that adaptive advice giving has been found to be desirable in computer-assisted instructional systems (Hannafin, 1984).

To determine if the use of CASE has a negative impact on learning introductory systems development concepts, including methodologies, Heiat and Heiat (1992) conducted two introductory systems analysis and design courses that used different approaches toward teaching a systems development methodology. No CASE tools were used during one of the courses while in the second course the students received hands-on training in CASE while they were learning a systems development methodology. In both classes the students were twice examined on their understanding of the course principles. No significant difference in performance was found between the two groups of students, suggesting that the use of CASE does not have adverse effects on student learning. Because the CASE tool chosen for the above study, BriefCASE, offers almost no methodology support, the opportunity exists to repeat this study with CASE tools offering various levels of methodology support.

7.4.3 CASE Methodology Support Effects on Structured Design

An obvious extension to this study is to investigate the impact that methodology correctness has upon the output of structured design. This can be investigated using two different methods. In the first method, the functional specification created with a CASE tool is used to derive a set of structure charts. The consequences of not having methodologically correct specifications can be measured by tracking methodology errors uncovered in the functional specification as they propagate through the design specifications. In the second method, a set of structure charts is derived with the assistance of a CASE tool and the number of methodology errors in the structure charts are noted. Vessey et al. (1992) have identified a set of structured design methodology rules whose CASE tool implementation mechanisms can be evaluated in order to compare the level of structured design support offered by various CASE tools. From these comparisons, and a study of the feasibility of automating structured design methodology rules, hypotheses can be generated and tested.

7.5 Conclusion

When CASE was introduced to the MIS profession it was touted as a means to obtain "revolutionary" advances in both system quality and analyst/programmer productivity. An examination of the empirical and anecdotal data on the impact CASE has had on the systems development process finds that the impact has been "evolutionary" rather than "revolutionary". It is the responsibility of MIS researchers and practitioners to identify areas in which CASE may improve the systems development process. One of the primary purposes of CASE is to serve as a companion to the systems development methodology used during the development process. This study has shown that the level of companionship (for structured analysis) provided by a CASE tool varies from tool to tool, and within a particular tool, from rule to rule. Further, the study indicates that the level of methodology support provided by a CASE tool for a particular rule has an effect on the number of violations of the rule. Further research under a variety of systems development scenarios may enable the identification of an "optimal" set of CASE tool methodology support mechanisms for the entire systems development life cycle. Through the assistance of these "optimal" methodology support mechanisms, the goal of “revolutionary” quality and productivity improvements may one day be realized.

APPENDIX A

SUBJECT BACKGROUND SURVEY

Name ______________________________

1. Which MIS and/or computer courses have you already taken? Please write the grade you received for the course in the appropriate place regardless of where you took the course. The numbers in parenthesis are the corresponding courses at the University of Arizona.

_____ Intro. to MIS (111)
_____ Intro. to Programming (121)
_____ Data Structures (301)
_____ Data Communication (307)
_____ Database (331)
_____ Other (please specify course)
_____ Other (please specify course)
_____ Other (please specify course)

2. Which, if any, of the above courses did you take at another school? Where were the course(s) taken?

3. Which MIS course(s) are you currently taking at the U of A?

4. To what extent are you familiar with the following programming languages? Please circle the appropriate response based upon the following code:
1 = I have never been exposed to this language
2 = I have been exposed to this language but have forgotten it
3 = I remember some of what I learned about this language
4 = I could write a program in this language with a reference book at my side
5 = I am fluent in this language

Pascal	1	2	3	4	5
FORTRAN	1	2	3	4	5
C	1	2	3	4	5
BASIC	1	2	3	4	5
COBOL	1	2	3	4	5
Assembly	1	2	3	4	5

5. Have you ever been registered in MIS 341 before this semester?

Yes No

If you circled yes, when were you registered for the course?

6. Do you have any experience with systems analysis and design outside of the classroom?

Yes No

If you circled yes, please explain.

7. Do you have any experience with CASE tools?

Yes No

If you circled yes, what tool(s) have you used?

8. Do you own a personal computer?

Yes No

9. Please answer the following questions:

Age ____________

Sex (circle one) Male Female

Highest level of education attained ______________________

What grade do you want to get in this course? ______________

What do you see yourself doing after you graduate?

Are there any circumstances which may make it difficult for you to handle the serious time commitment this course requires?

APPENDIX B

HOTEL REQUIREMENTS SPECIFICATION

INTRODUCTION

In this project, you will study the procedures used at the Wildcats Hotel for activities such as checking in guests, charging guests for services used, checking out guests, and producing reports for both management and guests. You are asked to undertake the analysis and design activities required for implementing a system to automate these procedures (note: you are not asked to implement the system). As part of the project, you will develop data flow diagrams, data dictionary entries, minispecs, and structure charts using the CASE tool assigned to you.

BACKGROUND

Wildcats Hotel is a popular hotel/resort in Tucson. Recently, the executive manager of Wildcats Hotel contacted the MIS department to ask for assistance. The hotel was in the process of developing an automated system for critical hotel functions such as guest check-ins, check-outs, etc. (more details are given in the following sections). The call was directed to the students of MIS 341, who were expected to have considerable knowledge in developing systems. The students agreed to help Wildcats Hotel in their efforts.

In the following sections, you will be given detailed descriptions of each of the major activities/processes that are involved. After all the major activities have been described, you will be given information on the actual data stores/files to be used by the system and the data elements that comprise each store.

MAJOR ACTIVITIES

Initial interviews with hotel management and staff suggest that the new information system would need to support normal business transactions, such as check-in, room billing and check-out. It would also need to handle charges from the restaurant, golf course, bars, pay TV, in-room beverage and snack service, etc. Additionally, the system would need to provide a full range of management reports and decision support information. Finally, because management is emphasizing customized services to hotel guests, the proposed system is also expected to enable guests to generate individual reports for their own use.

Currently, the hotel conducts its operations with a patchwork of separate systems (some automated and some manual) that address different functions. Moreover, these systems are often incompatible with each other, with the result that hotel staff frequently have to transfer information manually from one system to another. For example, there is a separate system for handling guest check-ins and a separate system for handling guest check-outs and billings, whereas charges (both room and non-room) are entered manually by the hotel staff. Whatever management reporting is undertaken is done manually.

Based on the results of interviews with the hotel management, the analyst identified the following six major activities to be supported by the proposed system: 1) guest registration (check-in), 2) guest charges (i.e., charges other than for the room), 3) guest room charges update, 4) report generation by guests, 5) guest check-out (i.e., billing and payment handling at check-out) and 6) report generation by hotel management. Each are described in detail below.

1. GUEST CHECK-IN

A new convention center scheduled to open next fall is expected to dramatically increase the number of guests using the hotel. Interviews with hotel staff suggest that the existing system for guest check-ins will not be able to handle this expected growth in volume. Some important problems in the existing system are:

a) Guests frequently have specific room preferences. The current system does not allow front-desk staff to identify rooms that are vacant in a particular category (single, double, suite, etc.). The interviews suggest that it would be strongly desirable to be able to give guests a list of rooms that are currently vacant in the category that they have indicated. Guests can then select a particular room for stay.

b) Once the system receives information on the specific room selection made by the guest, it then ensures that the room is no longer available (until the guest checks out of it). The system then sends this room information to the process described below.

c) Currently, there is no way of knowing whether the guest who is checking in has stayed at the hotel before. This can be done by including a store containing information on previous guests who have stayed at the hotel (apart from a store for guests who are currently staying). Information on previous guests can thus be moved into the store for current guests without having to reenter it.

When the guest checks in, it is also necessary that a record be created in the store for billing the guest. Currently, guests who check-in are not asked about the number of nights they expect to stay at the hotel. In the absence of such information, hotel management has found it difficult to forecast room occupancy and to schedule hotel operations accordingly. The proposed system is thus expected to capture this information at check-in. Information on the total room charges expected to be incurred by the guest can thus be maintained from the time of check-in in the file used to bill the guest.In addition, a record is also created in the store for processing guest payments.

2. GUEST CHARGES

Currently, although a wide variety of services (restaurant, bar, golf course, etc.,) are available in the hotel, guests are expect to pay "on the spot" for each service. This has frequently led to complaints from hotel guests who feel that this is a cumbersome procedure. Consequently, management has decided to allow guests to begin to charge a wide variety of services **to their room** after check-in. These charges will be added to the room bill and paid for by the guest at the time of check-out. Charges could include drinks at the hotel bars, meals at the restaurant, local tours, greens fees from the golf course, and many other services.

a) From the interviews, the analyst has identified that one major concern about allowing guests to charge to their rooms is that of fraud. Unless there is a mechanism for verifying that their guest does indeed stay in the room number he/she indicates at the time of making the charge, the wrong room may sometimes be billed for the service. The proposed system should thus address this concern by checking the guest name with the room number in which he/she is staying.

b) Once the room number has been verified, the proposed system should be able to capture information on the service being charged to the guest, at the time the charge is incurred. Interviews with staff suggest that two kinds of information should be stored by the proposed system with regard to the charge:

   1) The system should keep track of all charges made in the hotel, in order of the instant (accurate to the minute) at which the charge was incurred. Such information should be stored in a separate repository related to charges.

2) In addition to capturing information on the charge, the system should also update the total amount billed to the guest who is making the charge, at the instant that the charge is made. In the past, charges were billed to the guest only at the time of check-out. This frequently resulted in errors, disputes between hotel guests and staff, etc. Hotel staff feel that an immediate posting of the charge to the guest's bill would reduce these problems. Also, the proposed system should be able to generate a receipt (relating to the charge) for the guest, at the time the charge is made.

3. ROOM CHARGES

Although the proposed system should capture information on the number of nights that a guest expects to stay in the hotel, the hotel staff suggest, based on previous experience, that the number of nights a guest **actually** stays in the hotel is often quite different. In keeping with management's mission of generating accurate and timely information, the hotel staff feel that the system should at all times be able to track the actual number of nights spent in the hotel by the guest. Such information would be used subsequently in computing the total room charges that the guest had incurred at any given moment. Check-out time is officially 11 A.M, although a grace period of 1 hour is usually given. Any guest who has not checked out by 12 noon is charged for an additional day. Room charges are computed in whole number of days -- fractions of a day are not allowed. Previously, a manual process was used to identify those guests who had not checked out by 12 noon. However, this often led to errors and complaints.

4. GUEST REPORTS

Management was particularly excited about the possibility of enabling hotel guests to generate their own reports from the system. The executive manager of the hotel had attended a hotel managers convention in which some of the "leading edge" hotels had described how pleased their guests were with this service. The management of "Wildcats hotel felt that this facility (available via the TV set in each room) would give them a competitive advantage over other hotels in the area. The analyst identified the following reports which the guest could select from (note: only one report can be obtained for each request or transaction):

a) Total Bill Report After providing proper identification to the system, the guest could generate a report that would provide the current bill total, the current room charges total and the current charges (other than room) total that had accrued to the room the guest was staying in. In the past, such information was available only after going through a long and tedious procedure: the guest had to inquire at the front desk regarding his/her current bill total, and the hotel staff would subsequently have to search manually through different records to compute the total. Management felt that their guests would be very happy at being able to generate such information from the proposed system at the "touch of a button".

b) Details of Charges Report In the past, guests had been allowed to charge to their room. However, many guests had complained at check-out of erroneous charges that had been billed to them. Disputes over charges at check-out frequently led to nasty confrontations in the presence of guests who were waiting in line. Due to these problems, management had decided some years ago to discontinue the facility of allowing guests to charge to their room. Instead (as described earlier), guests were expected to pay for the service at the time the charge was incurred.

   Although management now felt that restoring the facility to pay for a charge at the time of check-out was desirable (since guests clamored for this privilege), it was nevertheless concerned that the old problems could recur. Unless corrected, this was especially likely to create problems during conventions, since a large number of hotel guests tended to check-out at the same time.

   The executive manager felt that enabling hotel guests to access information on the **details** of all their previous charges at any time during their stay could help reduce the problem. Hotel guests would then be able to immediately notify management of any errors, rather than wait until check-out. This would also give the hotel staff more time to correct the problem, rather than have to resolve it while the guest was checking out. Specifically, management felt that every hotel guest should be able to track each and every charge that had been made against his/her room up to that moment of his/her stay.

c) Events Report A wide variety of cultural activities and events take place year-round in Tucson. In the past, guests have complained that hotel staff often do not have adequate information on these cultural events. A year ago, the hotel developed a system to keep track of such cultural and other events. This information was used by the hotel staff to answer queries by customers. Management now feels that customer service can be improved by allowing hotel guests to access such information directly, instead of through intermediaries. The proposed system should thus allow the generation of such reports from the events system.

5. GUEST CHECK-OUT

The current check-out and billing system compiles the guest's bill at time of check out and is essentially a manual process. Several problems were identified with the current system: 1) an anticipated inability to handle the expected increase in the number of guests, 2) the amount of time necessary to process a guest check-out has increased considerably, 3) incorrect bills have resulted in a significant increase in customer complaints. After extensive interviews with management and staff, the analyst identified the following major functions that the proposed system was to perform:

a) At the time of check-out, the guest returned the room keys to the hotel staff who in turn had to produce an itemized bill that described each charge (room, non-room related) incurred by the guest during his/her current stay at the hotel. To serve as an additional verification of the charges, the proposed system had to establish that the sum of the individual charges made by the guest did indeed correspond to the total charges that were billed for. This itemized bill was then presented to the customer for payment.

b) Once the payment was received from the guest, the system had to update many different pieces of information. Specifically, the system had to ensure that the room that had been occupied by the guest was now restored to its vacant status (to enable other guests to move into it). Information on payments made by hotel guests also had to be updated with the information on this guest's payment. Moreover, since the guest was now checked out, information related to him/her had to be deleted from the list of guests who were being billed.

c) Finally, with the guest's status now changed to being a previous guest, information about the guest had to be included with information on previous guests. The system had to generate a receipt for the guest for the payments made by him/her.

6. MANAGEMENT REPORTS

In the past, the existing arrangements were particularly poor at generating relevant and useful information for hotel management. The executive manager returned from her conference convinced that there were a variety of reports which needed to be produced "at the touch of a button" for hotel management. Moreover, management had to find it very easy to use any proposed new system -- the system would have to ask them only the minimal information (such as the report being requested) and then go ahead and produce the desired report. From interviews, the analyst identified the following three kinds of reports that had to be produced (note: only one report will be generated per request or transaction):

a) Management was especially interested in knowing the funds flowing into the hotel. This information would be required for at least the past six months. The primary sources of funds were the payments made by hotel guests. Not only would such information reflect the inflow from a beginning date specified by the manager (and up to the current date), but would also be used later for any subsequent enhancements to the system.

b) Since the hotel offered a wide variety of hotel rooms (doubles, singles, suites, etc.), management was interest in knowing the percentage occupancy ratios (and information on the guests currently occupying them) for the different room types in the hotel. By putting together information about the current guests and about the type of rooms that they occupied, a wide variety of useful information could be accessed immediately by management.

c) Apart from reports on payments and room occupancy, management was also interested in obtaining a picture of the types of charges that were being made by hotel guests. For example, the executive manager foresaw a situation where she wanted to compare the revenue generated by the food section in the restaurant with the revenue generated by the bar. Such information would be useful in assessing the profitability of each of these chargeable services. In order to provide such information, the proposed system had to be able to access all the charges made at the hotel. It was not necessary; however, to identify the specific guest who was making a particular charge.

EXTERNAL ENTITIES (SOURCES AND SINKS)

The analyst identified the following external entities to the Wildcats Hotel system:

1. Hotel Management

Management was both a source (because it asked for certain reports) and a sink (because it received the desired reports).

2. Hotel Guests

Clearly, hotel guests were important sources and sinks. As sources, they provided numerous inputs to the system (as described above). As sinks, they received numerous outputs from the system.

3. Events System

The events system was an external system that had to provide information about events in Tucson when asked to by the hotel guests.

DATA STORES (OR FILES)

The analyst identified the following files for the system described earlier (note: assume that these are the only ones used):

1. CHARGE FILE

The basic purpose of this file is to track all charges made by all guests in the hotel, in the chronological order in which the charge is made. Each record consists of five elements, one of which (i.e., charge date) is itself a record. Each record is created and written to the file whenever a charge to a room is made. The file contains information on charges made to only those rooms which are currently occupied. When a guest checks out, all charges to that room are deleted from the file. The composition of the charge record is as follows:

- charge date
  - day of charge
  - month of charge
  - year of charge
  - hour of charge
  - minute of charge
- charge type
- charge description
- charge amount
- room number charged to

2. BILL FILE

The basic purpose of this file is to keep up-to-date information on the expenses incurred by a guest at the hotel. There is one record for each room currently occupied in the hotel. Each record consists of four elements, in addition to the room number. The record keeps a running total of all guest charges, room charges (room rate * number of nights stayed so far by the guest) and the total current bill (current charges + current room charges). It also contains information on the expected room charges for the guest (expected number of nights that will be stayed * room rate). A record is created and written to the file upon guest check-in. It is modified every time a guest makes a charge to the room and at 12 noon every day when the system computes the current room charges. The record is deleted from the file when a guest checks out. The fields in the record are:

room number
current charges total
current room charges total
current bill total
expected room charges total

3. ROOM FILE

The purpose of this file is to store basic information on each room in the hotel relating to its occupancy and its nightly rate. There is a record in this file for each room in the hotel. The file is read and modified at the time of guest check-in, and is updated at the time of guest check-out. The file is also accessed at 12 noon every day for identifying the room rate to be used in computing current room charges for the guest. Each record consists of the following four elements:

room number
room type
vacancy status
nightly rate

4. CURRENT GUEST FILE

The purpose of this file is to store relevant information (including personal information) on guests who are currently staying at the hotel. Personal information, such as name, address, company name and auto tag information is either entered for the first time (if the guest has never stayed before at the hotel) or taken from the **PAST GUEST FILE** if the guest has previously stayed at the hotel. There is a record for each guest in the hotel. Each record is created at the time of check-in and updated at the time of check-out. The file is also accessed when verifying guest id's each time a guest makes a charge, when computing the current room charges accumulated by the guest, and also when generating management reports. At the time of guest check-in, the actual check-out date fields in this record are set to zero. Each record consists of the following fields:

- room number
- check-in date
  - day of check-in
  - month of check-in
  - year of check-in
  - hour of check-in
  - minute of check-in
- actual check-out date
  - day of actual check-out
  - month of actual check-out
  - year of actual check-out
  - hour of actual check-out
  - minute of actual check-out
- guest name
  - guest last name
  - guest first name
- guest address
  - guest street number
  - guest apartment number
  - guest city
  - guest state
  - guest zip code
  - guest country
- guest phone number
- guest company name
- guest auto tag
  - guest tag number
  - guest auto registration state
- number of members in guest party
- number of nights expected to be stayed

5. PAST GUEST FILE

The purpose of this file is to archive relevant information on past guests who have stayed at the hotel. In addition to personal information on the guest, the file also stores information related to all previous stays by the guest. The file is read each time a guest checks into the hotel, and is updated each time a guest checks out. There is a record for each guest who has stayed at the hotel. Each record consists of the following fields:

- guest name
    - guest last name
    - guest first name
- guest address
    - guest street number
    - guest apartment number
    - guest city
    - guest state
    - guest zip code
    - guest country
- guest phone number
- guest company name
- guest auto tag
    - guest tag number
    - guest auto registration state
- stay # 1
    - check-in date
        - day of check-in
        - month of check-in
        - year of check-in
        - hour of check-in
        - minute of check-in
    - actual check-out date
        - day of actual check-out
        - month of actual check-out
        - year of actual check-out
        - hour of actual check-out
        - minute of actual check-out
    - room number stayed in
    - number of members in guest party
    - number of nights of expected stay
    - number of nights of actual stay
    - total charges during stay
    - total room charges during stay
    - total amount billed for stay
    - total amount paid for stay
- stay # 2

.

.

- stay # n

6. PAYMENT FILE

The purpose of this file is to capture detailed information on the charges (both room and other charges) made to the guest during his/her stay at the hotel, and the amount paid by the guest for these charges. There is a payment record for each guest. In addition to other information, each payment record contains a listing of each charge made to the guest.

A payment record is first created when the guest checks in and is updated when the guest checks out after paying for his/her hotel expenses. To facilitate management reporting (and to respond to any subsequent queries after check-out by the guest), the payment record corresponding to the guest is retained in the payment file for a period of six months after the guest has checked out. A separate system (not to be considered in the proposed system) deletes the record after the six months have elapsed.

Each payment record contains the following fields:

- guest name
  - guest last name
  - guest first name
- room number
- check-in date
  - day of check-in
  - month of check-in
  - year of check-in
  - hour of check-in
  - minute of check-in
- actual check-out date
  - day of actual check-out
  - month of actual check-out
  - year of actual check-out
  - hour of actual check-out
  - minute of actual check-out
- charge # 1
  - charge date
    - day of charge
    - month of charge
    - year of charge
    - hour of charge
    - minute of charge

charge description
charge amount
charge # 2
.
.
charge # n
total charges during stay
total room charges during stay
total amount paid for stay
payment type (i.e., cash, check or credit card)

REFERENCES

Adelson, B., & Soloway, E. (1985). The role of domain experience in software design. IEEE Transactions on Software Engineering. SE-11, 1351-1360.

Alavi, M. (1984). An assessment of the prototyping approach to information systems development. Communications of the ACM. 27, 556-563.

Alavi, M. (1993). Making CASE an organizational reality. Strategies and new capabilities needed. Information Systems Management. 10(2), 15-20.

Amundsen, B., & Christoffersen, B. (1987). Can today's design tools support an integrated design method? Tietotekniikka-87.

Andrews, W. C. (1983). Prototyping information systems. Journal of Systems Management. 34(9), 16-18.

Arthur, L. J. (1985). Measuring programmer productivity and software quality. New York: John Wiley & Sons.

Baram, G., & Steinberg, G. (1989). Selection criteria for analysis and design CASE tools. Software Engineering Notes. 14(6), 73-80.

Baram, G., Steinberg, G., & Nosek, J. (1990). Evaluation of ease of use of CASE tool by first time users. In B. Whitten & J. Gilbert (Eds.), Proceedings of the Annual Meeting of the American Institute for Decision Sciences (pp. 934-936). Cincinnati, OH: American Institute for Decision Sciences.

Basili, V. R., & Reiter, R. W., Jr. (1981). A controlled experiment quantitatively comparing software development approaches. IEEE Transactions on Software Engineering. SE-7, 299-320.

Basili, V. R., Selby, R. W., & Hutchens, D. H. (1986). Experimentation in software engineering. IEEE Transactions on Software Engineering. SE-12, 733-743.

Batra, D., Hoffer, J. A., & Bostrom, R. P. (1990). Comparing representations with relational and EER models. Communications of the ACM. 33, 126-139.

Beck, L. L., & Perkins, T. E. (1983). A survey of software engineering practice: Tools, methods, and results. IEEE Transactions on Software Engineering. SE-9, 541-561.

Beizer, B. (1990). Software testing techniques (2nd ed.). New York: Van Nostrand Reinhold.

Benander, B. A. (1990). Program design techniques: Programmer preference and relationship to program correctness and program development effort. The Journal of Computer Information Systems. 32(2), 66-71.

Benington, H. D. (1983). Production of large computer programs. Annals of the History of Computing. 5, 350-361. (Reprinted from Proceedings of the ONR Symposium on Advanced Programming Methods for Digital Computers, 1956, pp. 15-27.)

Boehm, B. W. (1976). Software engineering. IEEE Transactions on Computers. C-25, 225-240.

Boehm, B. W. (1981). Software engineering economics. Englewood Cliffs, NJ: Prentice Hall.

Boehm, B. W. (1986). A spiral model of software development and enhancement. Software Engineering Notes. 11(4), 14-24.

Boehm, B. W., Gray, T. E., & Seewaldt, T. (1984). Prototyping vs. specifying: A multi-project experiment. In W. E. Howden & J. -C. Rault (Eds.), Proceedings of the 7th International Conference on Software Engineering (pp. 473-485). Los Angeles: IEEE Computer Society Press.

Boehm, B. W., McClearn, R., & Unfrig, D. (1975). Some experiences with automated aids to the design of large-scale reliable software. IEEE Transactions on Software Engineering. 1, 125-133.

Bordoloi, B., Courtney, L., & Paranusiwaean, R. (1992). A framework for evaluating CASE tools [Summary]. In R. T. Sumichrast (Ed.), Proceedings of the Annual Meeting of the American Institute for Decision Sciences (p. 882). Cincinnati, OH: American Institute for Decision Sciences.

Bostrom, B. T. (1988). Information systems development supporting methodologies with computerized tools. Proceedings of the Polish-Scandinavian Seminar on ISDM.

Brooks, F. P. (1987). No silver bullet: Essence and accidents of software engineering. Computer. 20(4), 10-19.

Brooks, R. E. (1980). Studying programmer behavior experimentally: The problems of proper methodology. Communications of the ACM. 23, 207-213.

Brooks, W. D. (1981). Software technology payoff: Some statistical evidence. The Journal of Systems and Software. 2, 3-9.

Burkhard, D. L. (1989). Implementing CASE tools. Journal of Systems Management. 40(5), 20-25.

Cadre Technologies. (1988). Teamwork/PCSA user's guide. Providence, RI: Author.

Card, D. N., Church, V. E., & Agresti, W. W. (1986). An empirical study of software design practices. IEEE Transactions on Software Engineering. SE-12, 264-270.

Card, D. N., McGarry, F. E., & Page, G. T. (1987). Evaluating software engineering technologies. IEEE Transactions on Software Engineering. SE-13, 845-851.

Carey, J. M., & McLeod, R. (1988). Use of system development methodology and tools. Journal of Systems Management. 39(3), 30-35.

Carmel, E., & Becker, S. (1993). A process model for software package development. Manuscript submitted for publication.

Carroll, J. M., & Carrithers, C. (1984). Training wheels in a user interface. Communications of the ACM. 27, 800-806.

Carroll, J. M., & McKendree, J. (1987). Interface design issues for advice-giving expert systems. Communications of the ACM. 30, 14-31.

Cerveny, R. P., Garrity, E. J., & Sanders, G. L. (1986). The application of prototyping to systems development: A rationale and model. Journal of Management Information Systems. 3(2), 52-62.

Chapin, N. (1979). Some structured analysis techniques. Data Base. 10(1), 16-23.

Chou, D. C., Kelley, G., & Landram, F. (1992). Managing software quality assurance in CASE environment. Manuscript submitted for publication.

Colter, M. A. (1982). Evolution of the structured methodologies. In J. D. Couger, M. A. Colter, & R. W. Knapp (Eds.), Advanced system development/feasibility techniques (pp. 73-96). New York: John Wiley & Sons.

Colter, M. A. (1984). A comparative examination of systems analysis techniques. MIS Quarterly. 8, 51-66.

Conger, S. A. (1994). The new software engineering. Belmont, CA: Wadsworth.

Couger, J. D. (1973). Evolution of business system analysis techniques. Computing Surveys. 5, 167-198.

Couger, J. D. (1982). Fourth generation development techniques for computer-based systems. In J. D. Couger, M. A. Colter, & R. W. Knapp (Eds.), Advanced system development/feasibility techniques (pp. 71-72). New York: John Wiley & Sons.

Crockett, H. D., Hall, G. R., & Wheeler, M. E. (1992). Conceptual application of total quality management (TQM) to the systems development lifecycle (SDLC). In R. T. Sumichrast (Ed.), Proceedings of the Annual Meeting of the American Institute for Decision Sciences (pp. 982-984). Cincinnati, OH: American Institute for Decision Sciences.

Crosslin, R. L., Bergin, T. J., & Stott, J. W. (1993). Critical factors influencing the future of computer-aided software engineering. In T. J. Bergin (Ed.), Computer-aided software engineering: Issues and trends for the 1990s and beyond (pp. 616-637). Harrisburg, PA: Idea Group.

Crow, G. B. (1990). BriefCASE -- The collegiate systems development tool. Cincinnati, OH: South-Western.

Davis, G. B. (1982). Strategies for information requirements determination. IBM Systems Journal. 21(1), 4-30.

Davis, G. B. (1987). Evaluation of computer aided software engineering packages. Information Technology -- Journal of the Singapore Computer Society. 1(4), 51-55.

Day, D. L. (1993). Precis of behavioral and perceptual responses to constraint management in computer-mediated design activities. Electronic Journal of Communication [On-Line], 3(2). Available e-mail: comserve@rpitsvm.bitnet Message: send day v3n293

DBMS. (July 1991). CASE tool roundup. DBMS. pp. 62-69.

DeGrace, P., & Stahl, L. H. (1990). Wicked problems, righteous solutions. A catalogue of modern software engineering paradigms. Englewood Cliffs, NJ: Yourdon Press.

DeMarco, T. (1982). Controlling software projects. Englewood Cliffs, NJ: Yourdon Press.

Dennis, A. R., George, J. F., Jessup, L. M., Nunamaker, J. F., Jr., & Vogel, D. R. (1988). Information technology to support electronic meetings. MIS Quarterly. 12, 591-624.

DeSanctis, G., D'Onofrio, M. J., Sambamurthy, V., & Poole, M. S. (1989). Comprehensiveness and restrictiveness in group decision heuristics: Effects of computer support in consensus decision making. In J. I. DeGross, J. C. Henderson, & B. R. Konsynski (Eds.), Proceedings of the Tenth International Conference on Information Systems (pp. 131-140). Baltimore, MD: Association for Computing Machinery.

Everest, G. C., & Alanis, M. (1992). Assessing user experience with CASE tools: An exploratory analysis. In J. F. Nunamaker, Jr. (Ed.), Proceedings of the 25th Hawaii International Conference on System Sciences (pp. 343-352). Los Alamitos, CA: IEEE Computer Society Press.

Falkner, R. (1991). Quality-driven software. Computerworld. 25(17), pp. 93, 96.

Fenton, N. E. (1991). Software metrics. A rigorous approach. London, Chapman & Hall.

Freeman, R. J. (1993). Measuring the effects of CASE. In T. J. Bergin (Ed.), Computer-aided software engineering: Issues and trends for the 1990s and beyond (pp. 542-551). Harrisburg, PA: Idea Group.

Frolick, M. N., Wilkes, R. B., & Rainer, R. K. (1993). Is CASE living up to its promises? Laboratory experiments comparing manual and automated design approaches. Journal of Computer Information Systems. 33(4), 72-76.

Gane, C. (1990). Computer-aided software engineering: The methodologies, the products, and the future. Englewood Cliffs, NJ: Prentice Hall.

Gibson, M. L. (1989). The CASE philosophy. Byte. 13(4), pp. 209-218.

Ginzberg, M. J. (1981). Early diagnosis of MIS implementation failure. Management Science. 27, 459-478.

Glass, M. C., Hughes, J. G., Johnston, W., & McChesney, I. (1989). Critical analysis of tools for computer-aided software engineering. Information and Software Technology. 31, 486-496.

Golden, J. R., Mueller, J. R., & Anselm, B. (1981). Software cost estimating: Craft or witchcraft. Data Base. 12(3), 12-14.

Granger, M. J., & Pick, R. A. (1991). Computer-aided software engineering's impact on the software development process: An experiment. In J. F. Nunamaker, Jr. (Ed.), Proceedings of the 24th Hawaii International Conference on System Sciences (pp. 28-35). Los Alamitos, CA: IEEE Computer Society Press.

Guimaraes, T. (1985). A study of application program development techniques. Communications of the ACM. 28, 494-499.

Haase, V., & Koch, G. (1982). Developing the connection between user and code. Computer. 15(5), 10-11.

Hannafin, M. J. (1984). Guidelines for using locus of instructional control in the design of computer-assisted instruction. Journal of Instructional Development. 7(3), 6-10.

Heiat, A., & Heiat, N. (1992). The effect on student learning of integrating CASE tools in MIS curricula. Interface: The Computer Education Quarterly. 14(1), 43-45.

Henderson, J. C., & Cooprider, J. G. (1990). Dimensions of I/S planning and design aids: A functional model of CASE technology. Information Systems Research. 1, 227-254.

Henken, P. (1988). Improving structured analysis with video. Journal of Systems Management. 39(5), 17-23.

Hutchins, E. L., Hollan, J. D., & Norman, D. A. (1986). Direct manipulation interfaces. In D. A. Norman & S. W. Draper (Eds.), User centered system design: New perspectives on human-computer interaction. Hillsdale, NJ: Lawrence Erlbaum Associates.

International Business Machines. (1990). Euphoria hotel room management system version 2.0 requirements/external design document (enhanced JAD design exercise solution).

Intersolv. (1989). Excelerator version 1.9 application guide. Rockville, MD: Author.

Jankowski, D. J., & Norman, R. J. (1992). Computer-aided software engineering (CASE) technology in the information systems curriculum: Current practice. The Journal of Computer Information Systems. 32(3), 6-14.

Jones, C. (1986). Programming productivity. New York: McGraw-Hill.

Jones, C. (1991). Applied software measurement. New York: McGraw Hill.

Kapur, G. (1986, December 1). Productivity tools betray promises of MIS nirvana. Computerworld. 20, pp. 61-78.

Kemerer, C. F. (1989). An agenda for research in the managerial evaluation of computer-aided software engineering (CASE) tool impacts. In B. Shriver (Ed.), Proceedings of the 22nd Hawaii International Conference on System Sciences (pp. 219-228). Los Alamitos, CA: IEEE Computer Society Press.

Kemerer, C. F. (1992). How the learning curve affects CASE tool adoption. IEEE Software. 9(3), 23-28.

Kendall, R. C. (1977, July 25). Management perspectives on programs, programming, and productivity. Computerworld. 11, p. 21.

Khailany, A., Sanchez, P., & Lee, L. (1985). On software maintenance costs. In B. Hartman & J. Ringuest (Eds.), Proceedings of the Annual Meeting of the American Institute for Decision Sciences (pp. 321-324). Cincinnati, OH: American Institute for Decision Sciences.

Kievit, K., & Martin, M. (1989). Systems analysis tools -- Who's using them? Journal of Systems Management. 40(7), 26-30.

Ledgard, H. F. (1987). Professional software: Vol. 1. Software engineering concepts. Reading, MA: Addison-Wesley.

Lempp, P., & Lauber, R. (1989) What productivity increases to expect from a CASE environment: Results of a user survey. In E. J. Chikofsky (Ed.), Computer-aided software engineering (CASE) (pp. 105-111). Washington, DC: IEEE Computer Society Press. (Reprinted from Productivity: Progress, Prospects, and Payoff, June 1988, pp. 13-19).

Lindholm, E. (1992). A world of CASE tools. Datamation. 38(5), pp. 75-81.

Linos, P. K. (1992). ToolCASE: A repository of computer-aided software engineering tools. Software Engineering Notes. 17(2), 74-78.

Long, L. E., & Long, N. (1990). Computers (2nd ed.). Englewood Cliffs, NJ: Prentice Hall.

Loy. P. (1993). The method won't save you (but it can help). Software Engineering Notes. 18(1), 30-34.

MacDonell, S. G. (1993). CASE and 4GL product users' participation in software engineering research. Software Engineering Notes. 18(3), A42-A43.

Mahmood, M. A. (1987). System development methods -- A comparative investigation. MIS Quarterly. 11, 293-311.

Mann, J. E. (1992). Empirical research on the adoption and use of system development methodologies: Literature review and extrapolation to future research. In R. T. Sumichrast (Ed.), Proceedings of the Annual Meeting of the American Institute for Decision Sciences (pp. 878-880). Cincinnati, OH: American Institute for Decision Sciences.

Martin, J., & McClure, C. L. (1988). Structured techniques: The basis for CASE. Englewood Cliffs, NJ: Prentice Hall.

McCall, J.A., Richards, P.K., & Walters, G.F. (1977). Factors in software quality: Vol. 1. Concept and definitions of software quality. (Contract No. F030602-76-C-0417). Sunnyvale, CA: General Electric Company.

McClure, C. (1989a). CASE is software automation. Englewood Cliffs, NJ: Prentice Hall.

McClure, C. (1989b). The CASE experience. Byte. 14(4), pp. 235-246.

McDermid, D. C. (1990). Software engineering for information systems. Oxford, UK: Blankwell Scientific Publishing.

McGaughey, R. E., & Gibson, M. L. (1990). CASE and information system quality (Working Paper No. AUCOB00007). Auburn, AL: Auburn University, College of Business.

McKeen, J. D. (1983). Successful development strategies for business information systems. MIS Quarterly. 7(3), 47-65.

McMenamin, S. M., & Palmer, J. F. (1984). Essential systems analysis. Englewood Cliffs, NJ: Yourdon Press.

Moher, T., & Schneider, G. M. (1982). Methodology and experimental research in software engineering. International Journal of Man-Machine Studies. 16, 65-87.

Nash, K. S. (1992). Tempered hopes best route to CASE. Computerworld. 26(41), pp. 1, 20.

Necco, C. R., Gordon, C. L., & Tsai, N. W. (1987). Systems analysis and design: Current practices. MIS Quarterly. 11, 461-475.

Necco, C. R., Tsai, N. W., & Holgeson, K. W. (1989). Current usage of CASE software. Journal of Systems Management. 40(5), 6-11.

Norman, R. J., and Nunamaker, J. F., Jr. (1989). CASE productivity perceptions of software engineering professionals. Communications of the ACM. 32, 1102-1108.

Nunamaker, J. F., Jr., Dennis, A. R., Valacich, J. S., Vogel, D. R., & George, J. F. (1993). Group support systems research: Experience from the lab and field. In L. M. Jessup & J. S. Valacich (Eds.), Group support systems (pp. 125-145). New York: MacMillan.

Omar, M. H. (1992). Seeing is believing: Demos enhance the learning of new MIS topics and tools. Interface: The Computer Education Quarterly. 14(1), 24-26.

Palvia, P., & Nosek, J. T. (1990). An empirical evaluation of system development methodologies. Information Resources Management Journal. 3(3), 23-32.

Page-Jones, M. (1988). The practical guide to structured systems design. Englewood Cliffs, NJ: Yourdon Press.

Page-Jones, M. (1992). The CASE manifesto. CASE Outlook. 6(1), pp. 33-42.

Pallatto, J. (1989, January 23). Survey uncovers truth of real life CASE software. PC Week, pp. 39, 46.

Pressman, R. S. (1992). Software engineering: A practitioner's approach. New York: McGraw-Hill.

Purvis, R. L., & Sambamurthy, V. (1992). A comparative investigation of system design methodologies [Summary]. In R. T. Sumichrast (Ed.), Proceedings of the Annual Meeting of the American Institute for Decision Sciences (p. 864). Cincinnati, OH: American Institute for Decision Sciences.

Putnam, L. H. (1978). A general empirical solution to the macro software sizing and estimating problem. IEEE Transactions on Software Engineering. SE-4, 345-361.

Rob, P., & Coronel, C. (1993). Database systems. Design, implementation, and management. Belmont, CA: Wadsworth.

Roberts, T. L. (1980). Evaluation of computer text editors (Doctoral dissertation, Stanford University). Dissertation Abstracts International. 40, 5338B.

Robillard, P. (1989). A project-based software course: The myth of the "real world". In N. E. Gibbs (Ed.), Proceedings of the 1989 SEI Conference on Software Engineering Education (pp. 297-308). New York: Springer-Verlag.

Rowe, J. M. (1993). Can enforced standardization affect CASE usage? Journal of Systems Management. 44(3), 29-33.

Rozman, I., Györkös, J., & Rizman, K. (1992). Understandability of the software engineering method as an important factor for selecting a CASE tool. Software Engineering Notes. 17(3), 43-46.

Rubin, H. A. (1983). Macro-estimation of software development parameters: The ESTIMACS system. SOFTFAIR -- Software Development: Tools, Techniques, and Alternatives (pp. 109-118). Los Alamitos, CA: IEEE Computer Society Press.

Runyan, L. (1989). Escaping technology's tar pit. Datamation. 35(7), p. 26.

Sakthivel, S. (1991). Information systems development methodologies: A comparative analysis. In S. Melnyk (Ed.), Proceedings of the Annual Meeting of the American Institute for Decision Sciences (pp. 845-847). Cincinnati, OH: American Institute for Decision Sciences.

Schmidt, A. (1992). Working with Excelerator version 1.9. Englewood Cliffs, NJ: Prentice Hall.

Shafer, L. I., & Shafer, D. F. (1993) Establishing a CASE toolbox. 15 steps to selecting CASE tools. Information Systems Management. 10(1), 15-23.

Shultz, S. (1989). Du Pont's I/S team builds a good case for CASE tools. Chief Information Officer Journal. 1(4), pp. 26-28.

Silver, M. S. (1988a). On the restrictiveness of decision support systems. In R. M. Lee, A. M. McCosh, & P. Migliarese (Eds.), Organizational decision support systems (pp. 259-270). New York: Elsevier Science Publishers.

Silver, M. S. (1988b). User perceptions of decision support system restrictiveness: An experiment. Journal of Management Information Systems. 5(1), 51-65.

Silver, M. S. (1990). Decision support systems: directed and nondirected change. Information Systems Research. 1, 47-70.

Silver, M. S. (1991a). Decisional guidance for computer-based decision support. MIS Quarterly. 15, 105-122.

Silver, M. S. (1991b). Systems that support decision makers: Description and analysis. New York: John Wiley & Sons.

Slusky, L. (1989). Academic training in SDLC/R methodology with Excelerator. Proceedings of the 18th Annual Western Decision Sciences Institute.

Stamps, D. (1987). CASE: Cranking our productivity. Datamation. 33(13), pp. 55-58.

Statland, N. (1989). Payoffs down the pike: A CASE study. Datamation. 35(7), pp. 32-33, 52.

Subramanian, G. H., & Gershon, M. (1991). The selection of computer-aided software engineering tools: A multi-criteria decision making approach. Decision Sciences. 22, 1109-1123.

Sumner, M. (1993). Factors influencing the adoption of CASE. In T. J. Bergin (Ed.), Computer-aided software engineering: Issues and trends for the 1990s and beyond (pp. 130-155). Harrisburg, PA: Idea Group.

Sumner, M., & Sitek, J. (1986). Are structured methods for systems analysis and design being used? Journal of Systems Management. 37(6), 18-23.

Teichroew, D., & Hershey, E. A., III. (1977). PSL/PSA: A computer-aided technique for structured documentation and analysis of information processing systems. IEEE Transactions on Software Engineering. SE-3, 41-48.

Teng, J., & Sethi, V. (1990). A comparison of information requirements analysis methods: An experimental study. Data Base. 21(1), 27-39.

Teresko, J. (1990, April 2). What MIS should be telling you about CASE. Industry Week, pp. 82-85.

Troy, D. A., & Zweben, S. H. (1981). Measuring the quality of structured designs. The Journal of Systems and Software. 2, 113-120.

Vessey, I., Jarvenpaa, S. L., & Tractinsky, N. (1992). Evaluation of vendor products: CASE tools as methodology companions. Communications of the ACM. 35(4), 90-105.

Visible Systems. (1989). The CASE primer. Waltham, MA: Author.

Walston, C. E., & Felix, C. P. (1977). A method of programming measurement and estimation. IBM Systems Journal. 16(1), 54-73.

Wenig, R. (1991). Introduction to C.A.S.E. technology using Visible Analyst Workbench. New York: MacMillan.

Whitten, J. L., Bentley, L. D., & Barlow, V. M. (1989). Projects and cases for use with systems analysis and design methods. Homewood, IL: Irwin.

Windsor, J. C. (1986). Are automated tools changing systems analysis and design? Journal of Systems Management. 37(11), 28-32.

Wrigley, C. D., & Dexter, A. S. (1987). Software development estimation models: A review and critique. In H. Barki (Ed.), Proceedings of the Administrative Sciences Association of Canada: MIS Division. (pp. 125-138). Toronto, Ontario: Administrative Sciences Association of Canada.

Wrigley, C. D., & Dexter, A. S. (1988). A model for estimating information system requirements size: Preliminary findings. In J. DeGross & M. Olson (Eds.), Proceedings of the 9th International Conference on Information Systems. (pp. 245-255). Baltimore, MD: Association for Computing Machinery.

Wu, S. Y., & Wu, M. S. (1994). Systems analysis and design. Minneapolis/St. Paul, MN: West.

Wynekoop, J. L., & Conger, S. A. (1991). A review of computer aided software engineering research methods. In H. Nissen, H. Klein, & R. Hirscheim (Eds.), Information Systems Research: Contemporary Approaches & Emergent Traditions. (pp. 301-325). Amsterdam: North-Holland.

Yellen, R. E. (1990). Systems analysts performance using CASE versus manual methods. In J. F. Nunamaker, Jr. (Ed.), Proceedings of the 23rd Hawaii International Conference on System Sciences (pp. 497-500). Los Alamitos, CA: IEEE Computer Society Press.

Yourdon, E. (1986). What ever happened to structured analysis? Datamation. 32(11), pp. 133-138.

Yourdon, E. (1989). Modern structured analysis. Englewood Cliffs, NJ: Yourdon Press.

Yourdon, E. (1992). Decline & fall of the American programmer. Englewood Cliffs, NJ: Yourdon Press.

Yourdon, E., & Constantine, L. L. (1979) Structured design: Fundamentals of a discipline of computer program and systems design. Englewood Cliffs, NJ: Yourdon Press.

Zucconi, L. (1989). Selecting a CASE tool. Software Engineering Notes. 14(2), 42-44.

Zultner, R. (1988). The Deming approach to software quality engineering. Quality Progress. 21(11), 58-64.