1992

Common-cause analysis in human-software interaction: system design, error control mechanism, and prevention

Peom Park
Iowa State University

Common-cause analysis in human-software interaction: System design, error control mechanism, and prevention

Park, Peom, Ph.D.

Iowa State University, 1992

Common-cause analysis in human-software interaction: system design, error control mechanism, and prevention

by

Peom Park

A Dissertation Submitted to the
Graduate Faculty in Partial Fulfillment of the
Requirements for the Degree of
DOCTOR OF PHILOSOPHY

Department: Industrial and Manufacturing Systems Engineering
Major: Industrial Engineering

Approved:

Signature was redacted for privacy.

In Charge of Major Work

Signature was redacted for privacy.

For the Major Department

Signature was redacted for privacy.

For the Graduate College

Members of the Committee:

Signature was redacted for privacy.

Iowa State University
Ames, Iowa
1992

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ACKNOWLEDGEMENTS

I would like to express my appreciation to many persons who assisted me in this research and in the completion of my Ph.D study. Dr. S. Keith Adams, my major professor, deserves special recognition for his encouragement and guidance throughout my research and study life. Without his assistance, consideration, and patience, this research never would have been accomplished. Dr. Way Kuo who was my former advisor and is one of the committee members, is responsible for providing the stepping stones on which I began my study towards a doctoral degree. His unique and convincing perceptions of the software reliability proved to be a great source of motivation. I truly indebted much to them for these.

I would like to thank my committee members: Dr. Geraldine Montag, Dr. H. T. David, Dr. William J. Kennedy, Dr. Doug W. Jacobson, all of whom provided valid criticisms and comments concerning through research progress to my final dissertation. The special thanks go to Dr. John Jackman, Dr. Jo Min and many colleagues in Industrial and Manufacturing Systems Engineering to advise and to support for the experiment and friendship.

Last, but most importantly, I want to acknowledge the members of my family, parents and brothers, for providing my passion and study. Ockran Park, my wife, was wonderful support to this achievement during all study periods.

CHAPTER 1. INTRODUCTION

Overview

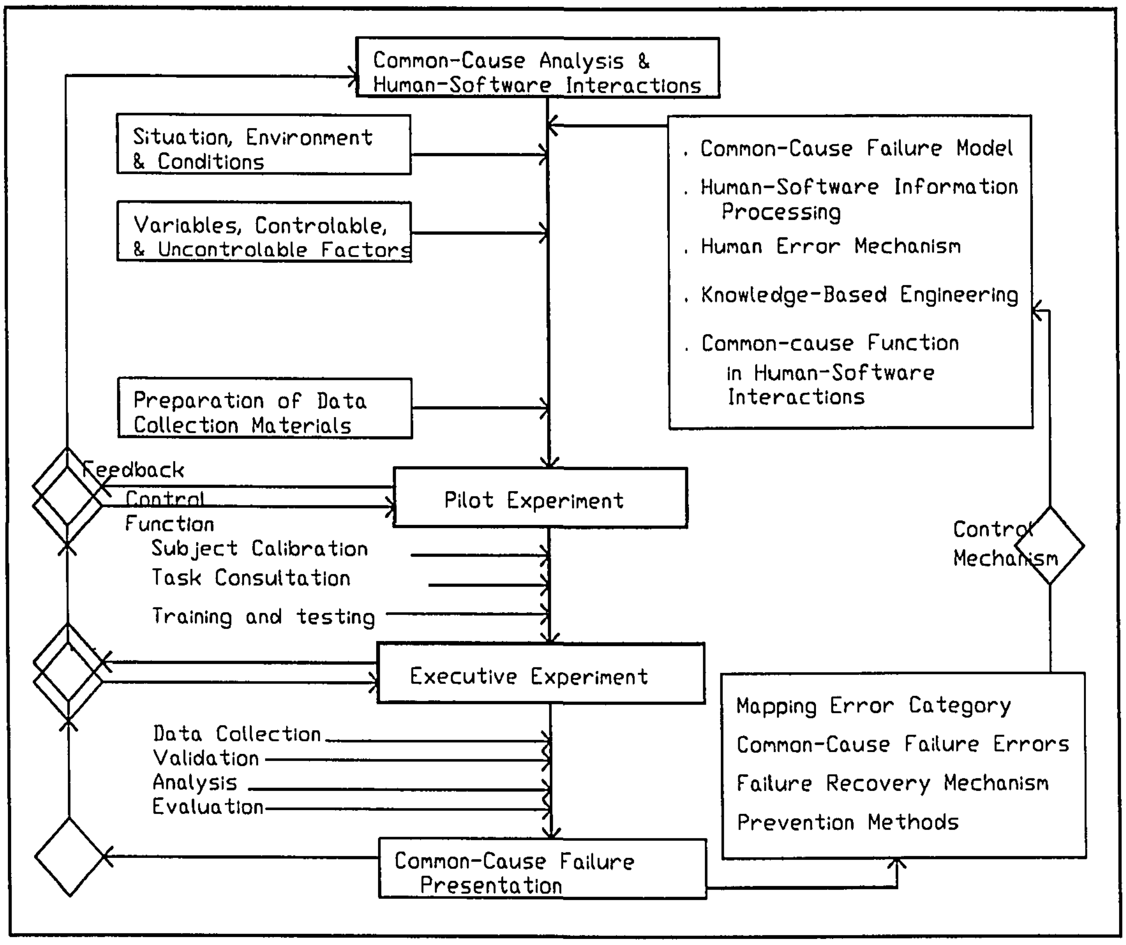
This research introduces the analysis and a new design domain of common-cause human error in human-software interactions. This study is concerned with common-cause human domain errors during software system development. This includes the contents, conditions, and their characteristics in human-software interaction. It also concerns interactions between the human, who is presumed responsible for overseeing the software system, usage of the software system and software development. Also of concern is how to reduce and to prevent human errors in software development systems.

In these days common-cause failure studies [76] [30] [116] in the human-system area have been receiving wide attention especially in the software systems area. This is because the assumption of statistically independent failure of redundant systems is easily violated in real human-software interaction processing systems. Since the software components are not independent of each other in regard to failure behavior, software redundancy does not improve reliability except in multi-version software development. Multi-version software system development is often requested to improve of reliability, especially in ultra-high reliability systems such as nuclear power control, air traffic control, space shuttle missions, and war games. The major common-cause errors found in this research can contribute strongly to internal common-cause failure effects in a multi-version software development project. Human error, the human reliability function, the principle of the common-cause and its effect, and the probabilistic concept of common-cause are reviewed in this work.

There are three main components in human-software interaction; the human as a software engineer, software as an operator, and the hardware system as a software development work station. It is important to analyze the characteristics and the environment of each subsystem. A software development system will be derived to analyze the task of software development. Human-software information processing will be discussed in order to clarify the human behavioral process in human-software interaction.

The common-cause error model includes three analytical reasoning categories and a common-cause function established in terms of human-software information processing systems, human error mechanisms, and cognitive control domains. It is used to characterize the human factors mechanisms behind typical categories of errors considered as occurrences of human-software task mismatches.

An experiment to develop an improved design concept, its procedure, and analysis was conducted to define common-cause errors in the human domain of software development. The major role of this experiment is to find contents, environments, and conditions of common-cause human domain errors, and a design procedure for the analysis of common human behavioral factors. Overall research design scheme is shown in Figure 1.1. In this experiment, each software development by a subject, and the effect of common-cause failure are analyzed to evaluate human-software interaction and to improve software development productivity. Finally, a prevention



Common-Cause Analysis & Human-Software Interactions
Situation, Environment & Conditions
Variables, Controlable, & Uncontrolable Factors
Preparation of Data Collection Materials
. Common-Cause Failure Model
. Human-Software Information Processing
. Human Error Mechanism
. Knowledge-Based Engineering
. Common-cause Function in Human-Software Interactions
Feedback
Control Function
Pilot Experiment
Subject Calibration
Task Consultation
Training and testing
Control Mechanism
Executive Experiment
Data Collection
Validation
Analysis
Evaluation
Mapping Error Category
Common-Cause Failure Errors
Failure Recovery Mechanism
Prevention Methods
Common-Cause Failure Presentation

Figure 1.1: The Research Design Scheme in Human-Software Interaction

method for the common-cause human error control mechanism is introduced with aspects of knowledge-based engineering, fuzzy set application, and intelligent design technique.

Multi-Version Redundant Software Systems and the Common-Cause Effect in Human-Software Interaction

A failure in programming task can occur during any phase of software development. Potential failures can sometimes be found by software engineers as the result of design review, and code proof reading. A software failure is a departure of operation from specified requirements in setting up or modifying a program. The common-cause effect, as a reliability component of the common-cause failure system, serially connected with other system components, in human-software interaction, is affected by internal common-cause human domain errors.

The component structure of AND-OR rules in a k out of n component structure assumes that failures of different components are independent of each other [76]. This means that there can be no failures that result from the same cause [116]. Reliability is often increased in hardware systems by providing redundant components. In software systems, the situation is different. In hardware systems, the causes of failure associated with physically individual but functionally identical units in hardware component systems, are frequently independent. This phenomenon does not occur in software systems because multi-copies of a program are identical not only in function but also in the faults that can cause failures.

However, there is a possible exception in the situation of software development if multi-version software components are developed by different teams. There is some possibility that many faults introduced may be independent of each other with redundant software components developed by separate teams following the same specific requirements. This described by Knight et al. [56] who point out experiments in multi-version programming which seem to indicate that failures in different versions are clearly not completely independent of each other. They do not appear to be all common either. Multi-version programming may well improve the reliability level, but not to the extent totally independent components would. Having totally independent components may be cost effective for critical modules of systems with ultra-high reliability requirements, such as nuclear power plants, air traffic control systems, space shuttle missions, and war games.

The common-cause effect is a system component that is not well recognized. It is serially-connected with the human-software system, operating system, and hardware system. Common-cause failure effect can be defined as the consequences a common-cause failure mode has on the operations, function or status of an item/task. Failure effects are classified as local effect, next higher level and end effect [4]. Here, common-cause failure can be defined as the simultaneous failure of more than one component, or more than one component failing due to a single cause [42]. It is also defined by I. A. Watson [116] as inability of multiple, first-in-line items to perform as required in a defined critical time period, due to a single underlying defect or physical phenomena, such that the end effect is judged to be a loss of one or more systems. Human aspects of internal common-causes are also present in common-cause effect at human-software interactions of multi-version software development. Because this common-cause effect affects the productivity and the development cost of a software project, removal common causes is very important in improving reliability in an ultra-high reliability software project.

Literature Search

Deborah Mitta [75] presented a methodology for quantifying expert system usability which is considered from a designer's prospective. A linear multivariate function for measuring usability is described and procedures for selecting function variables are provided. The usefulness of the usability function as a design tool is investigated. The six variables for expert useability are: user confidence, the user's perception of difficulty, correctness of solution, the number of responses required of users, inability of expert system to provide a solution, rate of help requests.

Thomas A. Thayer et al. [112] presented results of a study of data, principally error data, collected from four software development projects. This study was designed to determine what might be learned about various types of errors in software, the effectiveness of the development and test strategies in preventing and detecting errors, and the reliability of the software itself. This study provided guidelines for data collection and analysis on other projects.

Albert Endres [33] classified error into six groups; machine error, user or operator error, suggestions for improvement, duplicate, documentation error, and program error. Program errors were classified as machine configuration and architecture (10%), dynamic behavior and communication between processes (17%), functions offered (12%), initialization (8%), addressability (7%), reference to names (7%), counting and calculating (8%), and others (16%). It was possible to distinguish causes for errors in 6 categories; technological, organizational, historical, group dynamic, individual, and other.

Edward A. Youngs [121] discussed systematizing the description of errors that programmers make by collecting protocol data from 42 programmers. Eight functionally defined constructions accounted for more than 75 percent of all 1189 errors committed: (1) allocation (16%), (2) assignment (29%), (3) iteration (10%), (4) I/O formatting (6%), (5) other I/O (8%), (6) parameter/subscription list (5%), (7) conditional execution (5%), (8) vertical delimiter (4%).

Jens Rasmussen [90] classified cognitive control domains: skill, rule, and knowledge-based behavior. He also described psychological mechanisms in the area of human-task mismatches.

Modeling and predicting human error was studied by David D. Woods [120]. This research included a limited rationality approach and some directions in error modeling.

James Reason [92] studied a general framework for locating the principal limitations and biases giving rise to the more predictable varieties of human error. Three types of error were identified: skill-based, rule-based, and knowledge-based mistakes.

Common-cause failure in system interaction and statistical theory are discussed in the following papers: *Review of Common-cause Failures* by I. A. Watson [116]; *Rational Belief and the Common-Cause Principle* by Bas C. Van Fraassen [37]; *Causal Forks and Common Causes* by Wesley C. Salmon [102]; *Causal Inference and Causal Explanation* by Clark Glymour [38].

Definition of Problems and Terminology

Problem situation with common-cause effect

The mission of a specific software development project is to set up system components of human-software interaction. Each configuration is composed of a computer work station, a Central Operating Processor (COP) whose computer assigns and controls all work at the local working stations, and a Multi-Version Software (MVS) development load. One approach to software design research using such a system that tends to be expensive, is to install two independent versions of MVS developed by two completely separate software development teams/engineers. The common-cause effect affected by internal common-cause human domain errors is determined using redundant components in this case as in Figure 1.2.

In the given example [76], the system reliability is

$$(0.99)(0.95)\{1-[1-(0.98)R_x^{1-r}]^2\}R_x^r.$$

If there are no common-cause error effect , then $r = 0$ and R_x becomes $0.809(\lambda_x = 0.0662$ failure/cpu hr.). However, the chances are that r is relatively large, that is, similar common errors are made by each team. If $r = 0.5$, then $R_x = 0.922(\lambda_x = 0.0254$ failure/cpu hr.). In the case of $r = 0$, the development cost for the MVS software will be about \$580,000 (\$290,000 for each copy of the software). Similarly, if $r = 0.5$ the software development cost will be about \$1,150,000. An additional \$250,000 will be incurred for the second unit of MVS hardware. The total cost will be \$830,000 or \$1,400,000, depending on the value of r in the Musa's study.

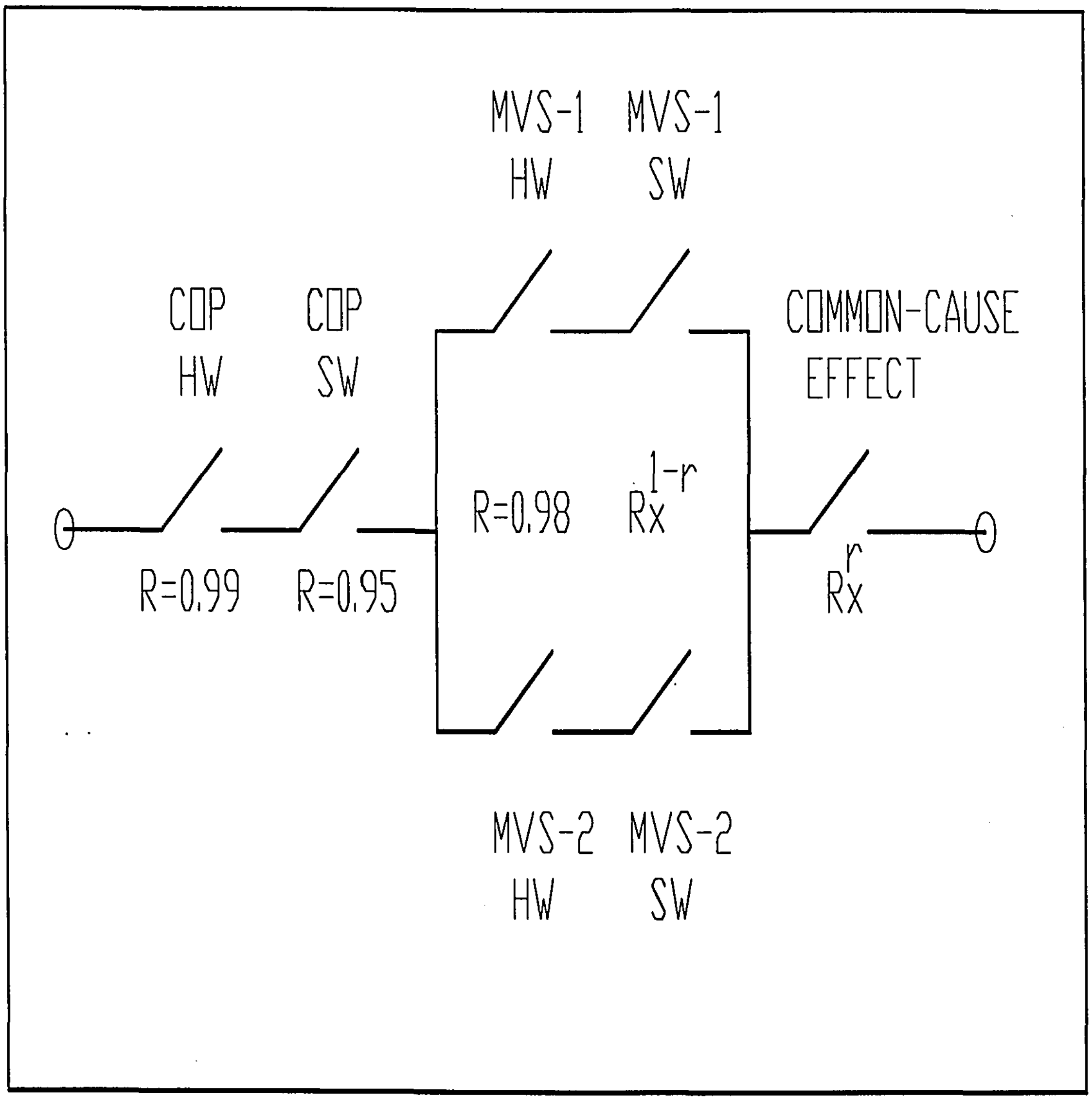


Figure 1.2: Event Diagram of A Multi-Version Redundant Software System in Human-Software Interaction

Scope of the problem

This study deals with the problem of common-cause human domain error in human-software interaction, that is, the major causal factors in common-cause failure effects on the multi-version software development. Questions to be addressed include: how to analyze reasons for common-cause errors, how to design common-cause error control mechanisms, and how to define methods for their prevention. Specially, the following questions need to be addressed:

(1) What are common-cause failures and their internal common-cause human domain errors in human-software interactions?

(2) What are the contents and conditions of human-based errors affected by a common-cause effect in a multi-version redundant software development system ?

(3) How can internal common-causes by human-based errors be reduced and prevented in the software development?

(4) How can software engineers be aided by a human-based common-cause error control mechanism in the design of high reliability software system?

(5) How should interactive systems between the human, who is presumed responsible for overseeing software systems, and software development efforts which are human oriented be designed for high reliability efficiently?

Some of the requirements and motivations for common-cause error analysis that can be extended to human-software error control and prevention in software development. They are:

(1) There must be a coherent methodology and processing mechanism to control and guide a software project to successful completion.

(2) A new design-based knowledge for multi-version redundant software development

is needed to train experienced software engineers.

(3) A large portion of experienced software engineers do not have a sensitivity to human-software interactive error mechanisms and methods of preventing errors.

(4) Software development has resulted in many incorrect human programming behaviors which have led to low quality software and excessive costs.

(5) There are unique aspects of software development without direct hardware/operation analogs; thus not all the training learned about past hardware/operation development is applicable to the software development task.

Common-cause failure, failure, errors, and reliability

Common-cause failure is defined as "the simultaneous failure of more than one component [42]." Here, a failure of a component or subsystem is said to be a propagating failure when the failure changes the programming conditions, environments or requirements in such a way as to cause the failure of other components of software development. It is said to be a common-cause failure if more than one component fails due to a single cause (usually assumed to be external to the programming conditions of the human-software information processing system). Such common causes may be from the human domain attributable to psychological behavior or to physiological capacity, or to external disruption by man-made or natural events.

In hardware reliability theory where multiple components fail due to a single cause, a common-cause failure is said to have occurred. This can easily be extended to software components. A straightforward method to incorporate these common-cause failures is given in Dhillon [30]. Let r be defined as the fraction of component failures that are common-cause. Each component failure intensity λ is the sum of an independent failure intensity $(1-r)\lambda$ and a common-cause failure intensity $r\lambda$.

A **failure** of a human-software interaction system occurs when that system does not perform its service/execution in the manner specified, whether because it is unable to perform the service/execution at all, or because the results and the external state[1] are not in accordance with the specifications. **Failure** is "a departure of the external results of program operation from program requirements on a run [76]." A departure is the occurrence of a discrepancy between the desired output result stated in the requirement specifications for the specific run and the actual output result. Therefore, it represents a defect in a transformation. The output result is the set of values of output variables with a program execution. A discrepancy is defined as "the difference between the actual value of an output variable with an execution and the value expected by the requirement specifications [76]." The time of a failure is the time at which the discrepancy first occurs. The type of failure is defined as the conjunction of both run type or input state and discrepancy. The allocation of causes to human or components in human-software interaction systems is a purely pragmatic question regarding the stop rule applied for analysis after the fact.

Fault is defined as a defective, missing, or extra instruction or set of related instructions that is the cause of one or more actual or potential failure types. There cannot be multiple faults causing a failure. The entire set of defective instructions that is causing the failure is considered to be the fault. The requirement that the instructions be related is specified so that the count of the number of faults cannot be

[1]The *external state* of a system is the result of a conceptual abstraction function applied to its internal state. The *internal state* of a system is the aggregation of the external states of all its components [73].

changed arbitrarily by a regrouping of instructions. The characteristics of a fault are [76]: (1) it is the cause of deviation from a standard; (2) it is found on the causal path by tracing backwards from this effect; (3) it is accepted as a familiar and therefore reasonable explanation; (4) a cure is known.

Human error consists of any significant deviation from a previously established, required or expected standard of human performance, that results in unwanted or undesirable time delay, difficulty, problem, trouble, incident, malfunction, or failure [85]. In another way, it is described as the failure to carry out a specified task (or the performance of a forbidden action, or improper performance of a task) that could lead to disruption of scheduled operations or result in damage to property and component. Errors can arise from many causes, but most of them can be grouped in one of four categories [76]: communication, knowledge, incomplete analysis, or transcription. In real situations where arguments of precisely what is or is not a human error are less important than what can be done to prevent them, the operational definition may be restricted to those errors (a) which occur within a particular set of activities, (b) which are of some significance or criticality to the primary operation under consideration, (c) involve a human action of commission or omission, and (d) about which there is some feasible course of action which can be taken to correct or prevent their reoccurrence [22].

Human reliability is defined as "the probability of accomplishing a job or task successfully by humans at any required stage in a system operation within a specified minimum time limit [28]." Here, human-software reliability can be defined as the probability of successful performance with human-software task ability and reliable systems at any required stage in an operation of the human-software interaction system within a specified duration of time.

Human-Software Information Processing system is defined by a network system of human and software components capable of accepting information, processing it according to a plan and a control, and producing the desired results or goals.

CHAPTER 2. SYSTEM INTERACTIONS AND INFORMATION PROCESSING IN SOFTWARE DEVELOPMENT

Human-Software System Interactions

Human-software interaction represented by Figure 2.1 consists of three elemental components: human, software, and hardware. It is the software that gives the computer its individuality; the computer then works as a link to connect the system components. Considerable effort has been expended to establish theories and practices for attaining hardware reliability. One reason is that hardware is more general than the software.

Software tends to be specific to each system, although sometimes efforts are made to utilize standard program packages that have been verified in other applications. In contrast to hardware, only small samples of similar software are available and it is hard to verify inferences concerning reliability.

In identifying the scope of human-software interaction, it is well to keep in mind the meaning of an interaction, a link or a connection among the three components. The interaction can be addressed on three sides of the diagram in Figure 2.1.

(1) The human side of the interaction includes:

Personnel availability: manning levels and work levels

Personnel capability: skills and skill levels

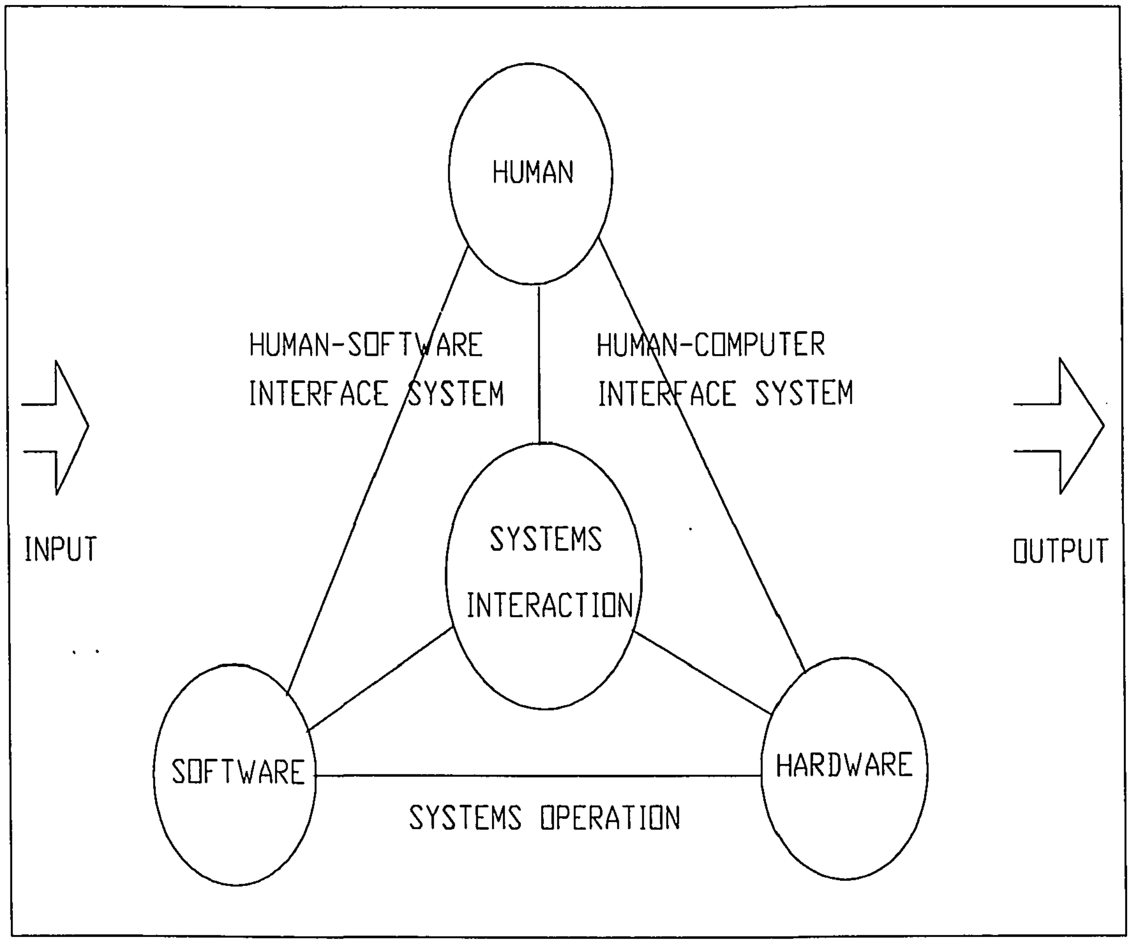


Figure 2.1: Three Components of Human-Software Interaction

Personnel performance: completion of assigned tasks

Personnel productivity: quantity produced per unit time

Personnel safety.

(2) The software side of the interaction includes:

Specification of requirements

Design: software design, process design

User-friendliness: user oriented, easy use, objective oriented

Interface with hardware: hardware capacity with software size

Software productivity: efficiency, effectiveness.

(3) The hardware side of the interaction includes:

Information displays: the information displayed and the display format

Display characteristics: symbol size, shape, color, density, etc.

Data organization: architecture producing hierarchy of data specificity

Dialogues: command modes, error messages, prompts, alerts, queries, etc.

Procedures: task sequences, decisions, and decision rules

Data entry devices: for data entry, manipulation, and designation

Documentation: hard copy manuals and aids.

The software engineer and the programming task

Will the software engineer solve a given problem? How-well will he or she be able to perform that task, and how will this system be well-adapted to achieve the intended goal? The answers depend on the following critical factors: the nature of the task, the availability of the needed expertise, and the ability to analyze and to perform the task in such a way that a computer program, using limited levels of reasoning, can work out what has to be done.

The conditions will tend to rule out certain applications from the start; the software engineer should be able to perform the task, know how he or she performs the task, be able to explain how to perform the task, have the time to explain how to perform the task, and be motivated to cooperate in the enterprise.

Even if the above conditions are met, there may be features of the task that limit the extent that skills can be mechanized. This occurs, for instance, if the task involves complex sensory-motor skills beyond the scope of current technology in robotics, computer vision, and high technological software operations; also if the task involves common-cause reasoning or arbitrary amounts of everyday knowledge.

To be effective would also require an enormous amount of knowledge about the world: knowledge of objects and their properties, software engineers (or teams) and their motivations, physical and psychological causality. The fact is that only the most rudimentary notions about how to impact this kind of common-cause, knowledge to computer software work exist. So any task that is not sufficiently self-contained to be encapsulated in a finite set of particular facts and general rules is definitely beyond the state of the art.

Typical failure mechanisms among human programmers, the operating system, and hardware systems

There are four typical failure mechanisms in human-software interaction, poor-quality fabrication, human-software design, overload of the component, and wear-out among the three components: human, software system, and hardware system [104]. The following examples are illustrative:

(1) Poor-quality fabrication:

(1a) Human:

Reload control button pushed in error during operation.

Wrong disk mounted on drive by operator.

Radar control switch put into track position by operator rather than scan position.

(1b) Software:

Typographical error in entering an instruction which eludes compiler checks.

Wrong version of a subroutine included by mistake.

Program has small incompatibility with operating system or hardware.

(1c) Hardware:

Bad solder joints.

Defective component installed.

Mechanical misalignment.

(2) Human-software design:

(2a) Human:

The human is required to enter data in response to a system request. One of the requests is ambiguous and wrong data are entered.

Assume that following a system crash, the operator must reenter certain key data. If the key sequence is illogical many errors will occur.

The operator follows an incorrect explanation in the operator's manual and inadvertently clears all memory.

(2b) Software:

When the operator returns from subroutine A to the main program he or she fails to clear all registers as they should.

The THEN ELSE branches are mistakenly interchanged in an IF statement.

The series expansion used for a special mathematical function does not converge for certain values.

(2c) Hardware:

Component with too low a rating is specified.

Metal parts are exposed to a corrosive atmosphere.

When an address is loaded from the front panel of a minicomputer, it erroneously clears the accumulator.

(3) Overload of a component:

(3a) Human:

An air traffic controller cannot handle more than 50 targets without overloading his or her vigilance capacity and making many errors.

The operator forgets the right sequence of commands on occasion because there are too many steps.

The human cannot react fast enough to enter control commands in an emergency situation.

(3b) Software:

A timesharing system designed to handle 24 terminals performs poorly when over 20 terminals are connected and its crash rate rises.

The input module of a text-editing system cannot keep up with a very fast typist.

An air traffic control system has a capacity of 100 planes. When more than 100 planes are entered, targets on the screen disappear without warning.

(3c). Hardware:

A capacitor with a maximum rating of 50 V is used in a circuit where 100-V transients occasionally occur.

An unexpected heavy load on a gear train breaks off some gear teeth.

The hardware cannot keep up with an input of 300 band, even though specifications call for operation at this rate.

(4) Wear-out:

(4a) Human:

Possibly errors due to cumulative fatigue.

(4b). Software:

No analogous effect.

(4c). Hardware:

A mechanical clutch begins to slip after 5000 hours of operation.

The insulation on certain wires cracks after 10 years of survival, causing short circuits.

High humidity eventually causes leakage failure of certain types of integrated-circuit packages.

Principles of human-software interaction

There are six intellectual principles of human-software interaction processing.

(1) The only way to adequately design a system is to build it. Brooks [20] describes

his *throwaway one* rule, which is a statement on the limits on human intelligence, or a more academic version of Murphy's Law [1]. It implies, however, that software development is an inherently iterative process.

(2) Software development is a logical rather than a physical system element. Software is developed or engineered; it is not manufactured like hardware, even though the software factory concept recommends the use of automated tools, such as *Fourth Generation Techniques* (4GT), for software development.

(3) Individual programmers have enormous differences in productivity. Although an imperfect measure, lines of program produced per day is an obvious means of evaluating output. By this measure the variations in output have been observed within the same programming shop among programmers of similar background. Clearly, all of these differences cannot reflect learned behavior.

(4) Software development costs are concentrated in the engineering of human-software interaction. Reliability and error-content measures are the key factors for software quality control and the software cost function. A much more costly class of errors consists of those which are detected in the field. Boehm [13] studied the relative cost of removing software errors, by phase of development as given in Figure 2.2[2].

(5) The human brain has intrinsic limits on the complexity of human-software interaction problems with which it can efficiently deal. Halstead introduced concept of software redundancy and program length now called by his name [40]. Tests have shown that there is a desired level of redundancy for optimal absorption of infor-

[1]In general Murphy's Law states that *If anything can possibly go wrong with a design, test, or experiment - it will* [7].

[2]The data sources were IBM-SDD, TRW, GTE, and BELL LABS programs. The upper and lower curves represent a 95 percent confidence interval [13].

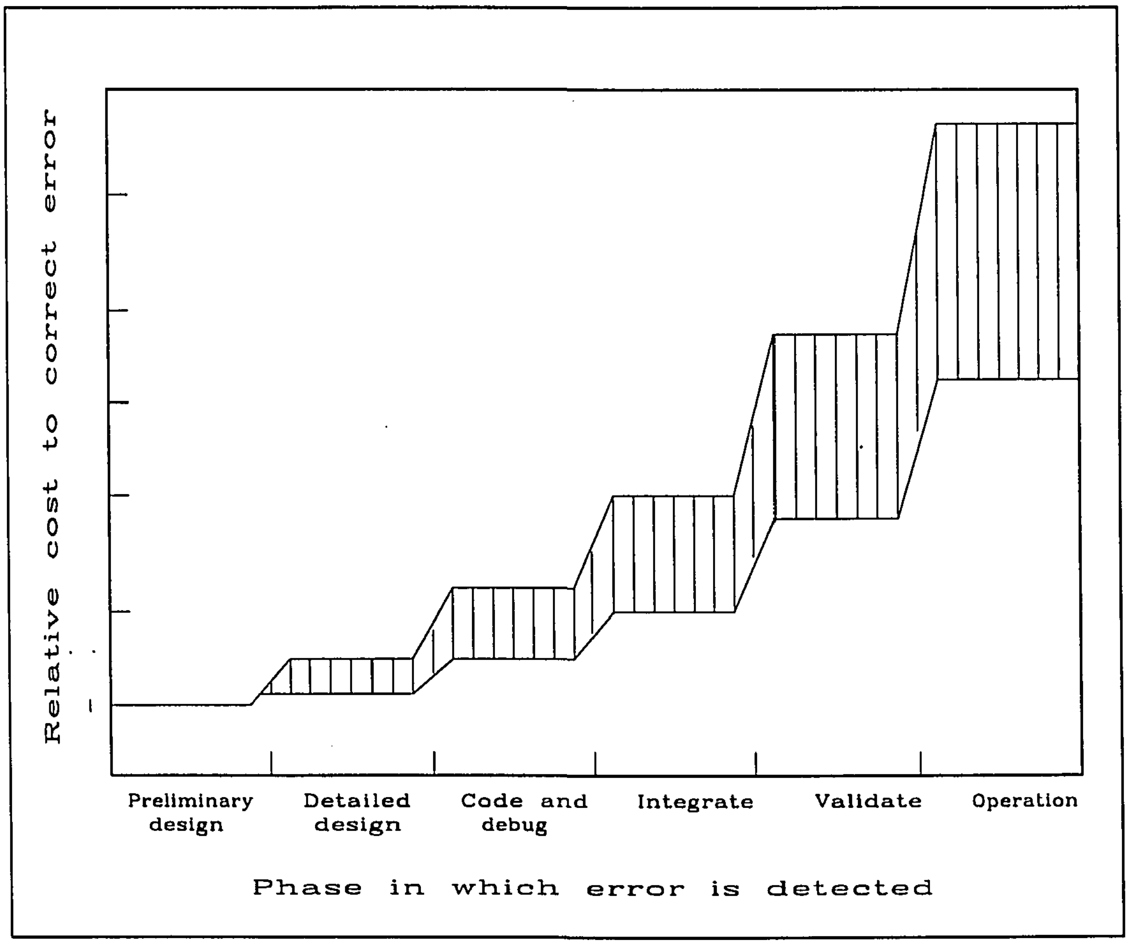


Figure 2.2: The Relative Cost of Fixing Errors Versus Phase of Development [13]

mation; being either too concise or too verbose inhibits understanding. However, Halstead length[3] is a concrete way of measuring the interactions of complexity and length. He mentioned that humans have an upper limit to the Halstead length they can handle. To deal with a problem requires reducing it to models of acceptable Halstead length. This can be done by simply ignoring details or by subdividing a problem into pieces, although this latter raises new, possibly very large costs of coordination.

(6) Human behaviors are very different between group and individual. Brooks [20] observed that six programmers for one month are not the same as one programmer for six months. It should not surprise economists familiar with the transaction costs of coordinating efforts. Subsequent result from Brooks have established extreme trade-offs between complexity and elapsed time (an 8% increase in complexity requires a doubling of staff, for example, according to one accepted rule).

Guidelines of modeling in human-software interaction systems

There are parallels between interaction modeling and software engineering produced by Raduchel [88].

(1) Every good model is a properly specified model.

(2) Tools are vital to good modeling.

(3) The tasks of implementing the estimation and solution techniques are no longer central to modeling.

(4) Modeling is usually a dynamic process with ongoing maintenance and management required.

[3]Halstead length equation: $N = \eta_1 log_2^{\eta 1} + \eta_2 log_2^{\eta 2}$ [40]

(5) The only way to adequately specify a model is to build it.

(6) Individual model-builders have tremendous variations in productivity.

(7) Few, if any, individuals can comprehend all the detail of a large model.

(8) Group modeling efforts are very different from individual efforts.

Human-Software Information Processing Systems

Human-Software Information Processing (HSIP) represented by Figure 2.3 is a part of experimental psychology concerned with the basic research problem of how information flows and is transformed within the human organism and software systems. The Information Processing System in human-software interaction is defined by a network system of human and software components capable of accepting information, processing it according to a plan and a control, and producing the desired results or goals. Human information processing in engineering psychology or human factors and computer data processing in software engineering are technologies that try to improve the performance of human-software interaction systems in which humans and softwares are each parts.

Human sensory capacity and stimulus response compatibility

A model developed by Welford [119] to identify skill mechanisms is presented as Figure 2.4. The lines show information flow and the boxes denote identifiable processing function.

The concept of Stimulus-Response Compatibility [36] is used to explain phenomena in reaction-time experiments, where spatial mappings of stimuli onto responses are varied. The spatial geometry of stimulus and response arrays can be manipulated

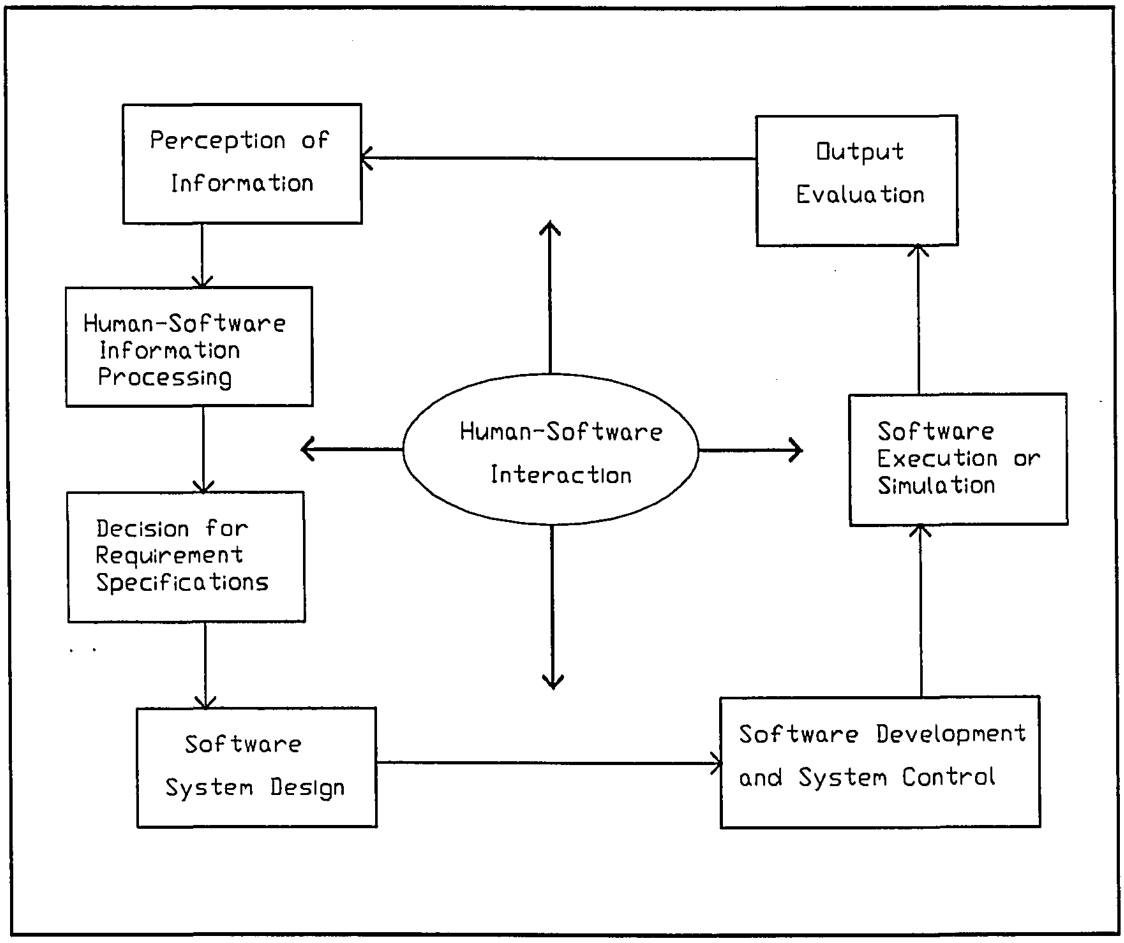

Perception of Information
Output Evaluation
Human-Software Information Processing
Human-Software Interaction
Software Execution or Simulation
Decision for Requirement Specifications
Software System Design
Software Development and System Control

Figure 2.3: Representation of Human-Software Information Processing Systems

in ways limited only by the software designer's ingenuity. The results of Fitts et al. [36] can be indicated by noting that reaction time is fastest and error rates lowest when there is a direct correspondence between the geometry of stimulus and response arrays.

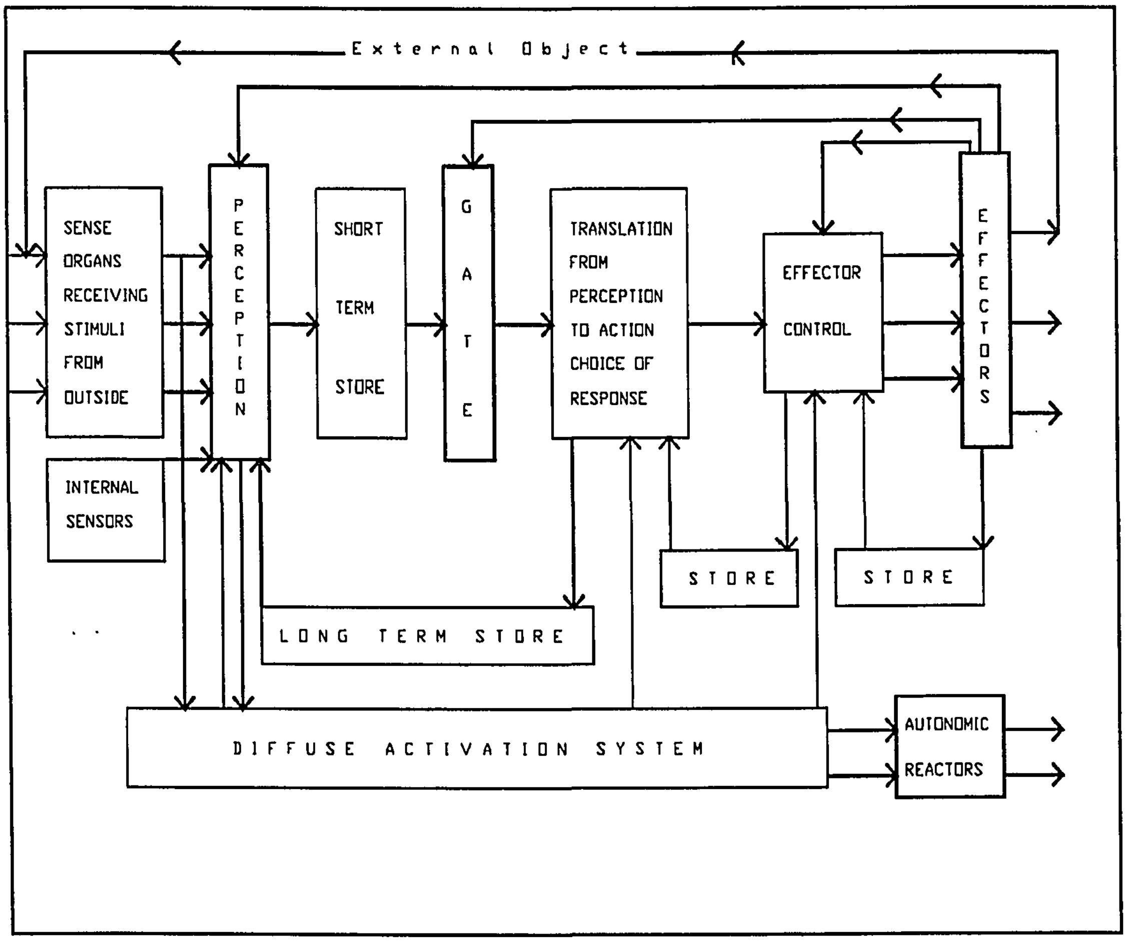


Figure 2.4: Hypothetical Block Diagram of the Human Sensory-Motor System [119]

The most common method of measuring S-R compatibility is to take a vote; that is, several arrangements are portrayed, and people are asked to select the mapping they find most desirable, called *population stereotype.* A slightly more convincing way of measuring S-R compatibility is to conduct an experiment; the fastest and most accurate mapping is obviously the most compatible.

Investigations of S-R compatibility making use of sophisticated mathematical treatments also focus upon properties of a hypothetical translation stage. Harm et al. [41] plotted a Latency Operating Characteristic (LOC) - a function relating Reaction Time (RT) to a measure of accuracy - for compatible and incompatible two-choice reactions. The result was that compatibility affected the *noise* level inside the translation stage, so that incompatible mapping caused elevations in correct and error RT. However, Duncan [32] disagreed that individual S-R bonds were most important and instead argued that systems of rules governed response generation under various S-R mappings. With S-R compatibility it is easy to apply this study to improving human productivity in any system that required operators to map the stimulus information given in displays to a set of controls.

The information channel of limited capacity

The Information Channel of Limited Capacity (ICLC) is the most important and the most influential theoretical construct in human-software information processing. The ICLC has developed within the areas of experimental psychology of reaction time, attention, and memory. Broadbent [17] clearly states the applied origins of his model: "In situations arising from technology our attention is compelled to the major variables in human behavior, and we cannot ignore them in favor of some artificial distinction. The researcher, remote from immediate practical pressures, may indeed be free to study major variables in which at this instant society does not seem to be interested; but he should not use this freedom in order to study minor variables, until there are no major ones within reach of our techniques. The necessity of some relevance to real life is a worthwhile intellectual discipline."

Human-software information processing theory tells us that not only is a particular stimulus important in human-software interactions but also is the set of stimuli from which that particular stimulus was selected; that is, behavior is controlled by events that did not occur on some particular occasion but might have occurred. It can be determined whether the amount of information generated at the source is the amount that reaches the receiver. One of the major question in determining the information channel capacity is the amount of information per unit time that can be transmitted through the human. Even though this amount varies by the coding schemes used in specific tasks, an important theoretical fact is that some fixed upper bound exists.

Figure 2.5 is the most lasting and influential component of Broadbent's [19] model which represents the human operator in terms of the flow of information. A selective filter mechanism protects the information channel of limited capacity. This filter selects which elements of the buzzing confusion of the world available to our senses gain entry. A model of attention with such a gatekeeper is now called an *early-selection* model of attention, and there has been much dispute about how and where sensitivity is imposed, even though there is general accord that selectivity is an important characteristic of human-software information processing. However, this filter model has weak points, such as weakening low priority information [114], and making contact with memory.

Another kind of research supports the limited-channel information capacity, based on the *Psychological Refractory Period* (PRP) effect analogous to the refrac-

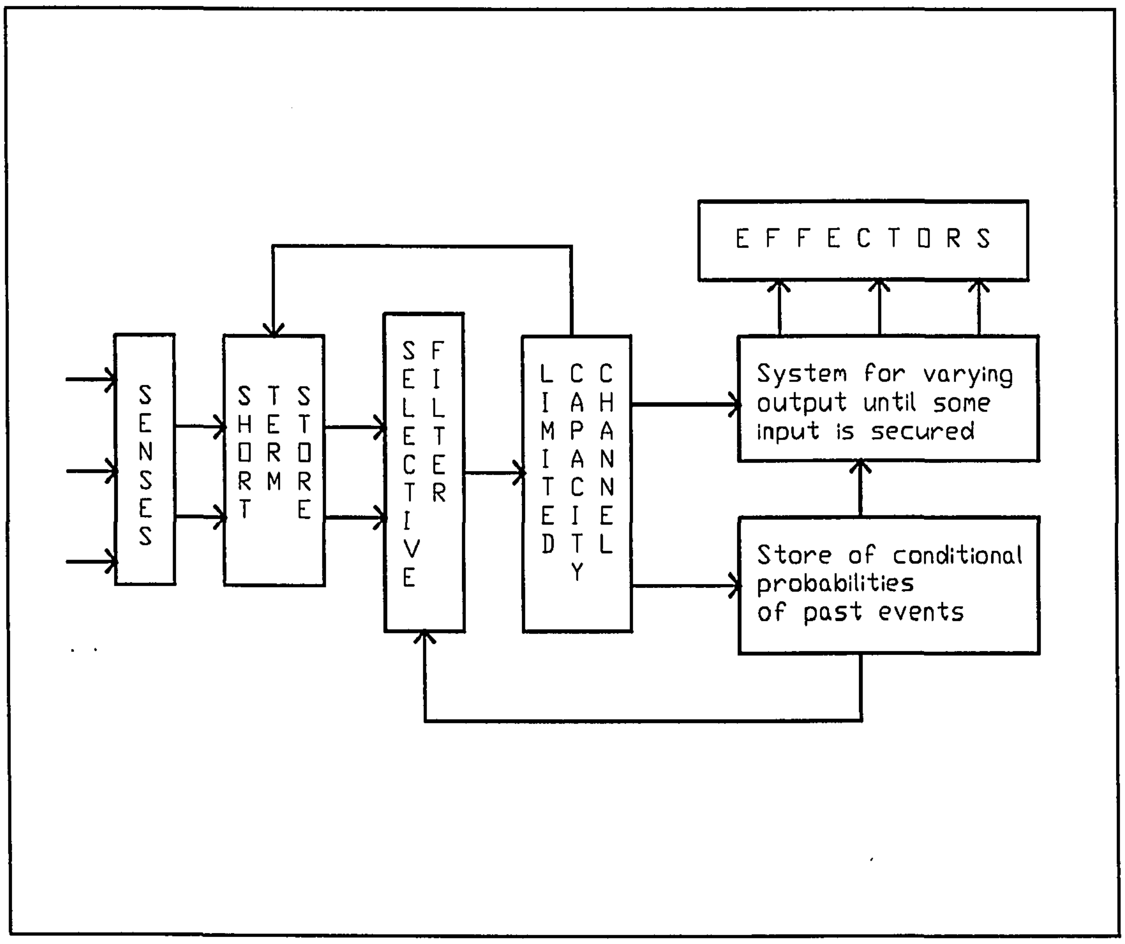

SENSES
SHORT TERM STORE
SELECTIVE FILTER
LIMITED CAPACITY CHANNEL
EFFECTORS
System for varying output until some input is secured
Store of conditional probabilities of past events

Figure 2.5: The Original Limited-Capacity Channel Model [19]

tory period of a single neuron. Broadbent [19] described this effect in terms of the second stimulus queueing up while the channel was busy processing the first stimulus. However, it has been argued that similar reaction time delays also occurred for the first stimulus [44]. Two kinds of information channel of limited capacity are introduced as follows:

(1) Single-channel capacity: Broadbent's [19] single-channel hypothesis was a forerunner of modern information processing theories, especially those which make use of limited central resources. This single-channel, limited-capacity system processes stimuli in a strictly serial manner. Incoming stimuli compete for resources in the sense that they compete for access to the channel in this model, with the capacity of the channel defined in terms of the rate of information transmission. Reaction time(R_t)[4] is a linear function of the $\log_2 P$ with definition of this single-channel capacity by Hick [45] and Hyman [47].

(2) Multi-channel capacity: With multichannel capacity, the appropriate model of the human in software systems is a system with a number of particular purpose processors and stores operating in parallel. Parallel processing is possible when tasks use different processors, but sharing of a particular processor is not possible. Another result is that capacity interference alone is not sufficient to account for the phenomena associated with dual-task performance.

[4]Hick's law: $RT = a + bT(s,r)$, reaction time is a linear function of stimulus and response

Time-shared systems

Time-shared systems involve the simultaneous performance of two separate and independent tasks. The software system operator is required to perform his or her tasks in operating systems together with his or her best ability. Performance on the primary task is required to be constant and capacity as the primary task varies in difficulty is mapped by performance on the secondary task. As the primary task demands increasing capacity, secondary task performance is progressively degraded.

The basic prediction of the limited-channel model is an interaction between task difficulty, and whether the primary task is performed alone or in concert with the secondary task: The drop in performance is greater for the difficult primary task than for the relatively easy primary task. This prediction is equally valid when only dual-task performance is considered, and both primary and secondary tasks have two levels(i.e., easy and difficult Kantowitz et al. [53], Kantowitz, [52]).

Allport et al. [2] reported that the human operator was better represented by several independent channels that operated in parallel, rather than by only a single channel.

Kantowitz et al. [53] examined an intermediate hybrid model (Figure 2.6) that is less parsimonious than the limited capacity single channel but more parsimonious that n independent channels. A hybrid model is one that is neither strictly serial or strictly parallel but contains both kinds of processing in its system architecture. Figure 2.6 represents a hybrid model to explain systems in time-sharing experiments [53]. They combined a motor-tapping task with a digit-naming task.

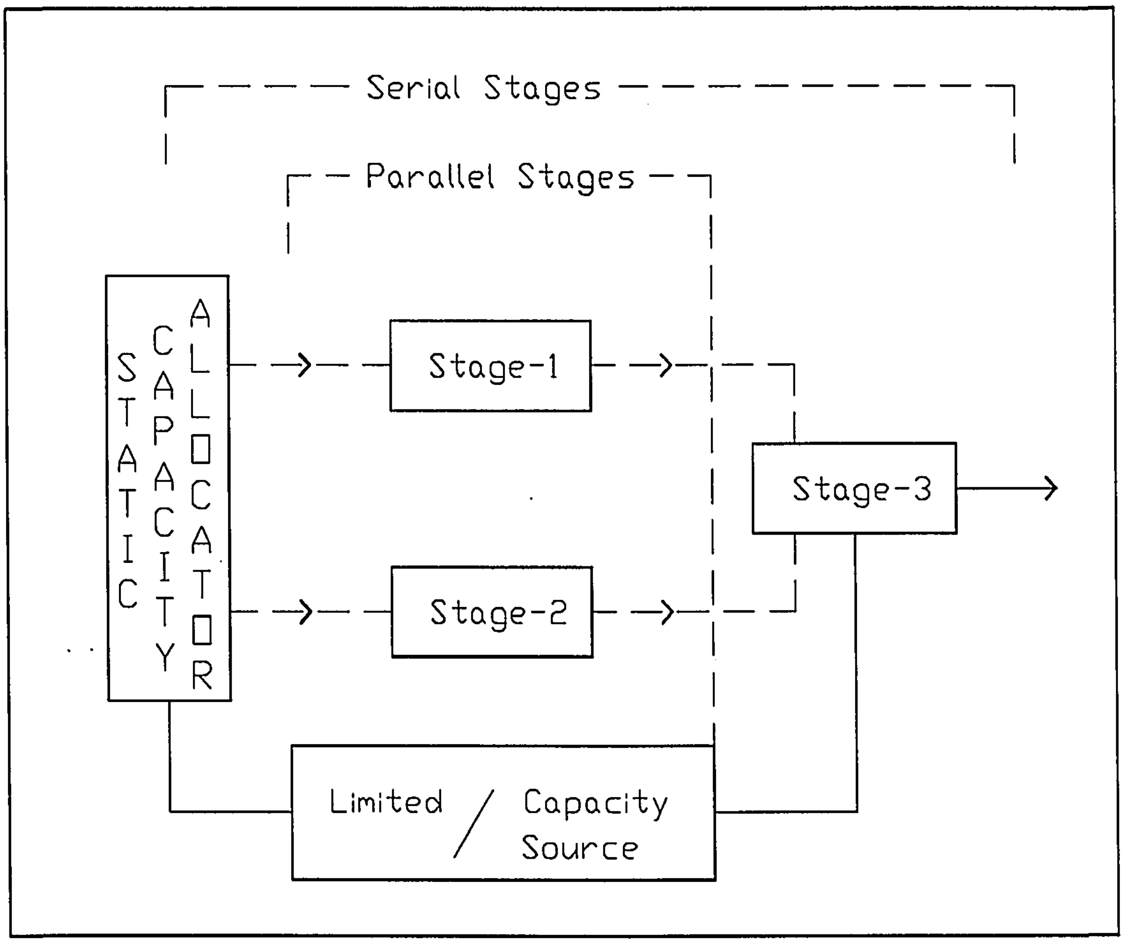


Figure 2.6: The Hybrid Capacity Model [53]

Knowledge-Based System Interaction

It simulates human reasoning about a problem domain, rather than simulating the domain itself. This distinguishes knowledge-based systems from more familiar programs that involve mathematical modeling. This is not to say that the program is a faithful psychological model of the knowledge-based, merely that the focus is upon emulating an knowledge-based problem-solving, that is, performing the relevant tasks as well as, or better than, the expert.

It performs reasoning over representations of human knowledge, in addition to doing numerical calculations or data retrieval. The knowledge in the program is normally expressed in some special purpose language and kept separate from the code that performs the reasoning. These distinct program modules are referred to as the knowledge-base and the inference engine, respectively.

It solves problems by heuristic or approximate methods which, unlike algorithmic solutions, are not guaranteed to succeed. A heuristic is essentially a rule of thumb which encodes a piece of knowledge about how to solve problems in some domain. Such methods are approximate in the sense that they do not require perfect data and the solutions derived by the system may be proposed with varying degrees of certainty.

Knowledge-based interaction systems required by an adaptive human-software interface

There are seven adaptive system interfaces in knowledge-based interaction between the human and software system.

(1) Knowledge of the programmer; that is, expertise with the system;

(2) Knowledge of the human-software interaction; that is, modalities of interaction and dialogue management;

(3) Knowledge of the operation/domain; that is, the ultimate purpose of the problem area and its goals;

(4) Knowledge of the human-software system; that is, the characteristics of the human-software interaction systems; and

(5) Knowledge of the programmer and program designer: A human operator model, combining information about the user's knowledge, capabilities, and preferences, should reflect the content of the operator's knowledge of the human-software system and the operation domain as well as their individual cognitive strengths and limitations. Major issues in building the programmer/designer model: (a) determining what information should be incorporated into the programmer/designer model; (b) determining how this model should be configured. Cognitive psychology issues play a major role in modeling the programmer because there are individual differences among software engineers with knowledge and experience. There are three techniques to construct and modify programmer models: (a) Classifying programmers as novices and update their status to experts as they demonstrate more proficiency; (b) Comparing the programmer's knowledge to a domain expert's knowledge; (c) Characterizing the programmer by a set of stereotypical traits.

(6) Knowledge of human-software interaction: An adaptive human-software interface should provide help that is appropriate to the context as well as to the particular operator. It should be able to track the recent human-software dialogue. This requires some knowledge of how interactions are structured and what information may

be implicit in them. Natural language[5] interfaces are inherently more adaptive in that they do not require learning any artificial command syntax for communicating with human-software interface systems. The following criteria for natural language systems are usable and friendly to novices and experts [60]:

(a) Syntactic coverage;

(b) Task-oriented semantic coverage;

(c) Flexibility in the presence of extra-grammaticality;

(d) Semantic resilience;

(e) User friendliness;

(f) Transportability.

(7) Knowledge of the task/domain: A programmer is trying to accomplish his task. There may be several levels between the immediate task and the overall programming task. If a human-software interface system is to be maximally supportive it must be able to assist the software engineer in achieving programming tasks. The system must be able to infer the information from the human-software interaction. (a) Task modeling: Although many adaptive human-software interface systems use a model of the programmer to gauge the amount and the type of adaptation, there are several systems that are not based upon user models. The adaptation is based upon the human-software system's performance on the task. (b) Task detection and plan inference: An adaptive human-software interface must know what the user wants to accomplish. There are two possible conditions under which plan recognition occurs. First, all possible plans of the programmer are known in the case of limited task domain. Second, all possible plans are not known in the case of any reasonably

[5]Natural language refers to the software engineer's native language.

complex system [80].

Fuzzy set application to knowledge-based human-software interaction

The common-cause error in human-software interaction can be controlled by the fuzzy set theory. Classical control theory provides a good design solution to linear single input, single output system problems. The fuzzy set theory, as a modern control theory, has also proven to be very useful for solving common-cause problems of linear multi-variable system that are of a deterministic or stochastic nature using state space space or frequency response methods in human-software interaction [106]. Common-cause human domain error normally are regulated by human software designer who adjusts the control mechanism. The following problems should be overcome to have an accurate description of the common-cause human domain error control strategy of an software engineer [54].

(1) The control mechanism of a software designer are often erratic, inconsistent or subject to error due to the imprecise nature of human information processes, and hence the programmer's control mechanism is difficult to interpret accurately.

(2) The software engineer frequently responds not only to single measurements, but to complex patterns of measurements and observations of unmeasurable variables, such as consistency and complexity, etc. These observations are then categorized subjectively and used as a basis for control mechanism.

In the approach for fuzzy models of human behavior aspects problem solving, Rouse [97] studied following three basic approaches, such as, the pattern recognition approach, the structured approach, and the rule-based approach.

(1) The pattern recognition approach: This approach has been used as a basis for

modeling the medical diagnosis process of a physician [34]. In this model for human-software interaction, the system designer directly transforms a three dimensional common-cause failure error attributes into membership values for the fuzzy set of possible solutions. The basic assumption is that the human/design has a repertoire of stored patterns that is sufficient for producing acceptable solutions to the common causes that are encountered.

(2) The structured approach: This approach was introduced by Rouse [99] in modeling human decision making in fault diagnosis tasks using fuzzy set theory. It concerns the common cause human error solver as using the structure of the common causes problem to infer membership in the fuzzy set of possible solutions. Thus, given the symptoms of a problem and network of transition relationship diagram, a fuzzy set of possible solutions is defined as those problem elements that have a path to all of the symptoms. The set is fuzzy in which the programmer may not have precise knowledge of the existence or lack of a path from each element to each symptom. The fuzzy fault diagnosis model was used to predict the number of actions that were required before a correct solution could be found. Results from common-cause error analysis were presented for simulated fault diagnosis tasks involving common-cause error control mechanism. It is very difficult for programmer to utilize information about elements which had not failed, and also to model a subject that makes many mistakes.

(3) Rule-based approach: A fuzzy rule-based model of human problem solving was development by Rouse et al [100]. This approach involves the use of rules that evoke actions which lead the software designer towards a human-software interaction solution. The model of Rouse and Hunt was designed to search the problem space in both

a symptomatic and topographic mode. The symptomatic search is based on the state variables in the system, where as the topographic search relies more on the functional structure of the system. Fundamentally the model attempts to choose an appropriate action based on the observed symptoms of the malfunction. If the model fails to recognize a familiar pattern then action is taken based on the functional topography of the malfunctioning interaction system.

Software Development System

There are five procedural phases in software development: requirement specification, software systems design, program coding, software systems validation, and fault-tolerant software systems. As shown in Figure 2.7, there are several factors to be considered in each phase.

Figure 2.8 represents three recovery zones for each software development procedural phase associated with four occasional common-cause failure domains.

Specifications of requirements and tasks

This first phase of software development defines the requirements and specifications for an acceptable solution to the problem. Requirements analysis focuses on the interface between the software and the user who needs to operate it. This task for the software development involves what the program is supposed to do; what real software project problems it is to solve, the inputs and the outputs of the program, and the available human, hardware, and operating software resources. These requirements then need to be translated into a set of explicit specifications for a software project. Requirement partitioning has been defined as the synthesis or grouping of elements of

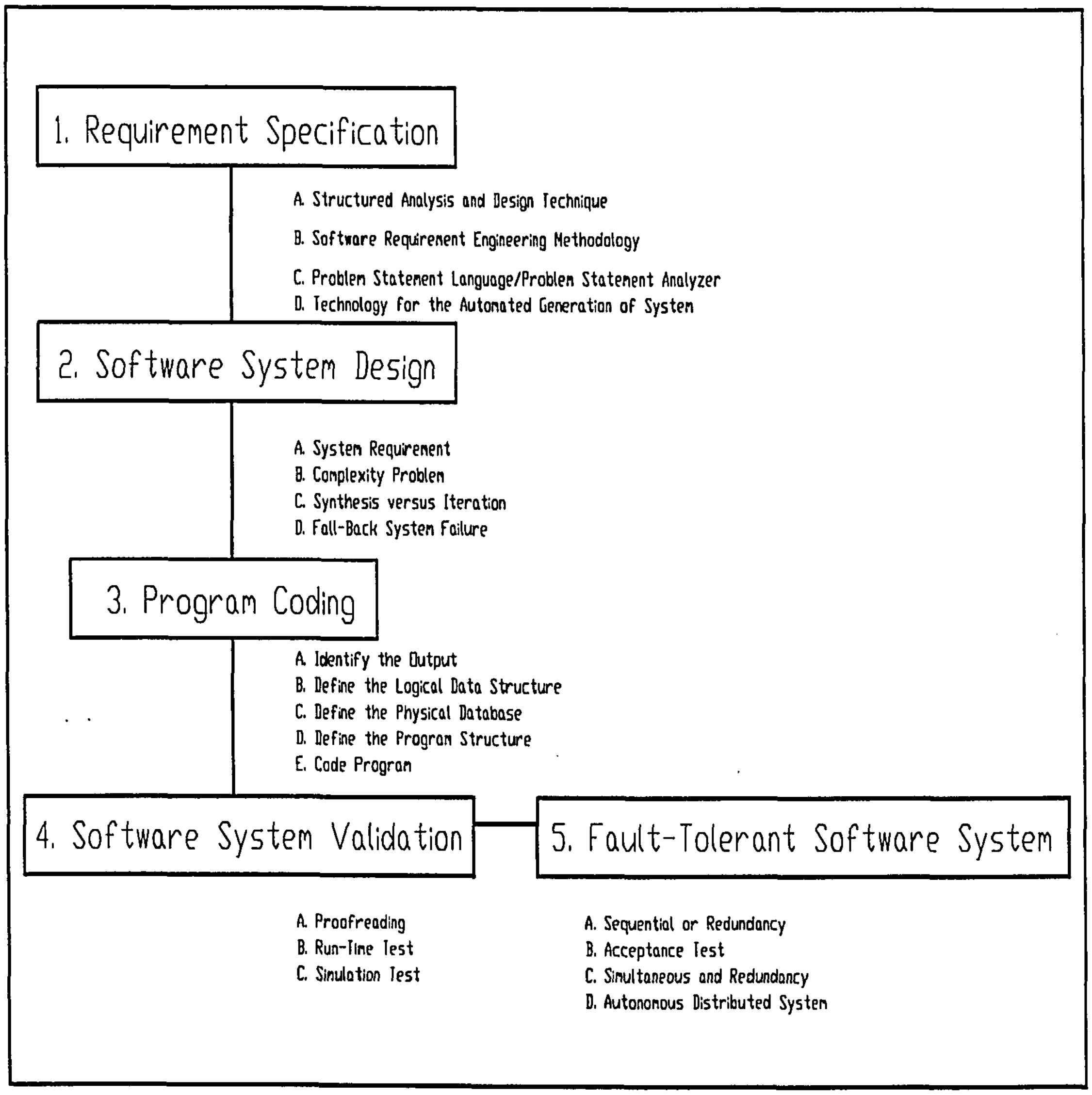

1. Requirement Specification
A. Structured Analysis and Design Technique
B. Software Requirement Engineering Methodology
C. Problem Statement Language/Problem Statement Analyzer
D. Technology for the Automated Generation of System
2. Software System Design
A. System Requirement
B. Complexity Problem
C. Synthesis versus Iteration
D. Fall-Back System Failure
3. Program Coding
A. Identify the Output
B. Define the Logical Data Structure
C. Define the Physical Database
D. Define the Program Structure
E. Code Program
4. Software System Validation
A. Proofreading
B. Run-Time Test
C. Simulation Test
5. Fault-Tolerant Software System
A. Sequential or Redundancy
B. Acceptance Test
C. Simultaneous and Redundancy
D. Autonomous Distributed System

Figure 2.7: Software Development and System Interactions

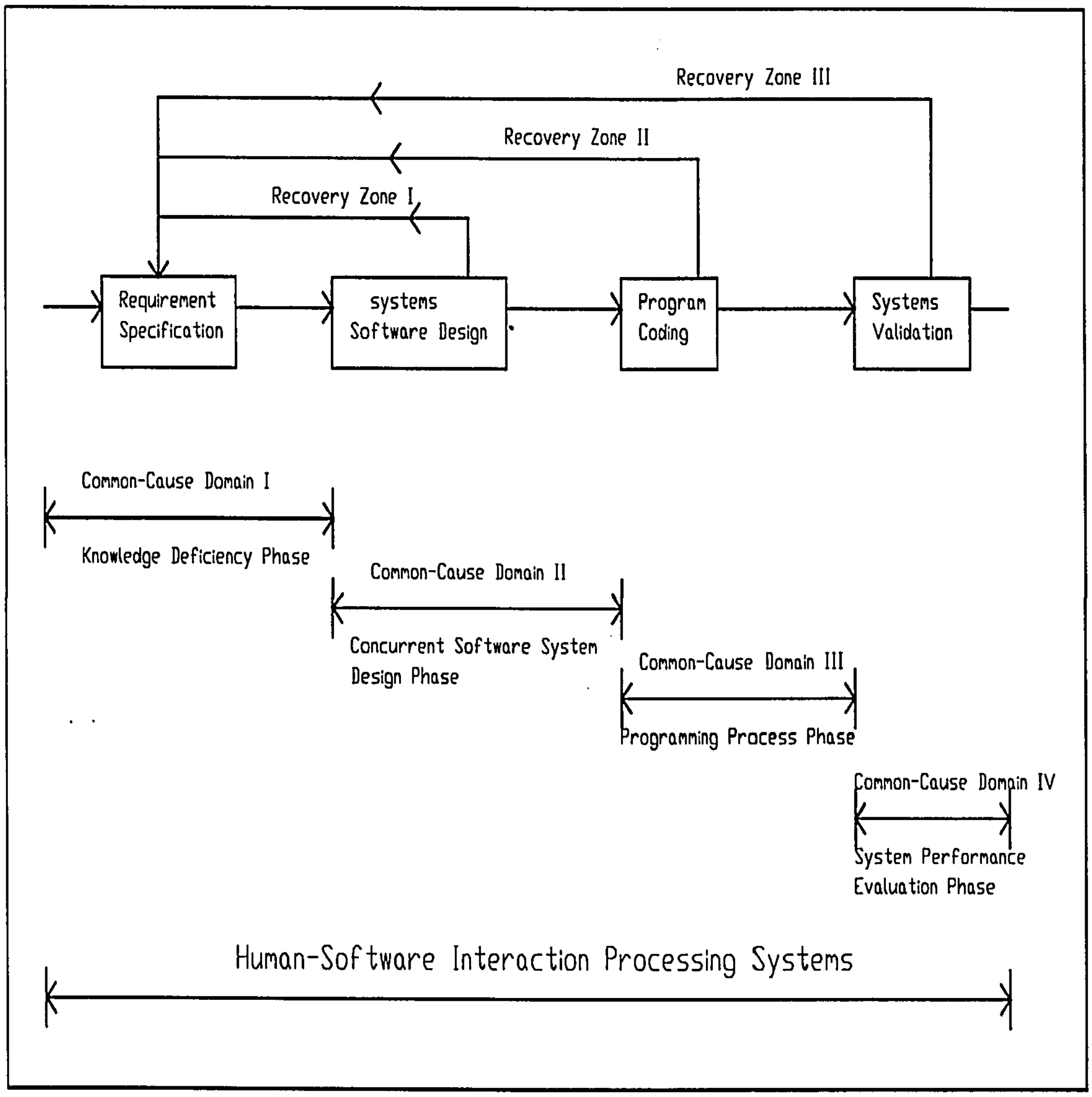

Recovery Zone III
Recovery Zone II
Recovery Zone I
Requirement Specification
systems Software Design
Program Coding
Systems Validation
Common-Cause Domain I
Knowledge Deficiency Phase
Common-Cause Domain II
Concurrent Software System Design Phase
Common-Cause Domain III
Programming Process Phase
Common-Cause Domain IV
System Performance Evaluation Phase
Human-Software Interaction Processing Systems

Figure 2.8: Common-Causes and Recovery Zones

decomposition according to a well-defined criteria into a logical programming space. It is more well related to nonfunctional than to functional requirements.

Defining specification requirements gives both the user and the software engineer a concrete description of the system. Included are the desired operating characteristics of executive program, execution speed, portability, modifiability, size, etc.. Allocation of specifications is the activity of mapping the logical programming space onto physical software resources. Specifications enable test data to be developed early where the performance of the human-software system can be tested objectively because the test data will not be influenced by the implementation.

There are four automatic analysis techniques of specification requirements for software development:

(1) SADT[6]: SADT is a structural analysis and design technique used as a tool for system definition, software requirements analysis, and system/software design. It consists of procedures that allow the analyst to decompose software function; a graphical notation, the SADT *actigram* and *datagram,* that communicates the relationships of information to function within software; and project control guidelines for applying the methodology [101].

(2) SREM[7]: SREM is an automated requirements analysis tool that makes use of a Requirements Statement Language(RSL) to describe "elements, attributes, relationships and structures." These RSL primitives are combined with narrative information to form the detail of a requirements specification [1].

(3) PSL/PSA[8]: This technique provides an analyst with capabilities that include:

[6]SADT:Structured Analysis and Design Technique

[7]SREM:Software Requirement Engineering Methodology

[8]PSL/PSA: Problem Statement Language/Problem Statement Analyzer

(a) description of information systems regardless of application area; (b) creation of a data base containing description for the information system; (c) addition, deletion, and modification of descriptors; and (d) production of formatted documentation and a variety of reports on the specification [111].

(4) TAGS[9]: This is composed of three key components: (a) a specification language called Input/Output Requirements Language(IORL); (b) a set of software tools for requirements analysis and IORL processing; and (c) an underlying TAGS methodology. Unlike SREM and PSL/PSA, the TAGS specification language was designed to accommodate both graphical and textual representations created by the analyst using an interactive tool [105].

Software development system design

The main goal of program design is to produce a cost effective design which satisfies its intended use. It may be assumed that this goal is to produce a design with low residual error content. In certain high-reliability applications this is not sufficient, and various techniques for self-checking and limiting the effect of an error are needed. Design is a very human-directed and highly interactive process in which the analyst uses a mixture of knowledge and intuition to generate initial approaches or configurations. There are four modifying descriptions in software development system design:

(1) Seven factors for system development can be considered in software design:

(a) Safety

(b) Reliability

[9]TAGS: Technology for the Automated Generation of System

(c) Fault Tolerance

(d) Safe shutdown and rapid recovery

(e) Maintainability

(f) Testability

(g) Extendability

(2) There are some important software quality measures needed to codify and reduce by analysis, experiment, or quantitative estimates in a software project. They are: the complexity or the problem, the required algorithm, the processing time, and the data-representation and memory requirements. Estimates of complexity are very useful to the programmer in the early design stage. Since some models focus on the testability of software, these may last to be useful in the design phase.

(3) Synthesis versus iteration: In synthesis, a clear-cut and straightforward algorithm can be used to evolve a design which exactly meets the requirement specifications. On the other hand, the iterative design process can be begun by assuming that an analysis technique exists. Then, one can propose intuitively some hypothetical design and subject it to analysis. A true synthesis procedure is a one-shot, open-loop design. However, because one must always be on guard for human error, the design is checked. An iterative design is generally checked several times.

(4) Fall-back from software failure: The principle of software fall-back is to bypass the malfunctioning task when a problem is detected[43]. This allows continuous operation of remaining tasks without shutting down the entire system. Problem errors occur more often in less important tasks than in those of major importance. When the less important program is bypassed, there is little effect on overall system performance because only some of the functions are lost. There are some considerations for fall-

back from failure:

Manual control and changeover processing

Changeover from Semiautomatic Control

Changeover from manual control

Program coding

Software development actually consists of two complementary but quite different processes: the selection and design of algorithms, and the coding of those algorithms into computer languages.

Coding is the process of translating an algorithm into a form mutually understood by people and computers. However, while nearly all software engineers are also skilled at coding, all coders are not skilled in design. Algorithmic design and selection is a creative process far more akin to creative writing than to indexing, and just as most persons can be taught to index or code, only a few prove to have real talent at creative writing or software development. The following steps are provided for programming coding:

Step (1) Identify the output

Step (2) Define the logical data structure

Step (3) Define the physical database

Step (4) Design the program structure

Step (5) Coding.

System validation of human-software interaction

A software engineering mistake is often made when implementing a verification and validation task that simply duplicates the testing program. Validation should be planned and initiated at the outset of the development program. Even though the actual program testing activity cannot be initiated until end item products become available, the testing of requirements and specifications and the design of software test cases and test tools should proceed concurrently with early phases of the development program.

A flowgraph analyzer is capable of detecting references to variables which are never initialized or never reused after receiving a value; these usually indicate errors. Other methods such as Proofreading, Run-time test, Simulation test can be applied to validating the program. Conway [24] described eight different meanings for a correct program in this phase:

(1) A program contains no syntactic error.

(2) A program contains no compilation errors or failures during program execution.

(3) There exist test data for which the program gives correct answers.

(4) For typical sets of test data, the program gives correct answers.

(5) For difficult sets of test data, the program gives correct answers.

(6) For all possible sets of data which are valid with respect to the problem specification, the program gives correct answers.

(7) For all possible sets of valid test data and all likely conditions of erroneous input, the program gives correct answers.

(8) For all possible input, the program gives correct answers.

Fault-tolerant human-software interaction system

There are two cases for a fault-tolerant system in human-software interaction. One is a failure in a fault-tolerant system which includes design features countered the effects of system faults. It is called system fault-tolerant in accordance with the faults they countered. In interpreting a failure for a fault-tolerant system, one can consider variations from requirements of external and not internal behavior. Therefore, an internal component failure may be counteracted by fault-tolerant features of the system. In other cases, a malfunction of a fault-tolerant feature that affects the program output will represent a system failure.

Systems with their components can be regarded as performing operations in order to provide responses to requests. B. Randell [89] discussed the idealized fault-tolerant component with three categorical groups of existing faults within a system from the viewpoint of a given component:

(1) faults within the component itself,

(2) faults in the sub-components or co-existing components that a component makes use of, and

(3) faulty requests made of the component by its environment, i.e. the enclosing component or the co-existing components with which it is interacting.

Randell's concentration is on three forms of structuring [89]:

(1) idealized fault-tolerant components: provide a means of system structuring which makes it easy to identify those parts of a system that have specific responsibilities for coping with given faults,

(2) recursive structuring scheme: involves using complete systems as the basic idealized fault-tolerant components of a distributed component system whose functionality

matches that of its component systems,

(3) atomic action: provides a means of structuring both forward and backward error recovery in distributed systems.

CHAPTER 3. COMMON CAUSE ERROR AND THE HUMAN RELIABILITY FUNCTION

The human reliability function is concerned with human error, common-cause failure, and common-cause effect in human-software interaction [6] [28] [65]. Human stress, human error estimates and human error rate prediction techniques are discussed in this chapter.

Human Error and Reliability in Human-Software Interaction

Human error can be defined as consisting of "any significant deviation from a previously established, required or expected standard of human performance, that results in unwanted or undesirable time delay, difficulty, problem, trouble, incident, malfunction, or failure [84]." In real world situations where discussions of precisely what is or is not a common-cause human error are of less importance than what can be done to prevent them, the operational definition may be restricted to those errors which: (1) occur within a particular set of activities, (2) are of some significance or criticality to the primary task under consideration, (3) involve a human action of commission or omission, and (4) could have been prevented through some feasible course of action [28].

A failure effect can be explained as the consequences a failure mode has on the operations, tasks, function or status of a system. Failure effects are classified as being of local effect, next higher level effect and end effect [4].

Generally speaking, sources of common-cause human error in software development tasks arise from several different factors, such as (1) human psychological and physiological stresses, (2) missing, incomplete, or erroneous knowledge, (3) inert knowledge (i.e. situation-relevant knowledge is not accessed under the conditions in which the task is performed). Stress with human error at a moderate level (see Figure 3.1) in some physiological and psychological situations is useful in increasing human effectiveness to its optimal level [28]. Obviously an over-stressed person will have a higher probability of making an error. In certain circumstances, there may be undesirable psychological or physiological tensions from work activities or environmental conditions that are beyond the reasonable or acceptable limits. In such cases stress and strain arise. Stress refers to some undesirable condition, circumstance, task, or other factor that impinges upon the individual, and strain refers to the effects of the stress. However, all common-cause human errors are not from these phenomena. There are many other of causal factors affecting software-task failure in human-software interaction.

Stress characteristics and stress check list factors in human-software interaction

There are at least eleven identified stress characteristics of a programmer in human-software interaction systems [71]. These are associated with the following situations:

(1) Information feedback to the programmer is inadequate for the determination of

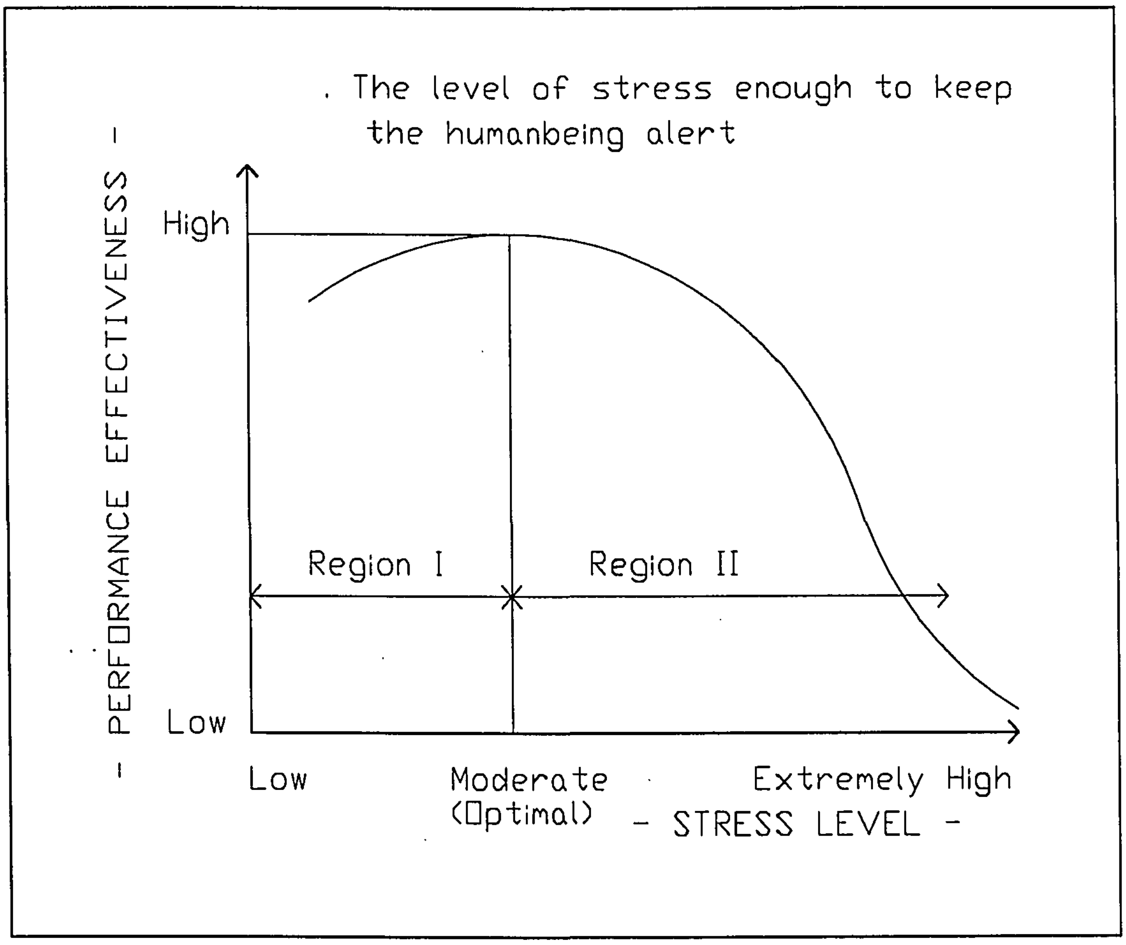


Figure 3.1: Performance Effectiveness as a Function of Stress Level.

correctness of his or her actions.

(2) The programmer is required to make comparisons of two or more displays quickly.

(3) Lack of knowledge or background for problem solving, understanding, and design causes pressure for lower level programmer.

(4) There is a requirement for prolonged design work by the software engineer.

(5) To perform a task, the sequence of steps needed is very long.

(6) More than one display are cumbersome to discriminate.

(7) There is a requirement to program in a manner more user friendly oriented than programmer oriented.

(8) Very competitive and high level intelligent design is requested for multi-version redundant software development.

(9) There is a requirement that decisions have to made on the basis of data collected from various sources.

(10) Other factors, such as short term memory [107], information overload, interference, multi-task overload, perceptual overload are present.

(11) Other demands are present that require or produce : vigilance, signal detection, information overload, uncertainty, lack of feedback, or time pressure.

Another list of stress inducing general environmental situations contains the following [8]:

(1) Having to work with programmers who have unpredictable temperaments;

(2) Being unhappy with the present job or program;

(3) Gaps in knowledge or familiarity with computer language, operating system, and hardware;

(4) Possibility of redundancy at work;

(5) Poor chances for promotion at work;

(6) Lacking the expertise to perform the required job;

(7) Poor health or physical;

(8) Performing under extremely tight time pressures;

(9) Having to take work home most of the time in order to meet deadlines;

(10) Excessive demands from superiors at work;

(11) Having to write a program below one's ability and experience.

Rook's model of human error occurrence

Rook [96] developed a mathematical model of error occurrence. This model can be used to compute the total probability of no functional failures over Z independent types of tasks. It requires the following assumptions:

(1) A number of different tasks involve a miss-function.

(2) In achieving the mission function, each task may be carried out more than once. In addition, one or more error modes may be associated with a task.

(3) The error modes are independent.

(4) The entire mission function may or may not fail totally due to an error.

The occurrence probability of a functional failure resulting from the kth error mode of the ith operational task is given as

$$F_{ki} = q_{ki} \cdot Q_{ki}$$

where

q_{ki}: the probability that the ith task arises in an error of the kth mode

Q_{ki}: the conditional probability that if the mode k error of the ith operational type occurs it will result in total function failure.

Human error estimates and reliability function

The basic unit of human reliability can be defined as the Human Error Probability(HEP), which is the probability of on error happening during some specified human task. HEP is defined as the number of errors of a specified type divided by the total number of chances for that error to occur [51].

Human error probability estimates can be provided per time rate and per demand rate, as follows [39]:

$$P_{he} = \frac{En}{Op_e}$$

where

P_{he}: the probability that when a specified task is carried out a human error will occur

En: the total number of known errors of a given type

Op_e: the total number of opportunities for the error.

A generalized human performance reliability function[5], [94], [95] is described with a time dependent human error(hazard) rate, $He(t)$ as:

$$He(t) = [\frac{-1}{Re(t)}][\frac{dRe(t)}{dt}]$$

where

$Re(t)$: human performance reliability at time t

$Re(t) = e^{-\int_0^t He(t)dt}$.

The human performance correctability function for continuous tasks is concerned with the human capability to correct self-generated human errors[94]. That function is defined by the probability that an error will be corrected in time t subject to a stress constraint associated with the task and its environment:

$$P_C(T) = 1 - e^{-\int_0^t rc(t)dt}$$

where

$r_C(t)$: the time dependent rate at which tasks are corrected, or

$$P_c(t) = \int_0^t f(t)dt$$

where

f(t): the probability distribution function associated with the time-to-correction completion.

Technique for Human Error Rate Prediction (THERP)

This methodology which was first established by Swain at Sandia Laboratories [110] has been developed to a level where it is regarded as the most powerful and systematic methodology for the quantification of human reliability. The basic tool used in THERP is the probability tree diagram (Figure 3.2[1]). Human error that may be defined as deviations from assigned tasks often appears as basic events in fault trees. A THERP analysis initializes by decomposing human tasks into a sequence of unit activities. Possible deviations are postulated for each unit activity. A Human Reliability Analysis (HRA) event tree is then used to visualize the normal sequence of unit activities together with the deviations. The event tree thus becomes a collection of chronologically associated human tasks. Each sub-branch of event tree represents either normal execution of a unit activity or an omission or a commission

[1]The probability of selecting the correct ignition command is t.

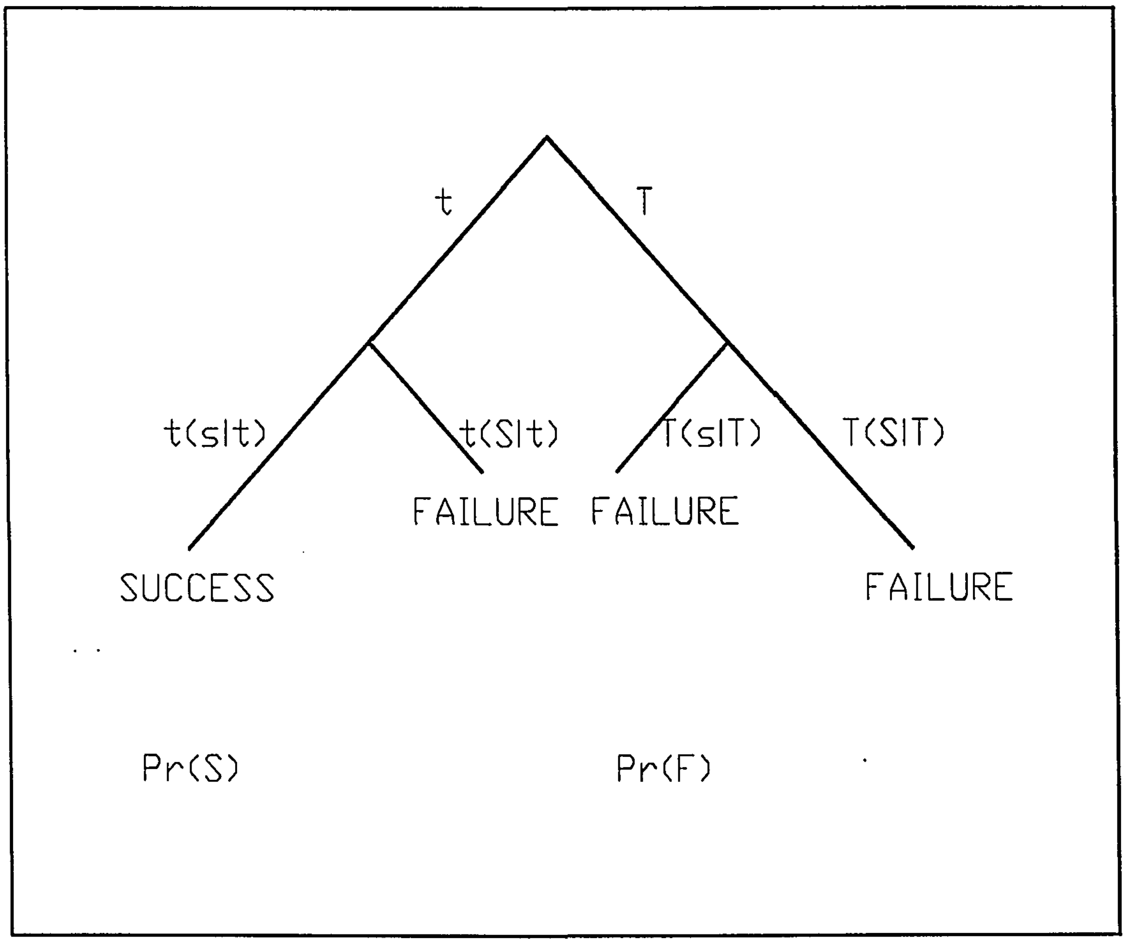


Figure 3.2: Probability Tree Diagram for a Programming Task.

error related to the activity. Human error appearing as a basic event in a fault tree can be defined by a subset of terminal nodes of the event tree.

The occurrence probability of the basic overall event is calculated after probabilities are assigned to the branches of the event tree. Probability estimates on the branches must reflect performance shaping factors specific to human operating systems, and other boundary conditions. Events described by branches can be statistically dependent. The probability of a correct outcome C, completing a programming task, is the product of the two probabilities in the tree [43]:

$$P_r(C) = t(s \mid t).$$

The probability of failure can be calculated as

$$P_r(F) = t(S \mid t) + T(s \mid T) + T(S \mid T)$$

Where

t: the probability of successfully locating the correct command

T: the probability of selecting the wrong command

$s \mid t$: the probability of a successful one-trial run given that the proper command was selected

$S \mid T$: the probability that the software fails to run again given that you have the proper command was selected.

The outputs of the THERP model are estimates of correct or failure probabilities for human behaviors and tasks.

Human-software systems reliability

Here, human-software reliability can be defined as the probability of accomplishing a task successfully by humans at any required stage in a human-software interaction within a specified minimum time limit.

Software task reliability prediction procedure: The main objective of the procedure of software task reliability prediction is to obtain subtask reliability estimates for which no previous reliability data is available [30]. To obtain a total task estimate, subtask estimates may be combined. The following six steps are from this method:

(1) Object Orientation: The tasks are to be performed when a complete operation is represented by each task. A task is composed of a series of subtasks.

(2) Subtask identification: Once the tasks to be performed are identified then the next logical step is to identify the subtasks of each task.

(3) Concurrent Design: Simultaneous design is provided for the software product and development process in human-software interaction and the system task of software development.

(4) Obtaining empirical data: This type of subtask data may be available from a number of sources such as in-house operation, experimental literature, laboratory, and so on.

(5) Estimating task performance: This is considered along with rating each subtask in accordance with its potential for error or level of difficulty. A scale from 1 to 10 points corresponding from least error to most error can be used to rate a subtask. This kind of rating is purely based on individual judgement.

(6) Analyzing the task performance error rate: To get a subtask reliability estimate,

the empirical data is expressed and the judged in comparison with a straight line. The line is tested for the goodness of fit. This line can be used to estimate subtask reliability.

(7) Determining each task reliability: The total task reliability is given by the product of subtask reliabilities taken from the equation of the straight line.

Common Cause Failures in Human–Software Interactions

Common-cause failures, which were overlooked 20 years ago, have been receiving wide attention especially in the software systems area. This is because the assumption of statistically-independent failures of redundant systems is easily violated in the case of human-software interaction. Common-cause failures concern the possibility that system or mission failure involving multi system component failure may occur due to a *common cause* , i.e. the loss during some critical period of multiple, redundant systems, component functions, due to an underlying common control mechanism, fault or phenomena.

A related study for common-cause failure initially, a small research group on Rare Event was to investigate and organize the program of work based on the findings of the task force in the following areas:

- rare event data collection and analysis;
- common mode failure analysis;
- human error analysis and quantification;
- statistics and decision theories applicable to rare events;
- inter disciplinary communication and tutorial programs on rare events programs and their solution.

I. A. Watson described that the analysis of common-cause may be complicated because of various considerations including [116]:

(1) recognition of many possible causes of common-cause failure and their identification;

(2) selection of models to be used in the quantification of system reliability;

(3) the availability of historical data;

(4) the comparative rarity of common-cause failure.

There is a different situation in a human-software system compared with a hardware system. A failure in one transaction processor(TP) software component would also occur in another since they are identical [76]. Both copies contain identical faults. Since the software components are not independent of each other in regard to failure behavior, software redundancy does not improve reliability. This is a commonly occurring and very important point for software components. A common-cause failure in the software development processing system is any instance where multiple components malfunction due to a single cause.

At first, in modeling common cause failures in software development, it is desirable to introduce the initiating events physically. An initiating occurrence is to be regarded as an external event such as a flood, earthquake, power outage, or fire which can cause the failure of several components simultaneously, due to the environmental stresses occasioned by its occurrence.

Also simultaneously, another common cause failure of several components occurs when one component has several functions, so that its failure prevents each of these individual functions.

Another possible common cause is the existence of standby components which are called into use when specified components have failed. The conditional waiting time until a failure in the standby component is observed is different from the waiting time until failure if it were in non-standby usage.

The following are some causes of common-cause failures in general systems [30]:

(1) External abnormal environments: dust/dirt, temperature, humidity/moisture, vibrations.

(2) Equipment failure resulting from some unforseen external event: fires, floods, earthquakes, tornadoes.

(3) Design deficiencies: During the design phase of the system some faults may have been overlooked. For example, the interdependence between electrical and mechanical items of a redundant system may have been overlooked during the design phase of a system.

(4) Operation and maintenance errors: Occurrence of these errors may be due to improper maintenance, carelessness, improper calibration, the same person performing maintenance on all redundant units repeating the same mistake on all of them, etc.

(5) Multiple items purchased from the same manufacturer: All these items may have same manufacturing defects.

(6) Common external power source to redundant units.

(7) Functional deficiency: misunderstanding of process variable behavior, inadequacy of designed protective action, inappropriate instrumentation, etc.

Common-cause failure analysis of redundant systems

There is a method for incorporating common-cause failure in a redundant network analysis of the human-software processing system [30]. There is an assumption that network units are identical and independent, and also that the same portion of common-cause failures is associated with other redundant network components.

It is assumed in the common-cause failure model that

$\gamma \equiv$ fraction of component or system failures that are common-cause

$\lambda_u = \lambda_i + \lambda_c$

where

λ_u: the component constant failure rate

λ_i: the component independent constant failure rate

λ_c: the component or system constant common-cause failure rate

Since

$$\gamma = \frac{\lambda_c}{\lambda_u}$$

then

$$\lambda_c = \gamma \lambda_u$$

By arranging these equations, we get

$$\lambda_i = (1 - \gamma)\lambda_u$$

Parallel Component Network Systems: A series-parallel component network system model in multi-version software is illustrated in Figure 3.3. This is actually a modified parallel network system to incorporate common-cause failures. The parallel portion of the network represents n independent failure components and the single component in series is a hypothetical component representing system common-cause failures [76]. The failure of the hypothetical common-cause failure component will cause system failure.

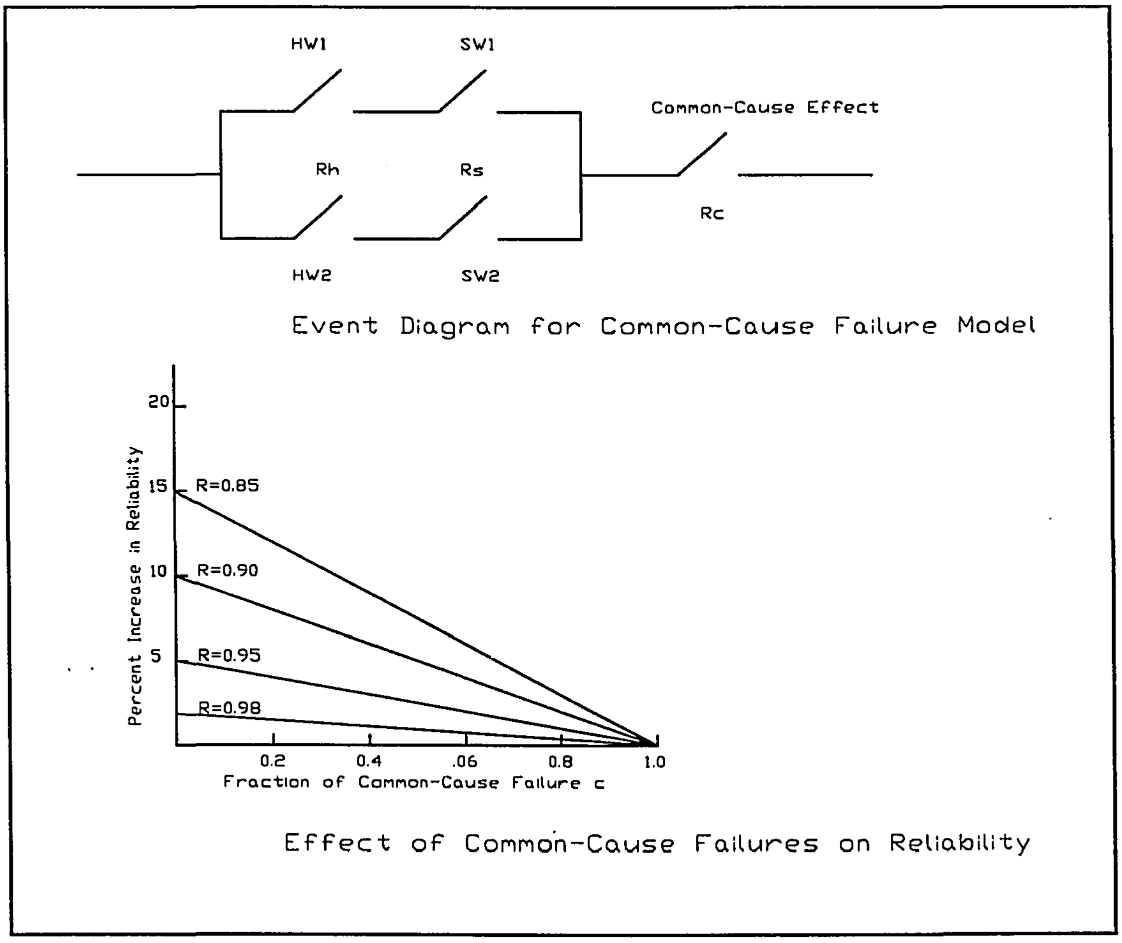

HW1
SW1
Common-Cause Effect
Rh
Rs
Rc
HW2
SW2
Event Diagram for Common-Cause Failure Model
Percent Increase in Reliability
20
15
10
5
R=0.85
R=0.90
R=0.95
R=0.98
0.2
0.4
.06
0.8
1.0
Fraction of Common-Cause Failure c
Effect of Common-Cause Failures on Reliability

Figure 3.3: Event Diagram for Common-Cause Failure Model and Effect of Common-Cause Failures on Reliability in Human-Software Interaction [76]

The human-software reliability, R_{hs}, of the human-software processing network given in Figure 3.3 is

$$R_{hs} = [1 - (1 - R_i)^n]R_C$$

where

R_i: the independent failure mode reliability of a component

R_C: the common-cause failure mode system reliability

n: the number of identical components

The time dependent reliability of the ith independent component with constant failure rate is

$$R_i(t) = e^{-\lambda_i t}$$

Similarly, the hypothetical common-cause failure component reliability is

$$R_C = e^{-\lambda_C t}$$

By substituting variables in all of these equations

$$R_{hs}(t) = [1 - (1 - e^{-(1-\gamma)\lambda_u t})^n]e^{-\gamma\lambda_u t}$$

To calculate the mean time to failure(MTTF), $R_{hs}(t)$ is integrated over the time interval $[0, \infty]$

$$MTTF = \int_0^{\infty} R_{hs}(t) = \frac{[\sum_{i=1}^{n}(-1)^{i+1}\binom{n}{i}]}{[\lambda_u\gamma + n\lambda_u(i - \gamma_i)]}.$$

The common-cause principle and statistical dependence are described in Appendix A.

CHAPTER 4. A COMMON CAUSE MODEL AND EXPERIMENTAL DESIGN IN HUMAN-SOFTWARE INTERACTION

Common-Cause Model and Function

Common-cause model

The common-cause model can be used to define internal common-cause human-based error and to develop a common-cause error control mechanism for human-software interaction. It can be explained in terms of four schematic and systematic design stages, as illustrated in Figure 4.1. The stages are as follows:

(1) Human-software interaction components: These system components are the basic elements and factors in human-software interaction. They are: the human working as a software engineer, software as a operating system, and hardware as a system work station. The common-cause error occurs in system interactions involving failures among these system components.

(2) Common-cause error protocol: Common-cause error protocol is the actual location and identification of common-cause error attributed to a common-cause effect in a redundant system of multi-version software development. It is distinguished within a given common-cause error mode by its individual identification, by a pattern recognition, and by a behavior domain.

(3) Common-cause error function: This is the function of common-cause error revealed in the existence and the performance allocation of each common-cause error mode using an evaluation typically by three variables such as frequency or error occurrence, error correction time, and point of error occurrence in time.
(4) Common-cause analysis, representation, and system redesign: This stage consists of the analysis and representation of common-cause error in human-software interaction. Several analytical methods have been provided to define common-cause human domain error, and to redesign the system interaction with representational results and prevention schemes involved with system development productivity, and common-cause error control mechanisms.

Common-cause function

The common-cause function is shown in the existence and in performance allocation of common-cause failure with its identification(I_i), pattern recognition(P_j), and behavior domain(B_k) of common-cause error mode. Each allocated common-cause error mode is evaluated by performance variables using common-cause error frequency($F_{i,j,k}$), error correction time($C_{i,j,k}$), point of error occurrence in time($O_{i,j,k}$) during the software development period. These are illustrated by Figure 4.2. The common-cause function, C_r is:

$$C_r = C(I_i(F_i, C_i, O_i), P_j(F_j, C_j, O_j), B_k(F_k, C_k, O_k))$$

subject to

$$\sum I_i = 1, 0 \leq I_i \leq 1$$

$$\sum P_j = 1, 0 \leq P_j \leq 1$$

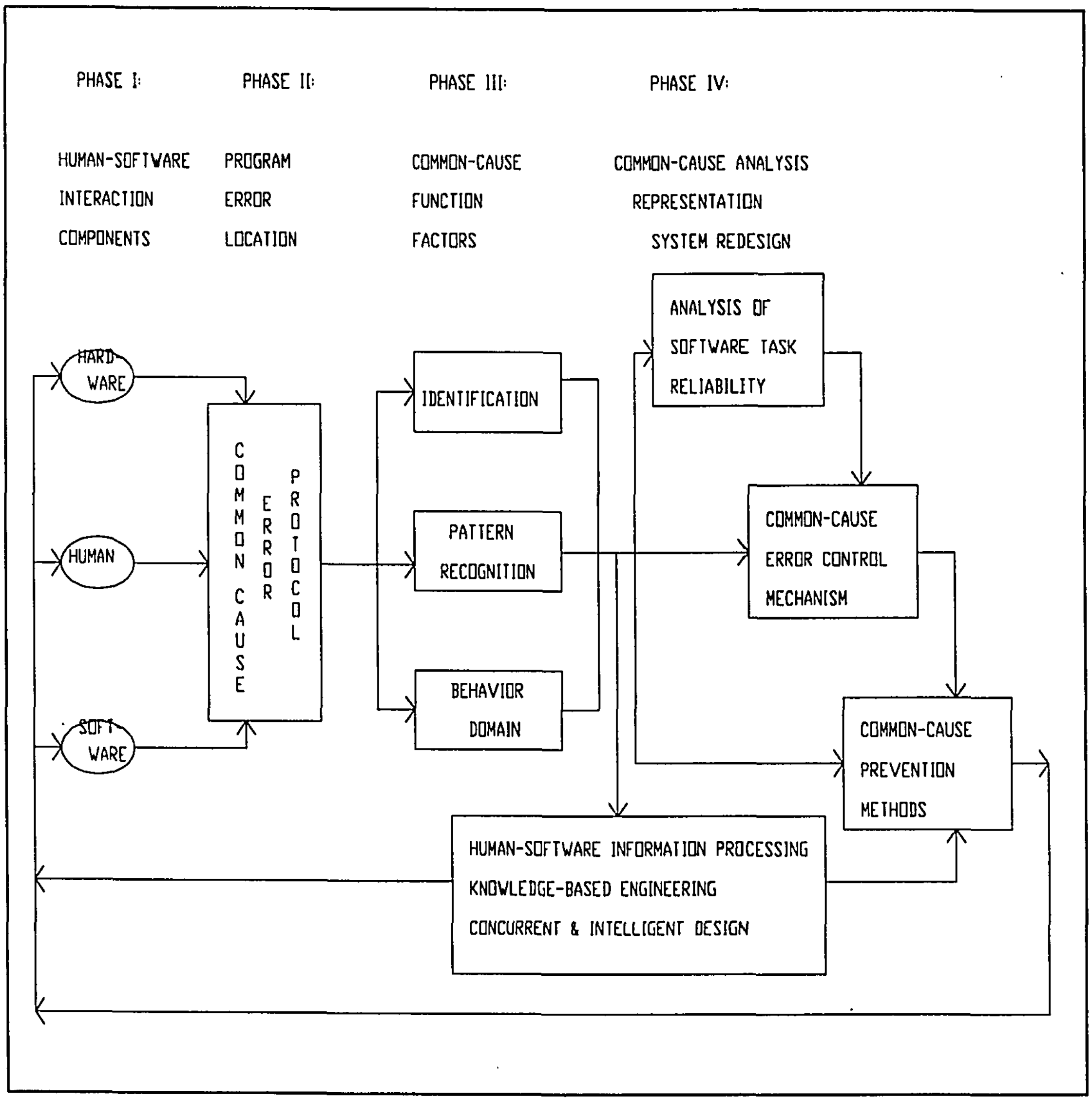

PHASE I:
PHASE II:
PHASE III:
PHASE IV:
HUMAN-SOFTWARE INTERACTION COMPONENTS
PROGRAM ERROR LOCATION
COMMON-CAUSE FUNCTION FACTORS
COMMON-CAUSE ANALYSIS REPRESENTATION SYSTEM REDESIGN
HARD-WARE
HUMAN
SOFT-WARE
COMMON CAUSE ERROR PROTOCOL
IDENTIFICATION
PATTERN RECOGNITION
BEHAVIOR DOMAIN
ANALYSIS OF SOFTWARE TASK RELIABILITY
COMMON-CAUSE ERROR CONTROL MECHANISM
COMMON-CAUSE PREVENTION METHODS
HUMAN-SOFTWARE INFORMATION PROCESSING
KNOWLEDGE-BASED ENGINEERING
CONCURRENT & INTELLIGENT DESIGN

Figure 4.1: Schematic Design Stages of the Common-Cause Model

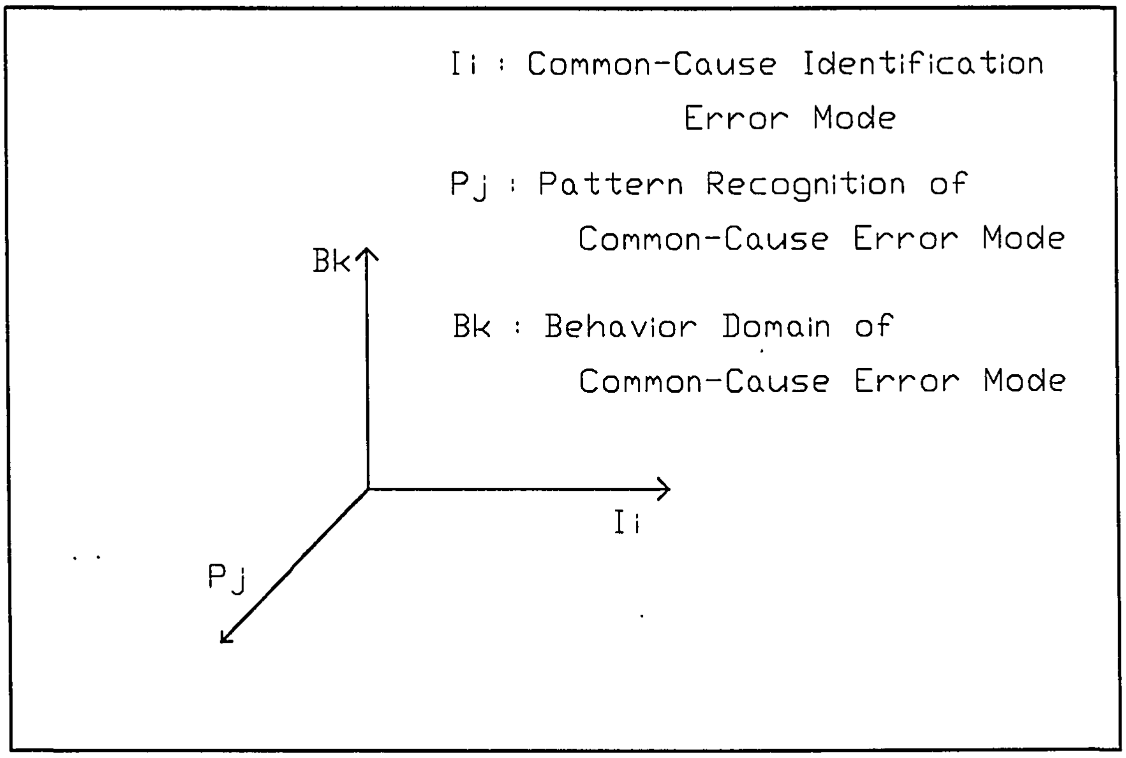

Ii : Common-Cause Identification Error Mode
Pj : Pattern Recognition of Common-Cause Error Mode
Bk : Behavior Domain of Common-Cause Error Mode
Bk
Ii
Pj

Figure 4.2: Three Common-Cause Error Modes and Evaluation Variables

$$\sum B_k = 1, 0 \leq B_k \leq 1$$

$$\sum F_i = 1, 0 \leq F_i \leq 1$$

$$\sum F_j = 1, 0 \leq F_j \leq 1$$

$$\sum F_k = 1, 0 \leq F_k \leq 1$$

$$\sum C_i = 1, 0 \leq C_i \leq 1$$

$$\sum C_j = 1, 0 \leq C_j \leq 1$$

$$\sum C_k = 1, 0 \leq C_k \leq 1$$

$$\sum O_i = 1, 0 \leq O_i \leq 1$$

$$\sum O_j = 1, 0 \leq O_j \leq 1$$

$$\sum O_k = 1, 0 \leq O_k \leq 1$$

where

I_i: programming identification error mode

P_j: reasoning pattern error mode

B_k: behavior domain error modes

$F_{i,j,k}$: Common-cause error frequency in each mode

$C_{i,j,k}$: Common-cause error correction time in each mode

$O_{i,j,k}$: Common-cause error occurrence time zone in each mode.

The common-cause function consists of these three reasoning factors of common-cause error mode, identification, pattern recognition, and behavior domain of common-cause error mode. Certain common-cause errors have these three different axes of reasoning modes, with which can be evaluated by the three subjects' performance variables, frequency, correction time, and point of occurrence in time, using the appropriate portion of the total amount of collected data relating to all errors.

Consideration factors and environmental conditions of human-software interactions

There are some modifying factors and environmental conditions for the three components in human-software interaction.

- The human side of the interaction factors including:

  Human availability: manning and working load levels

  Human capability: skill and knowledge levels

  Human performance: completion of required tasks

  Human productivity: quantity and quality produced per unit time

  Human safety, biomechanics, work physiology

- The software side of the interaction factors including:

  Specification of Requirements:

  Design: software product design, process design

  User-friendliness: human oriented, easy use, objective orientation

  Interface with hardware: hardware capacity with software size

  Software productivity: efficiency, utilization, cost, interactions

- The hardware side of the interaction factors including:

  Information displays: the display format, display device adaptation to human

  1 cmergonomics

  Display characteristics: symbol size, shape, color, density, etc.

  Data organization and output: the architecture producing hierarchical levels

  1 cmof data specificity

  Communications: command mode types, error messages, prompts, alerts,

  1 cm queries, etc.

Load procedures: task sequences, decision making and its principles

Data processing: data entry, manipulation, designation, data flow

System documentation: hard copy manuals and aids

Common-Cause Error Protocol and Common-Cause Factors

There are three features of internal common-causes, previously introduced, that can be used in determining the identification of programming error modes, pattern recognition, and behavioral error categories of common-cause errors in human-software interaction. They are: identification of common-cause error protocol(I_i), reasoning pattern error modes(P_j), and behavior domain error modes(B_k).

Identification of common-cause error protocol

There are eight identification modes($:I_i$) categories of typical human-based programming error from common-cause error protocols, which are used in the determination of the common-cause error that caused the failure. Each error protocol mode means the actual location of common-cause error and contributes to the common-cause effect at each stage of human-software interaction for multi-version redundant software development system [112].

I_1 System design and requirement errors:

Design not-responsive to requirements

Problem definition error

Requirement specification and task complexity error

Design specification, inappropriate methodology

I_2 Variable setting and program handling errors:

File not rewound before reading

Data not initialization not done

Program initialization, and dimensional declaration error

Variable setting and indexing error

Variable referred to by the wrong name

Incorrect variable type

Subscripting error

I_3 Program input and data base error:

Invalid input read from correct file

Input read from incorrect file or subroutine

Incorrect input format, statement referenced

Data base problem, and data manipulation error

Data sorted incorrectly

End of file encountered prematurely

I_4 Computation based errors:

Incorrect operand in equation

Incorrect use of parenthesis

Sign convention error

Units or data conversion error

Computation produces an over/under flow

Incorrect/inaccurate equation used

Precision loss due to mixed mode

Missing computation

Rounding or truncation error

I_5 Program logic errors:

Incorrect operand in logical expression

Logic activities out of sequence

Wrong variable being checked

Missing logic or condition tests

Too many/few statements in loop

Loop iterated incorrect number of times (including endless loop)

Duplicate logic

I_6 Human-software system interface errors:

Wrong subroutine called or nonexistent subroutine call

Call to subroutine put in wrong place

Subroutine arguments not consistent in type, units, order, etc.

Software/data base interface error

Software user interface error

Software/software interface error

System configuration error

Software not compatible with project standards

I_7 System operation errors:

Operating system error

Operating command error

Interactions problems with hardware system

Test execution error

Compilation error

Operating-user misunderstanding error

Configuration control error

I_8 Output and output formatting errors:

Data written in wrong file and allocation error

Data base according to the wrong format statement

Data written in wrong format

Data written with wrong carriage control

Incomplete or missing output

Output field size too small

Line count, spacing, or page eject problem

Output garbled or misleading

Pattern recognition error modes

Common-cause reasoning patterns(:P_j) can be recognized with causal characteristics which implicate the identical elements of reason, perception, control mechanism, occurrence processing, stimulus response requirement, etc. Each identical property or reason matches a pattern recognition for the common-cause human error mode [121] [33].

P_1 Knowledge deficiency: There is a lack of knowledge based on the hardware system, operating system, human-software interface, field of specific requirements or problem solving methodology.

P_2 Design deficiency: During the design phase of the human-software interaction system, some common-cause errors have been overlooked. These are in preliminary and detailed design work, the design reviews, definition of variables and attributes, and all work done prior to coding. For example, the interdependence between system requirements and output results, or between logical units and data flow of programming have been overlooked during the design phase of a system.

P_3 Operation and maintenance errors: Occurrence of these common-cause errors may be due to improper maintenance, carelessness, or improper calibration. The same program performing maintenance on all redundant units of human-software system may repeat the same mistake on all of them.

P_4 Functional deficiency: This includes misunderstanding of process variable behavior or specific requirements, inadequacy of designed protective action, inappropriate use of methods or instrumentation, or inadequacy of component processing in human-software interactions.

P_5 Syntax error: These result from expressions which are incorrect in the language being used regardless of the context in which they appear (example:) ($\longleftarrow$ ( )). Detection of these errors may be allowed through a relatively superficial analysis using grammatical rules of programming language. The programmer may detect and correct such errors as a matter of course during the programming process.

P_6 Semantic error: These occur when syntactically correct components of a program imply conditions which are untrue or impossible in stated combinations (example: UNIT=DISK,UNIT=PUNCH: from IBM JCL DD statement). This statement is syntactically correct, but it is impossible to allocate two different physical units to a single logical unit. These kinds of errors may require extensive analysis covering various interacting aspects and components of a program.

P_7 Logical error: These produce incorrect results but otherwise cause no obvious malfunction of the program (example: 1.0 / X + 1.0 is not equal to 1.0 / (X + 1.0)). There is probably little which can be done in terms of redesigning compilers to aid the programmer in eliminating such errors. These errors show a lack of fit of the program to the calculation logic. Also, the program may exactly solve a different problem from the one intended.

P_8 Clerical error: These may appear to be either syntactic or semantic errors. They are only partly a function of the language used. They result from mispunched, misplaced, or mis-copied cards, misread program drafts, card shuffling, or incorrect tape mounting.

P_9 System complexity: In human-software interaction systems, especially with a large-scale programming project, special difficulties arise from system components, comparing and contrasting the given requirement, the type and size of computer used, selection of proper programming language, memory size and speed required, processing time, decomposing the problem into subproblem, functions, models, and analysis. This system complexity appears to be judgmental or managerial in nature and cannot be easily defined with a lack of relation to the specific tasks of the software engineer.

Programming behavior domain error modes

There is a common-cause error category in terms of the programmer's behavioral aspects ($:B_k$) or point of view (Figure 4.3). Such common-cause error factors may be representative of from the human information processing, knowledge based design, error control mechanism, and human behavioral science [90] [92].

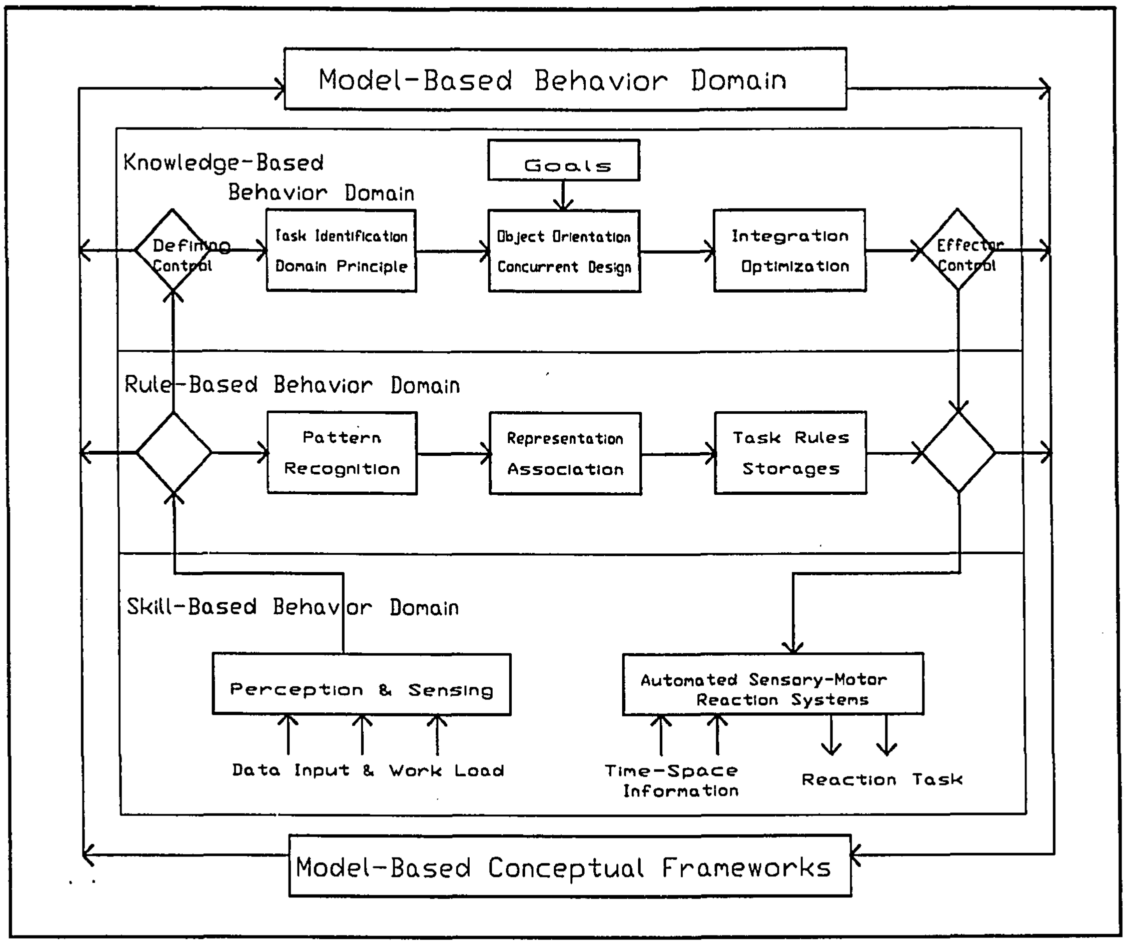

Model-Based Behavior Domain
Knowledge-Based Behavior Domain
Goals
Defining Control
Task Identification Domain Principle
Object Orientation Concurrent Design
Integration Optimization
Effector Control
Rule-Based Behavior Domain
Pattern Recognition
Representation Association
Task Rules Storages
Skill-Based Behavior Domain
Perception & Sensing
Automated Sensory-Motor Reaction Systems
Data Input & Work Load
Time-Space Information
Reaction Task
Model-Based Conceptual Frameworks

Figure 4.3: Programming Behavior Error Domain Mode

B_1 Skill-based behavior domain

$B_{1.A}$ Perception and sensing

$B_{1.B}$ Automated sensory-motor reaction systems

The skill-based behavior domain is a sensory-motor pattern, controlled and automated behavior, controlled by the structure of the adaptive patterns stored in the human nervous system. It means that this human error behavior is controlled by psychological laws and physiological mechanisms governing the human software processing structure and the concept of human behavioral perception and cognition. Some characteristics of this skill-based behavior mode are as follows:

(1) Sensory-motor variability

(2) Recency and frequency

(3) Topographic misorientation

(4) Environmental control signal

(5) Stereotype mismatching

(6) Shared schema features

(7) Adaptation and fine tuning

B_2 Rule-based behavior domain

$B_{2.A}$ Pattern matching and recognition

$B_{2.B}$ Representation and association

$B_{2.C}$ Working memory and rule interpreter

The rule-based behavior scheme is a human-software interaction that represents human reasoning with grammatical language structure and logical allocation rules. The rule-based systems represent the solution to a problem as a set of rules that specify "how some string of symbols may be transformed into other strings of symbols," such as a simple form of pattern matching. The transformation of one pattern to another in a rule-based language is understood to represent an IF-THEN implication. Rules can express associations between state and task. Some characteristics of this rule-based behavior mode are as follows:

(1) Habit robustness

(2) Typical fixation

(3) Availability

(4) Omission of an isolated function

(5) Over-simplification

(6) Alternative mistake

(7) Over-confidence

(8) Stereotype recognition

(9) Matching bias

B_3 Knowledge-based behavior domain

$B_{3.A}$ Task identification and domain principle

$B_{3.B}$ Object orientation and concurrent design

$B_{3.C}$ Integration and optimization

The knowledge-based behavior scheme is a human behavioral phase interacting with software development concerned with the design and implementation of programs which are capable of emulating human cognitive skills such as problem solving, task identification with domain principle, object orientation relative to the goal, concurrent design of software product and human-software interaction processing, and optimal system integration. The structure of the behavior is an evaluation of the situation, designing of a proper sequence of actions to achieve the goal. It depends upon fundamental knowledge of the processes, functions and anatomical structure of the system. Some characteristics of this knowledge-based behavior mode are as follows:

(1) Human variability

(2) Selectivity

(3) Adaptation

(4) Working memory limitation

(5) Errors in a causal structure

(6) Availability

(7) Matching bias revisited

(8) Need for human decision making

(9) Memory cueing/reasoning by analogy

(10) Incorrect and incomplete knowledge.

B_4 Model-based behavior domain:

A highly reliable human-software interaction model yields cognitive design base strategies to define models for adaptive interface. Communication strategies for basic system design, information processing, knowledge of components, and systems configuration of interface, must be represented explicitly. The following are some adaptive concepts of model base strategies and design: symbolic and quantitative model, performance and cognitive model, static and dynamic model, syntactic and semantic model, state-transition model, singular and multiple model, etc.

Experimental Design and Procedure

General description of experiment

This project involves an experiment in the cognitive aspects of software project design. Its purpose is to analyze common-causes of software development related human error and to identify software design factors contributing to common types of error occurring in human-software interaction. The results and analytical procedures developed during this study can be applied to improving reliability of software development and to providing guidelines for design of software development.

The main experiment was conducted with ten experts in programming (5 subjects for FORTRAN and 5 subjects for C) who were paid $6 per hour and were each given a programming assignment for determining the optimum sequence of machine replacement or an optimal inventory system. Three prior pilot experiments were conducted previously using 33 undergraduate students and 13 graduate students paid $6 per hour. These experiments conducted to data using beginners (level 1: 20 subjects), two year experienced subjects (level 2: 13 subjects), and 5-8 year experienced programming experts (level 3: 13 subjects) were based on the use of LOTUS-123, FORTRAN, or C in a programming application in shop scheduling and inventory control, given initial cost and demand data. The reader is referred to Appendix B for these programming requirements.

All materials, such as subject selection, requirement specifications, experimental procedures, data collection sheets, and analytical materials for the experiment were prepared and subject life data were gathered. For reliable subject calibration, subjects were trained using the actual requirements and overall experimental procedures in an initial session and consultation session. Their skill levels were evaluated according to programming experience and knowledge background for requirement specifications. All personal data were kept confidential. After three pilot experiments, the main experiment was conducted with data collection according to frequency of common-cause error occurrence, error correction time, and point of error occurrence in time for each of the categorical factors: identification of common-cause error mode, pattern recognition of common-cause error reasons, and behavior domain of common-cause error mode as explained earlier in this chapter.

As the programming subject set up and programmed according to the task requirements using FORTRAN or C, his/her programming task was observed by the supervisor who pre-classified and designed the common-cause function and common-cause error factor modes. During the experiment, the contents of common-cause human error in subject programming failure were, first, recorded with an explanation of the reasons for those failures, correction time, and point of error occurrence in time. Then, at the representational interview session held every 30-45 minutes, the common-cause error protocol was allocated to each of the common-cause factors, identification, pattern recognition, and behavior domains, by directed definition and cooperative decision with the supervisor and the subject. Experimental data was then validated and analyzed by statistical methods and a geometrical method using vector analysis and mapping designed for use in analyzing common-cause errors in human-software reliability and interactions. Figure 4.4 shows the experimental procedure and design used in this experiment involving human-software interaction processing. Results were derived using the following analytical methods: common-cause error mode data and table, mapping and geometric vector evaluation in hexahedron contours, value of common-cause function with simulated rating, historical common-cause error recovery time zone, transition relationship diagram, grouping of major common-cause factors, and correlation and regression analysis of categorical factors. Verification of the results using expert subjects was intended to identify clearly those factors related to the design of software development as distinguished from conditional factors associated with level of subject, type of language, and type of requirement.

Finally, the characteristics and the properties of common-cause failure modes in human-software interaction were determined by the analysis of experimental data collected on the ten expert subjects and compared with data from each of the categor-

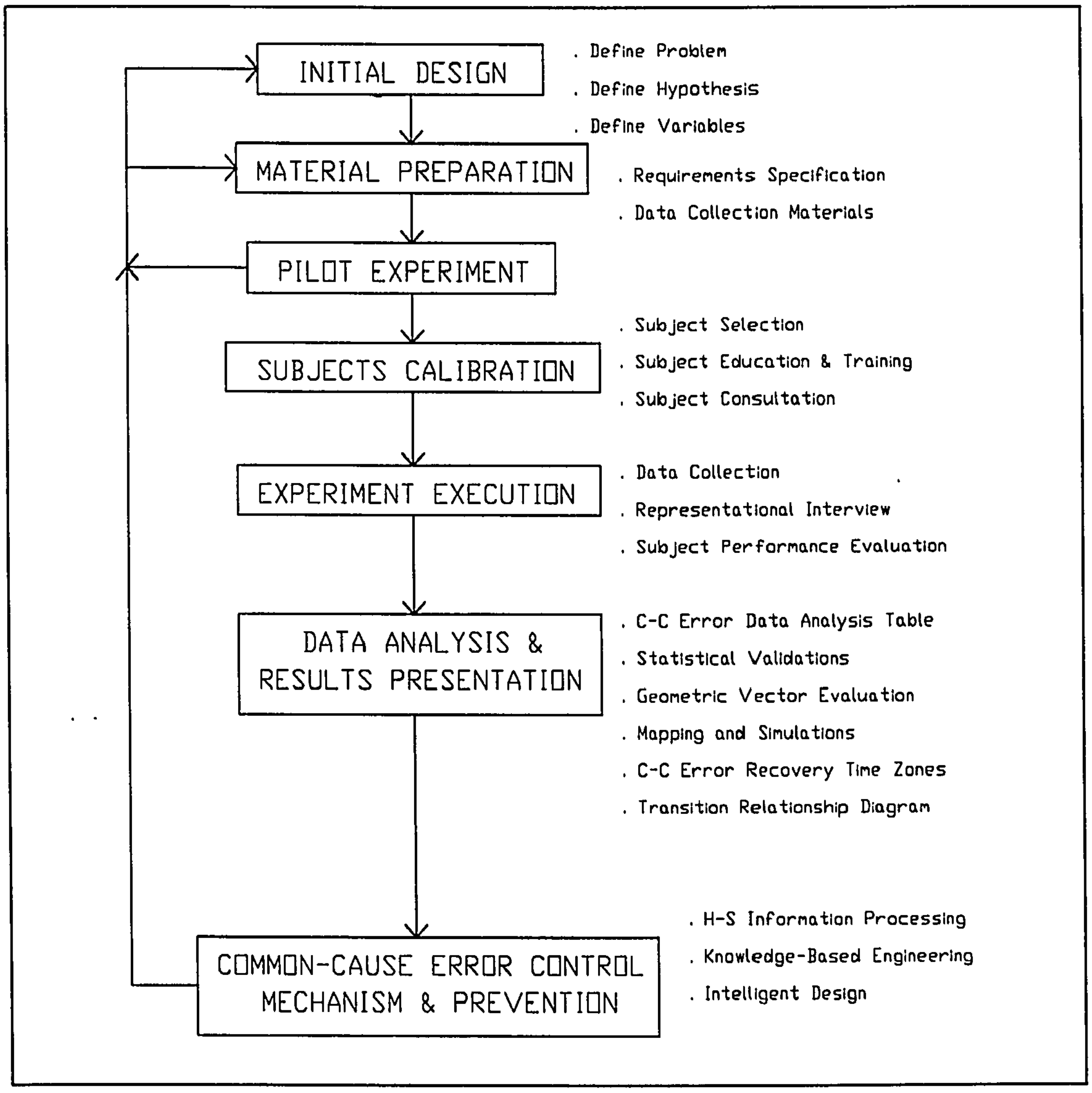

INITIAL DESIGN
. Define Problem
. Define Hypothesis
. Define Variables
MATERIAL PREPARATION
. Requirements Specification
. Data Collection Materials
PILOT EXPERIMENT
SUBJECTS CALIBRATION
. Subject Selection
. Subject Education & Training
. Subject Consultation
EXPERIMENT EXECUTION
. Data Collection
. Representational Interview
. Subject Performance Evaluation
DATA ANALYSIS & RESULTS PRESENTATION
. C-C Error Data Analysis Table
. Statistical Validations
. Geometric Vector Evaluation
. Mapping and Simulations
. C-C Error Recovery Time Zones
. Transition Relationship Diagram
COMMON-CAUSE ERROR CONTROL MECHANISM & PREVENTION
. H-S Information Processing
. Knowledge-Based Engineering
. Intelligent Design

Figure 4.4: The Experimental Procedure in Human-Software Interaction

ical conditions. Results obtained during the earlier pilot study conducted suggested a new cognitive paradigm designed to eliminate and reduce the most common types of human domain error related to design of software development. These results have direct application in common-cause error control and prevention.

Problems and hypotheses of experiment

Defining questions to be answered through the main experiment

(1) What are the contents and conditions of human-based errors affected by the common-cause effect in human-software interaction?

(2) What are the frequency of common-cause error, error correction time, and point of error occurrence in time in each of the common-cause error modes?

(3) What are the major reasoning common-cause factors for each error recovery time zone?

(4) What is the relationship between the behavior domain of common-cause error mode and identification of common-cause error mode, or pattern recognition of error?

(5) How much difference in common-cause reasoning factors is there among the categorical conditions of subject such as level of subject, type of language, and type of requirement?

(6) What are the alternative results with the different rated simulations?

Hypothesis More qualified expert subjects in software development will give a better performance, but the major portion of common-cause error properties in human-software interactions will not differ significantly among all subjects who have different categorical conditions such as level of programming expertise, knowledge of programming language, and type of task requirements.

Procedure and method of experiment

Variables and control factors of the experiment The chosen experiment designed to study the human aspects of software development included the following:

(1) Dependent variables:

Common-cause error modes:

Common-cause error identification mode (I_i)

Reasoning pattern error mode (P_j)

Behavior error domains (B_k)

Failure/error frequency ($F_{i,j,k}$)

Subject task performance factors:

Correction time to each common-cause error ($C_{i,j,k}$)

Point of failure/error occurrence time in each error mode ($O_{i,j,k}$)

(2) Independent variables:

Requirement specifications

Subject expertise level

Programming language

(3) Controllable Factors:

Type of task (requirements of assignment)

Program task size

Type of hardware and operating system

(4) Uncontrollable factors:

Subject factors (subject life data, programming experience, typing skill, intelligence, attitude, knowledge)

External common-cause factors (fires, earthquakes, tornadoes, etc.)

Abnormal environments (temperature, humidity/moisture, vibrations)

(5) Conditional factors:

Level of subject: expert 1, expert 2

Type of language: FORTRAN, C

Type of requirement: shop scheduling, inventory system

Preparation of experimental materials Experimental materials were prepared for requirement specifications, experimental procedures, data collection, and data analysis and representations (Appendix B).

(1) Programming requirements including determining the optimal sequence for machine replacement or optimal inventory policy using a simulation consisting of 300-400 lines using Fortran or C

(2) Subject level evaluation

(3) Consultation support

(4) Experiment procedure and subject note

(5) Data collection sheet, error and failure description modes

(6) Questionnaires for personal life data bank

Pilot Experiments A pilot experiment was needed for evaluating and testing the experimental design. Analysis of the pilot experiment resulted in a redesign and a re-assessment of effects to be observed.

(1) Project 1 (beginner subjects with LOTUS-123): Project 1 was a three week experiment using beginner subjects (level 1), and LOTUS-123, a spread-sheet man-

agement software package. It involved the analysis of an inventory control problem. Software users, as experimental subjects, simulated the performance of an inventory management procedure under random demands, selecting management parameters for optimum (that is, lowest cost) inventory control. During the software task, the subject as a programmer described his/her reasons for errors in programming behavior. An observer monitored the data collection and recorded a description of programmer's error modes.

Experimental period: Key experiment: 10/22, 1990 - 11/16, 1990

Group A: Tue. 8-11 (3hrs/w)

Group B: Thur. 8-11 (3hrs/w)

Subjects(20 subjects):

Group A: 10 students

Group B: 10 students

Experimental design tools:

Subject programming bases: LOTUS-123 (150-200 lines)

Data analysis: SAS, LOTUS-123

(2) Project 2 (2-year experienced intermediate programmers using FORTRAN): Project 2 was conducted by the 2 year experienced intermediate programmers (level 2) using FORTRAN. The task requirement involved the determination of an optimal sequence of machines to employ in providing service for a number of years. The development started with a manual exercise and design of a program to determine appropriate methods, then proceeded with the development of the FORTRAN program to implement the algorithm. During the FORTRAN programming, the programmer's task behavior was observed by the project navigator to collect common-cause errors in

human-software interactions.

Experimental period: Key experiment: 2/7, 1991 - 3/5, 1991

Group A: Thur. 8-11 (3hrs/w)

Group B: Thur. 11-2 (3hrs/w)

Subjects: (13 subjects)

Group A: 7 students

Group B: 6 students

Experimental design tools:

Subject programming bases: FORTRAN (180-230 lines)

Data analysis: SAS, LOTUS-123

(3) Project 3 (5-8 year experienced expert programmers using FORTRAN or C): Project 3 was conducted by 5-8 year experienced expert programmers (level 3) who were paid $6 per hour using FORTRAN or C. The task requirement involved the determination, using dynamic programming, of an optimal sequence of machines to employ in providing service for a number of years. The development started with a manual exercise for problem understanding and for designing the program to determine appropriate methods. It then proceeded with developing the program to implement the algorithm. During programming, the programmer's task behavior was observed by a supervisor, who was the project navigator, to measure the three factors of common-cause error modes in human-software interaction processing systems discussed previously.

Experimental period: Key experiment: 3/6, 1991 - 3/28, 1991

Subjects: (13 subjects: 5-8 years experienced experts)

Group A: 7 experts (FORTRAN)

Group B: 6 experts (C)

Experimental design tools:

Subject programming bases: FORTRAN or C (180-230 lines)

Data analysis: SAS, LOTUS-123

Subject selection and training Subjects were recruited using public advertisements. Their life data was gathered during an individual interview. They were educated and trained to the exact requirements and procedures of the experiment. They were also evaluated with respect to programming experience and knowledge background as objective data, and intelligence to problem solving, experiment attitude, and environmental conditions during the experiment as a subjective data for the subject calibration. The contents of this experimental phase are as follows:

(1) Screening and selecting of subjects

(2) Subjects life data collection

(3) Initialization session:

Problem definition, manual solving and mathematical validation

Requirements of specification

Procedures used in the experiment and data collection

(4) Training session:

Programming requirements

Hardware and operating systems

Data gathering and presentation

(5) Consultation session

Hardware and operating systems

Programming language

Programming requirement understanding

Common-cause error modes

(6) Experiment attitude with monitoring log-on time

(7) Subject calibration and evaluation factors: Subjects can be evaluated according to five categories for comparison regarding their task performance. The rating weights for these five factors are determined from interviews with an expert programmer.

Programming experience:

Programming experience (years)

Recurrence of programming (months)

Project scale involved (lines)

Knowledge background:

Knowledge of programming language

Familiarity with hardware

Familiarity with operating system

Educational background of requirement

Intelligence:

Problem solving ability

Creativity of entire approach

Requirement understandability

Recognition of project process

Experimental attitude:

Concentration to task

Commitment to regulation

Preparation effort to task

Conditions in the work environment:

Entire condition of work station

Noise, temperature, humidity, etc.

Subject physical conditions

Extra mental, psychological stress

Main experiment and data collection The main experiment was then carried out and common-cause error data gathered with the following conditions:

Experimental Laboratory: human-software interaction laboratory

Hardware setting: work station: DECstation 2100

Operating setting: VINCENT: ULTRIX

Representation interview session in each 30-45 minutes

Supervisor monitoring using simultaneous logging terminal

(1) Project 4 (5-8 year experienced expert programmers using FORTRAN or C): Project 4 was carried out using 5-8 year experienced expert programmers who were paid $6/hr with FORTRAN or C. Two task requirements involved the determination of an optimal sequence of machines to employ in providing service for a number of years using dynamic programming or determination of an optimal inventory policy using simulation. Three educational sessions were employed: an initial session, a training session, and a consultation session. The program development started with a manual exercise used for problem understanding and for designing the program to determine appropriate methods. This was followed by writing of the program to implement the algorithm. During programming, the programmer's task behavior

was observed by a supervisor, the project navigator, using a second terminal to measure error frequency, error correction time, point of occurrence in time for the three factored common-cause error modes discussed previously.

Experimental period: Key experiment: 10/4, 1991 - 11/7, 1991

Subjects: (10 subjects: 5-8 years experienced experts)

Subject level: Expert 1, Expert 2

Group A: 5 experts (:FORTRAN)

Group B: 5 experts (:C)

Requirement Specifications:

Optimal machine replacement - using dynamic programming

Optimal inventory policy and system simulation

Experimental design tools:

Subject programming bases: FORTRAN or C (300-400 lines)

Data analysis: SAS, LOTUS-123

• Common-cause data collections: The supervisor(project experimenter) recorded the contents of subject common-cause error including correction time and time of occurrence on the data collection sheet. During the data collection session, all of subject task and behavior were monitored and checked by the supervisor using a parallel simultaneous logging terminal, and these monitoring properties were taped in the video tape recorder. There were two different categories of collected data as follows:

(1) Objective data collection: contents of error, frequency, correction time, and point of occurrence in time;

(2) Subjective data collection: reason of common-cause error, identification of common-

cause error mode, pattern recognition of common-cause error mode, behavior domain of common-cause error mode, and evaluation factors of subject task performance.

• Representation of common-cause error modes: At the representation interview session held every 30-45 minutes, common-cause error protocol as the actual location of human error was associated with the contents of common-cause error. Common-cause error modes including their identification, pattern recognition, and behavior error domains were derived from the subject recognition of reasons for error and the supervisor's objective representational analysis together according to the review of recorded video tape.

interaction. The human error control mechanisms and prevention was viewed in the aspects of knowledge-based engineering approach, human-software information processing system, and human factors orientation. The major results are applied to intelligent design, knowledge based system, human-software interaction, and behavior domain model.

CHAPTER 5. COMMON-CAUSE ANALYSIS AND RESULT REPRESENTATION

Experimental data representing common-cause error in human-software interaction can be analyzed using statistical methods and geometrical modeling. Results enable one to define common-cause domain based on human error and to represent the common-cause error control mechanism. Then, statistically collected data are analyzed for the evaluation of the subject task, the statistical contents of the common-cause error experimental data, and their representational characteristics.

Analysis of Subject Task Data

In the pilot projects including Project 2 and Project 3, tasks were conducted by 26 subjects, averaging 22.9 years in age, 4.3 years of programming experience, and a typing speed of 4.7 pages per hour. The result was an average total frequency of error occurrence of 21.3, 182.5 minutes total error correction time during 438.7 minutes of total computing time per each version of software development. Using correlation analysis, which measures the strength of the linear relationship between two variables, the Pearson correlation coefficients of programming experience were -0.05419 for total frequency, -0.48282 for total time spent compute, and -0.42207 for total correction time. It means that more experienced programmer has less error and better performance. The analysis revealed that programming experience comparing the two levels, intermediate and expert, had a significant effect to the programmers' performance.

In the main experiment, two different tests (five subjects using dynamic programming and five using the inventory control system) were conducted using two respective languages (C and FORTRAN) with subjects averaging 24 years in age, 5.8 years of programming experience, and a typing skill of 4.4 pages per hour as shown in Table 5.1. Results consisted of a 32.1 average (9.4 standard deviation) total common-cause error frequency, and 255.3 minutes average total error correction time during 523 minutes total computing time per each version of software development. Time spent in understanding and problem solving was 109 minutes, and design time for programming was 170 minutes.

Table 5.2 was developed from interviews with the subject programming experts concerning the experiment in human-software interaction. The purpose of these interviews was to establish weight rating factors for subject evaluation in the programming experiment. In the five categories of subject evaluation factors, average rating from experts' responses are (a) programming experience (23%), (b) knowledge background (21%), (c) intelligence (23%), (d) experiment attitude (18%), (e) work environmental conditions (15%).

Subject evaluation factors (a) and (b) are evaluated by objective interview data, and factors (c), (d), and (e) are evaluated by subjective grading by experiment supervisor for aver all subject task performance during the experiment. Table 5.3 shows the subject overall performance score applied with rating factors to evaluate expert level during the programming experiment. As a result, with average evaluation score

Table 5.1: Subject Task Data in A Common-Cause Model Experiment[a]

S-ID[b]	Reqt[c]	Exp[d]	Freq[e]	Ct-T[f]	Com-T[g]	Sol-T[h]	Des-T[i]	Tot-T[j]
P4C01	P4-B	5	24.0	256.0	452	120	300	872
P4C02	P4-B	7	27.0	242.5	494	60	90	644
P4C03	P4-A	8	25.0	119.5	256	60	60	376
P4C04	P4-A	5	32.0	432.5	781	180	270	1231
P4C05	P4-A	5	52.0	487.5	886	180	210	1276
P4F06	P4-B	5	38.0	312.5	535	120	240	955
P4F07	P4-B	6	43.0	123.0	482	70	50	602
P4F08	P4-B	6	29.0	158.0	403	60	60	523
P4F09	P4-A	6	23.0	144.5	255	120	120	495
P4F10	P4-A	5	28.0	277.0	686	120	240	1046
MEAN:	.	5.8	32.1	255.3	523	109	170	802
S.D.:	.	1.0	9.4	127.8	208.2	46.3	104.2	318.9

[a]Expressed as time in min.
[b]S-ID: Subject Identification No.
[c]Reqt: type of requirement(A: dynamic programming, or B: inventory control).
[d]Exp: programming experience (years).
[e]Freq: frequency of common-cause error mode.
[f]Ct-T: correction time of error.
[g]Com-T: computing time of program.
[h]Sol-T: problem solving time.
[i]Des-T: program design time.
[j]Tot-T: total spent time.

Table 5.2: Weight Rating Factors for Subject Evaluation: Interview Search from Programming Experts

I[a]	a1[b]	a3	b1	b2	b3	b4	c1	c2	c3	c4	d1	d2	d3	e1	e2	e3	e4	
A	5	.	.	4	.	.	.	4	.	.	.	2	.	.	1	.	.	.
.	3	2	2	3	1	1	2	3	3	3	3	3	2	3	1	2	2	3
B	3	.	.	5	.	.	.	5	.	.	.	3	.	.	3	.	.	.
.	2	2	2	3	1	2	3	3	2	3	2	2	2	2	1	1	2	3
C	4	.	.	3	.	.	.	5	.	.	.	3	.	.	3	.	.	.
.	3	3	2	3	1	2	3	3	2	3	2	3	2	2	1	2	3	3
D	4	.	.	3	.	.	.	5	.	.	.	2	.	.	2	.	.	.
.	3	3	2	2	2	2	1	3	2	2	2	3	2	2	1	2	1	3
E	5	.	.	5	.	.	.	4	.	.	.	4	.	.	4	.	.	.
.	3	3	2	3	2	3	3	3	1	3	2	3	1	2	3	2	3	3
F	5	.	.	3	.	.	.	5	.	.	.	3	.	.	2	.	.	.
.	2	3	3	3	1	2	1	3	3	3	1	2	3	3	1	1	2	2
G	4	.	.	5	.	.	.	3	.	.	.	3	.	.	3	.	.	.
.	3	3	3	3	2	2	2	3	3	3	3	3	2	3	3	3	3	3
H	5	.	.	4	.	.	.	5	.	.	.	5	.	.	4	.	.	.
.	3	3	1	3	2	2	2	3	1	3	2	3	3	2	3	3	2	3
I	5	.	.	4	.	.	.	4	.	.	.	4	.	.	2	.	.	.
.	3	3	1	3	2	2	2	3	2	3	2	3	2	3	2	1	2	3
J	4	.	.	3	.	.	.	3	.	.	.	4	.	.	5	.	.	.
.	2	3	2	3	1	2	1	3	2	3	2	3	3	3	3	3	3	3
T[c]	44	.	.	39	.	.	.	43	.	.	.	33	.	.	29	.	.	.
.	27	28	20	29	15	20	20	30	21	29	21	28	22	25	20	20	23	29
R[d]	23	.	.	21	.	.	.	23	.	.	.	18	.	.	15	.	.	.
.	36	37	27	34	18	24	24	29	21	29	21	37	30	33	22	22	25	31

[a] I: programming expert interview.

[b] a,b,c,d,e: evaluation factors for expert level; a1,a2,a3: evaluation subfactors for a.

[c] T: total score(upper: score for a, b, c, d, e; lower: score for subfactors a1, a2, a3).

[d] R: rating percentage for subject evaluation.

of 4.3 and standard deviation of 0.5, a expert level 1 group included C03 (4.9), F09 (4.8), C02 (4.8), F06 (4.5), and F08 (4.3), and a expert level 2 group included F07 (4.2), C01 (4.1), C04 (3.8), C05 (3.6), and F10 (3.5).

Common-Cause Mode-Oriented Data Statistics

Experimentally collected data were analyzed using statistical methods and geometrical configurations to define the common-cause error reasons and to represent the error mechanism.

Common-cause error mode data and analysis table

Common-cause error modes are shown in Table 5.4 for the three factors I_i, P_j, B_k with three evaluating variables, frequency ($F_{i,j,k}$), correction time ($C_{i,j,k}$), and point of occurrence time ($O_{i,j,k}$) in each common cause error mode. Figures 5.1 5.2 5.3 show the portion of common-cause errors in each of the three error modes indicating their contents in terms of identification, pattern recognition, and behavior domain in human-software interaction. With the error occurrence frequency factor, the major reasoning categories in each common-cause error mode are: in the identification mode, I.3 (19.4%), I.2 (16.2%), and I.1 (15.9%); in the pattern recognition mode, P.2 (33.7%), P.3 (18.0%), and P.1 (15.7%); in the behavior domain mode, B.3 (43.6%) and B.2 (36.5%). When the error correction time factor is applied, I.1 (26.2%), I.5 (16.6%), and I.8 (13.9%) in the I_i mode; P.2 (44.8%) and P.1 (21.2%) in the P_j mode; and B.3 (62.7%) and B.2 (28.1%) in the B_k mode. On the aspect of error correction time per error frequency (CT/F), the major effort in each common-cause error mode resulted in I.1 (12.9 minutes/frequency), I.5 (11.6) and I.6 (10.1) in the

Table 5.3: Subject Level Evaluation with Rating Factors

S[a]	a1[b]	a2	a3	b1	b2	b3	b4	c1	c2	c3	c4	d1	d2	d3	e1	e2	e3	e4
Evl	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
1	3.6[c]	.	.	4.2	.	.	.	4.2	.	.	.	3.9	.	.	5.0	.	.	.
4.1[d]	4[e]	3	4	4	5	4	4	4	4	4	5	4	4	4	5	5	5	5
2	4.6	.	.	4.8	.	.	.	5.0	.	.	.	4.9	.	.	5.0	.	.	.
4.8	5	4	5	5	5	5	4	5	5	5	5	5	5	5	5	5	5	5
3	4.6	.	.	5.0	.	.	.	5.0	.	.	.	4.9	.	.	5.0	.	.	.
4.9	5	4	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5
4	3.9	.	.	3.8	.	.	.	3.4	.	.	.	3.2	.	.	5.0	.	.	.
3.8	4	3	5	4	4	4	3	3	4	3	4	3	4	3	5	5	5	5
5	3.7	.	.	3.4	.	.	.	3.2	.	.	.	3.3	.	.	5.0	.	.	.
3.6	4	4	3	3	4	4	3	3	3	3	4	4	3	3	5	5	5	5
6	4.2	.	.	5.0	.	.	.	4.0	.	.	.	4.6	.	.	5.0	.	.	.
4.5	4	4	5	5	5	5	5	4	4	4	4	5	5	4	5	5	5	5
7	4.6	.	.	4.8	.	.	.	4.0	.	.	.	2.9	.	.	5.0	.	.	.
4.2	4	5	5	5	5	5	4	4	4	4	4	3	3	3	5	5	5	5
8	3.6	.	.	4.4	.	.	.	4.8	.	.	.	3.9	.	.	5.0	.	.	.
4.3	4	3	4	4	5	4	5	5	4	5	5	4	4	4	5	5	5	5
9	4.3	.	.	5.0	.	.	.	4.8	.	.	.	4.9	.	.	5.0	.	.	.
4.8	4	5	4	5	5	5	5	5	5	5	4	5	5	5	5	5	5	5
10	3.3	.	.	3.4	.	.	.	3.2	.	.	.	2.9	.	.	5.0	.	.	.
3.5	4	3	3	3	4	4	3	3	3	3	4	3	3	3	5	5	5	5
Rat[f]	23	.	.	21	.	.	.	23	.	.	.	18	.	.	15	.	.	.
.	36	37	27	34	18	24	24	29	21	29	21	37	30	33	22	22	25	31
Avg	4.0	.	.	4.4	.	.	.	4.2	.	.	.	3.9	.	.	5.0	.	.	.
4.3	4.1	3.8	4.3	4.3	4.7	4.5	4.1	4.1	4.1	4.1	4.4	4.0	4.1	3.9	5	5	5	5
SD	0.5	.	.	0.6	.	.	.	0.7	.	.	.	0.8	.	.	0	.	.	.
0.5	0.4	0.7	0.7	0.7	0.4	0.5	0.8	0.7	0.7	0.8	0.4	0.8	0.8	0.8	0	0	0	0

[a]S: experiment subject identification no.

[b]a,b,c,d,e: evaluation factors for subject expert level; a1,a2,a3: evaluation subfactors for factor a.

[c]upper score: score for a,b,c,d,e.

[d]Evl score: final evaluation score for subject expertise level.

[e]lower score: score for subfactors a1,a2,a3.

[f]Rat: rating percentage for subject evaluation.

Table 5.4: Common-Cause Error Mode and Experimental Data Analysis: Total [a]

CCM[b]	Freq	$F_{i,j,k}$[c]	CT	$C_{i,j,k}$[d]	POT	$O_{i,j,k}$[e]	CT/F[f]
I.1	54	15.9%	697.0	26.2%	369.7	70.7%	12.9
I.2	55	16.2%	317.0	11.9%	217.7	41.6%	5.8
I.3	66	19.4%	224.0	8.4%	230.7	44.1%	3.4
I.4	30	8.8%	189.0	7.1%	260.0	50.9%	6.3
I.5	38	11.2%	442.0	16.6%	320.5	61.3%	11.6
I.6	35	10.3%	354.0	13.3%	235.4	45.0%	10.1
I.7	24	7.0%	72.0	2.7%	163.3	31.2%	3.0
I.8	38	11.2%	369.0	13.8%	359.2	68.7%	9.7
Tot	340	100.0%	2664.0	100.0%	523.0[g]	51.7%[h]	7.8[i]
P.1	55	15.7%	583.5	21.2%	254.1	48.6%	10.6
P.2	118	33.7%	1234.0	44.8%	292.4	55.9%	10.5
P.3	63	18.0%	168.5	6.1%	248.8	47.6%	2.7
P.4	17	4.9%	140.5	5.1%	250.5	47.9%	8.3
P.5	24	6.8%	83.0	3.0%	223.2	42.7%	3.5
P.6	9	2.6%	80.5	2.9%	238.8	45.7%	8.9
P.7	26	7.4%	216.0	7.9%	316.9	60.6%	8.3
P.8	22	6.3%	84.5	3.1%	207.7	39.7%	3.8
P.9	16	4.6%	163.0	5.9%	198.1	37.9%	10.2
Tot	350	100.0%	2753.5	100.0%	523.8	47.4%	7.9
B.1	55	16.3%	86.5	3.2%	246.5	47.1%	1.6
B.2	123	36.5%	754.0	28.1%	257.2	49.2%	6.1
B.3	147	43.6%	1681.5	62.7%	280.0	53.5%	11.4
B.4	12	3.6%	160.5	6.0%	289.2	55.3%	13.4
Tot	337	100.0%	2682.5	100.0%	523.0	51.3%	8.0

[a] I_i: identification of common-cause error mode, P_j: pattern recognition of common-cause error mode, B_k: behavior domain of common-cause error mode.

[b] CCM: common-cause error mode.

[c] $F_{i,j,k}$: portion(%) of total frequency.

[d] $C_{i,j,k}$: portion(%) of correction time.

[e] $O_{i,j,k}$: average point of occurrence in time from total 100 % completion time.

[f] CT/F: correction time per frequency(unit: time in min.).

[g] total 100% completion time.

[h] average percentage of $O_{i,j,k}$.

[i] average time of CT/F.

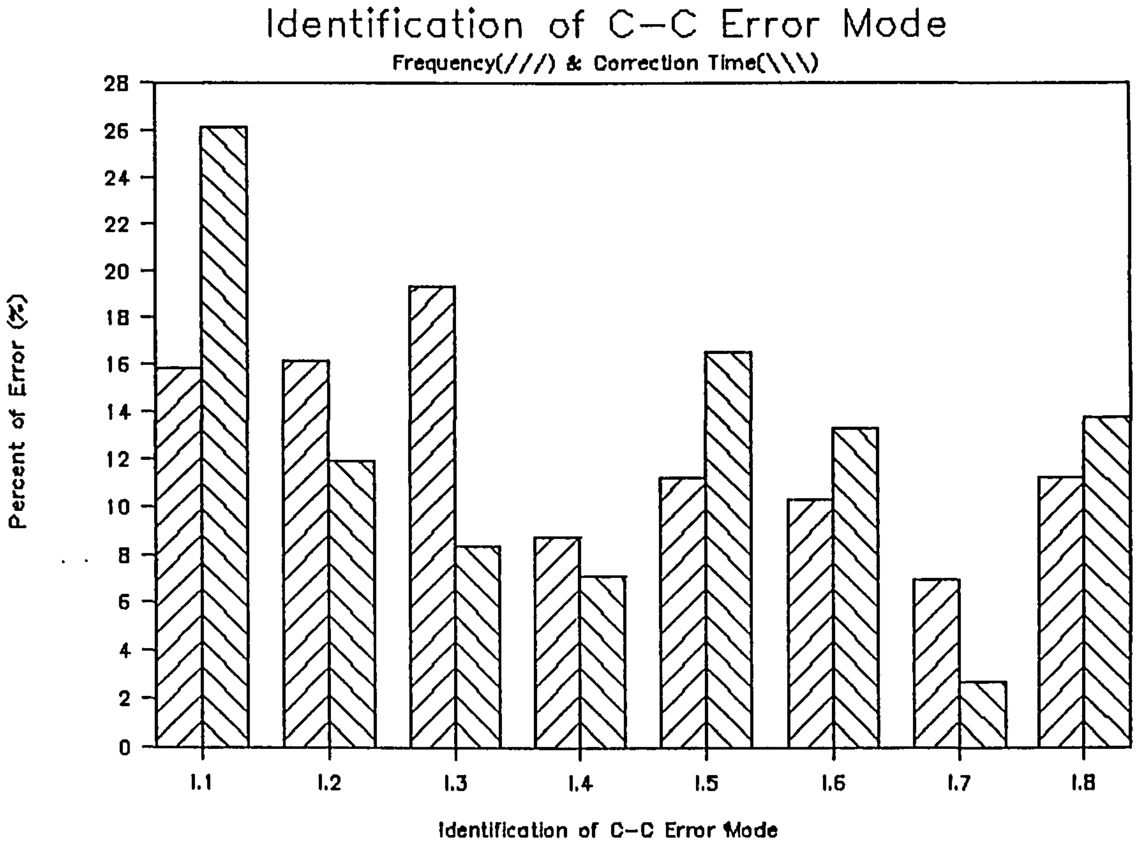


Figure 5.1: Portion of Identification of Common-Cause Error Mode

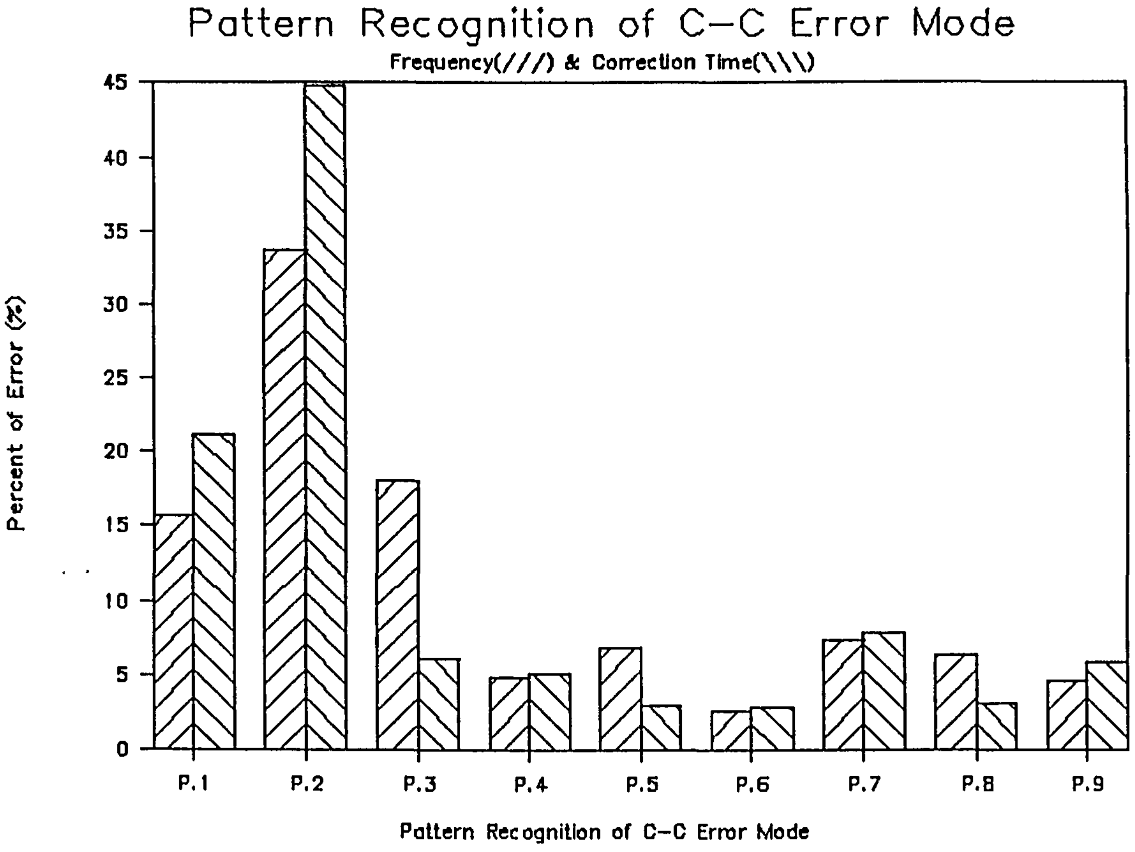


Figure 5.2: Portion of Pattern Recognition of Common-Cause Error Mode

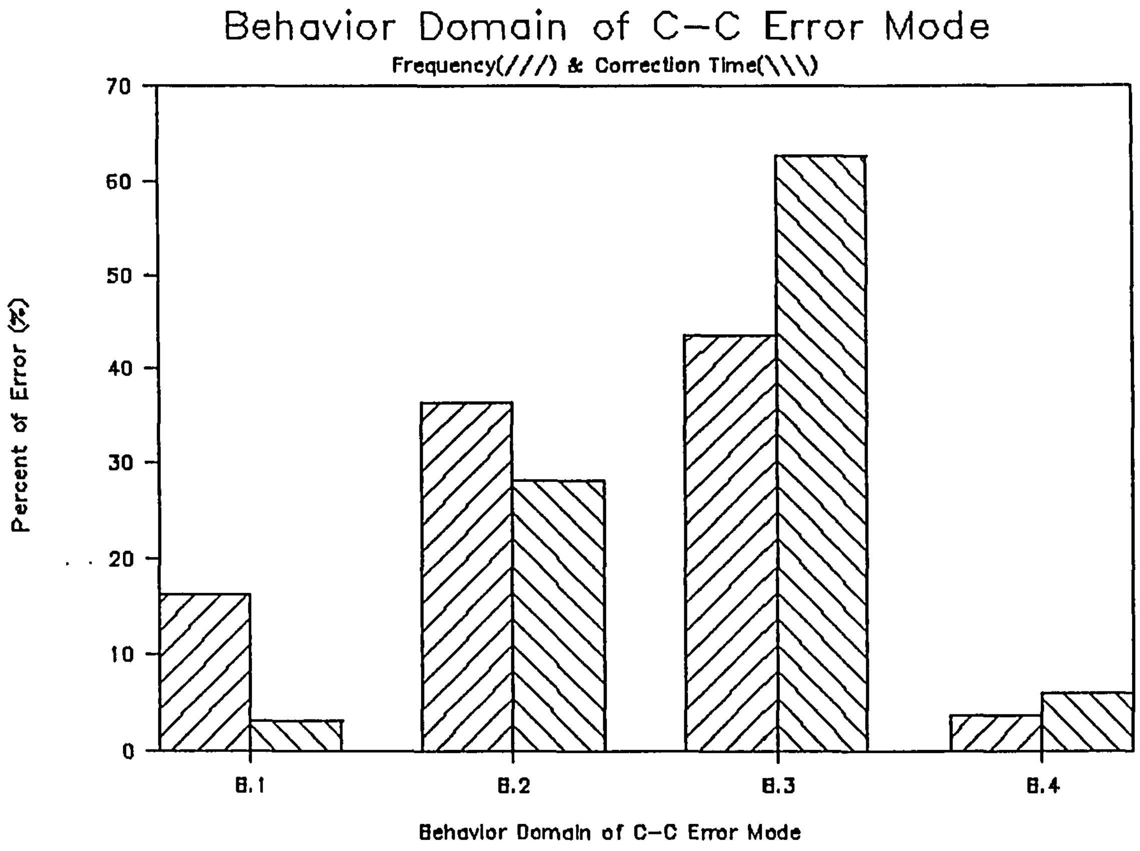


Figure 5.3: Portion of Behavior Domain of Common-Cause Error Mode

I_i mode; P.1 (10.6), P.2 (10.5), and P.9 (10.2) in the P_j mode; B.4 (13.4) and B.3 (11.4) in the B_k mode during the programming experiment.

Using a comparison of major reasoning modes with geometric vector evaluations between language C group and language Fortran group (Tables 5.5 and 5.6), the difference is I.5 and I.8 of second and third place in identification mode. P.6 and B.4 had bigger vector values in Fortran group. Using a comparison of major reasoning modes with geometric vector evaluations between requirement A and B (Tables 5.7 and 5.8), there was no difference on trend in the identification mode and in the behavior error domain mode. P.1 and P.7 were different rank of second and third place in pattern recognition. Using a comparison of major reasoning modes with geometric vector evaluations between programming expertise level 1 and level 2 (Tables 5.9 and 5.10), there was no difference on trend in the behavior error domain mode. I.4 and I.5 were different rank of third and fourth place in identification mode. P.5 had a bigger value of vector evaluation in expert level 2 as a better programming expertise subject group.

Figures 5.4, 5.5, and 5.6 show plots of proportional mean frequency based on six criteria for characteristics of each common-cause error mode. All trends are similar except for I.3 and I.4 in the common-cause identification mode. Figures 5.7, 5.8, and 5.9 show plots of proportional mean correction time based on six criteria for each common-cause error mode. There are no significant differences in pattern recognition and behavior domain. However, I.4 and I.5 in the identification mode have a little difference in correction time. Figures 5.10, 5.11, and 5.12 show plots of proportional mean occurrence time based on six criteria for each common-cause error mode.

Table 5.5: Common-Cause Error Mode - Data Analysis: Language-C [a]

CCM	Freq	$F_{i,j,k}$[b]	CT	$C_{i,j,k}$[c]	POT	$O_{i,j,k}$[d]	CT/F[e]	V(1:1:1)[f]
I.1	27	15.9%	461.5	28.7%	408.0	71.1%	17.1	78.3
I.2	30	17.6%	211.5	13.2%	228.0	39.7%	7.1	45.4
I.3	33	19.4%	113.0	7.0%	243.2	42.4%	3.4	47.2
I.4	8	4.7%	41.0	2.5%	255.2	44.5%	5.1	44.8
I.5	21	12.4%	306.0	19.0%	322.9	56.3%	14.6	60.7
I.6	18	10.6%	226.0	14.1%	267.1	46.6%	12.6	49.8
I.7	16	9.4%	33.0	2.1%	155.8	27.2%	2.1	28.8
I.8	17	10.0%	215.0	13.4%	415.6	72.4%	12.7	74.3
Tot	170	100.0%	1607.0	100.0%	573.8[g]	50.0%[h]	9.5[i]	53.7[j]
P.1	27	15.3%	323.5	18.9%	269.2	46.9%	12.0	52.9
P.2	64	36.4%	870.0	50.8%	303.4	52.9%	13.6	81.9
P.3	31	17.6%	76.0	4.4%	278.5	48.5%	2.5	51.8
P.4	12	6.8%	120.0	7.0%	270.4	47.1%	10.0	48.1
P.5	13	7.4%	58.0	3.4%	230.3	40.1%	4.5	41.0
P.6	5	2.9%	37.5	2.2%	173.1	30.2%	7.5	30.4
P.7	9	5.1%	105.0	6.1%	323.7	56.4%	11.7	57.0
P.8	6	3.4%	56.5	3.3%	224.8	39.2%	9.4	39.5
P.9	9	5.1%	66.0	3.9%	198.6	34.6%	7.3	35.2
Tot	176	100.0%	1712.5	100.0%	573.8	44.0%	9.7	48.6
B.1	26	15.6%	50.5	3.2%	257.9	44.9%	1.9	47.7
B.2	65	39.2%	457.0	29.2%	270.6	47.2%	7.0	67.9
B.3	74	44.6%	1035.0	66.2%	294.5	51.3%	14.0	94.9
B.4	1	0.6%	22.0	1.4%	179.0	31.2%	22.0	31.2
Tot	166	100.0%	1564.5	100.0%	573.8	43.7%	9.4	60.4

[a] I_i: identification of common-cause error mode, P_j: pattern recognition of common-cause error mode, B_k: behavior domain of common-cause error mode.

[b] $F_{i,j,k}$: portion(%) of total frequency.

[c] $C_{i,j,k}$: portion(%) of correction time.

[d] $O_{i,j,k}$: average point of occurrence in time from total 100 % completion time.

[e] CT/F: correction time per frequency(unit: time in min.).

[f] geometric vector evaluation value with 1:1:1 rating.

[g] total 100% completion time.

[h] average percentage of $O_{i,j,k}$.

[i] average time of CT/F.

[j] average vector value in V(1:1:1).

Table 5.6: Common-Cause Error Mode - Data Analysis: Language-Fortran [a]

CCM	Freq	$F_{i,j,k}$[b]	CT	$C_{i,j,k}$[c]	POT	$O_{i,j,k}$[d]	CT/F[e]	V(1:1:1)[f]
I.1	27	15.9%	235.5	22.3%	331.4	70.2%	8.7	75.3
I.2	25	14.7%	105.5	10.0%	207.3	43.9%	4.2	47.4
I.3	33	19.4%	111.0	10.5%	218.1	46.2%	3.4	51.2
I.4	22	12.9%	148.0	14.0%	274.7	58.2%	6.7	61.2
I.5	17	10.0%	136.0	12.9%	318.0	67.4%	8.0	69.3
I.6	17	10.0%	128.0	12.1%	203.7	43.1%	7.5	45.9
I.7	8	4.7%	39.0	3.7%	172.8	36.6%	4.9	37.1
I.8	21	12.4%	154.0	14.5%	302.7	64.1%	7.3	66.9
Tot	170	100.0%	1057.0	100.0%	472.2[g]	53.7%[h]	6.2[i]	56.8[j]
P.1	28	16.1%	260.0	25.0%	238.9	50.6%	9.3	58.7
P.2	54	31.0%	364.0	35.0%	281.3	59.6%	6.7	75.7
P.3	32	18.4%	92.5	8.9%	218.9	46.4%	2.9	50.6
P.4	5	2.9%	20.5	2.0%	225.8	47.8%	4.1	47.9
P.5	11	6.3%	25.0	2.4%	216.0	45.7%	2.3	46.2
P.6	4	2.3%	43.0	4.1%	282.5	59.8%	10.8	60.0
P.7	17	9.8%	111.0	10.6%	310.0	65.7%	6.5	67.2
P.8	16	9.2%	28.0	2.7%	200.8	42.5%	1.8	43.6
P.9	7	4.0%	97.0	9.3%	197.5	41.8%	13.9	43.0
Tot	174	100.0%	1041.0	100.0%	472.2	51.1%	6.0	54.8
B.1	29	17.0%	36.0	3.2%	235.2	49.8%	1.2	52.7
B.2	58	33.9%	297.0	26.6%	243.7	51.6%	5.1	67.2
B.3	73	42.7%	646.5	57.8%	265.4	56.2%	8.9	91.3
B.4	11	6.4%	138.5	12.4%	316.8	67.1%	12.6	68.5
Tot	171	100.0%	1118.0	100.0%	472.2	56.2%	6.5	69.9

[a] I_i: identification of common-cause error mode, P_j: pattern recognition of common-cause error mode, B_k: behavior domain of common-cause error mode.

[b] $F_{i,j,k}$: portion(%) of total frequency.

[c] $C_{i,j,k}$: portion(%) of correction time.

[d] $O_{i,j,k}$: average point of occurrence in time from total 100 % completion time.

[e] CT/F: correction time per frequency(unit: time in min.).

[f] geometric vector evaluation value with 1:1:1 rating.

[g] total 100% completion time.

[h] average percentage of $O_{i,j,k}$.

[i] average time of CT/F.

[j] average vector value in V(1:1:1).

Table 5.7: Common-Cause Error Mode - Data Analysis: Requirement-A [a]

CCM	Freq	$F_{i,j,k}$[b]	CT	$C_{i,j,k}$[c]	POT	$O_{i,j,k}$[d]	CT/F[e]	V(1:1:1)[f]
I.1	25	15.1%	370.5	23.8%	426.9	74.5%	14.8	79.7
I.2	29	17.5%	238.5	15.3%	214.8	37.5%	8.2	44.1
I.3	39	23.5%	113.0	7.3%	238.9	41.7%	2.9	48.4
I.4	11	6.6%	73.0	4.7%	276.2	48.2%	6.6	48.9
I.5	19	11.4%	291.0	18.7%	343.3	59.9%	15.3	63.8
I.6	19	11.4%	244.0	15.7%	198.2	34.6%	12.8	39.7
I.7	10	6.0%	16.0	1.0%	263.4	46.0%	1.6	46.4
I.8	14	8.5%	210.0	13.5%	446.3	77.9%	15.0	79.5
Tot	166	100.0%	1556.0	100.0%	572.8[g]	52.6%[h]	9.4[i]	56.3[j]
P.1	24	14.1%	275.0	17.5%	259.3	45.3%	11.5	50.6
P.2	65	38.2%	809.5	51.5%	310.6	54.2%	12.5	84.0
P.3	31	18.2%	76.5	4.9%	253.9	44.3%	2.5	48.2
P.4	10	5.9%	86.5	5.5%	238.7	41.7%	8.7	42.4
P.5	12	7.1%	49.5	3.1%	175.8	30.7%	4.1	31.7
P.6	3	1.8%	35.5	2.3%	247.5	43.2%	11.8	43.3
P.7	7	4.1%	105.0	6.7%	352.5	61.5%	15.0	62.0
P.8	9	5.3%	60.5	3.8%	239.4	41.8%	6.7	42.3
P.9	9	5.3%	74.0	4.7%	213.1	37.2%	8.2	37.9
Tot	170	100.0%	1572.0	100.0%	572.8	44.4%	9.3	49.2
B.1	27	16.5%	46.0	3.0%	261.4	45.6%	1.7	48.6
B.2	65	39.6%	474.0	30.6%	264.3	46.1%	7.3	68.1
B.3	69	42.1%	965.0	62.3%	303.6	53.0%	14.0	92.0
B.4	3	1.8%	64.0	4.1%	327.0	57.1%	9.5	57.3
Tot	164	100.0%	1549.0	100.0%	572.8	50.5%	9.5	66.5

[a] I_i: identification of common-cause error mode, P_j: pattern recognition of common-cause error mode, B_k: behavior domain of common-cause error mode.

[b] $F_{i,j,k}$: portion(%) of total frequency.

[c] $C_{i,j,k}$: portion(%) of correction time.

[d] $O_{i,j,k}$: average point of occurrence in time from total 100 % completion time.

[e] CT/F: correction time per frequency(unit: time in min.).

[f] geometric vector evaluation value with 1:1:1 rating.

[g] total 100% completion time.

[h] average percentage of $O_{i,j,k}$.

[i] average time of CT/F.

[j] average vector value in V(1:1:1).

Table 5.8: Common-Cause Error Mode - Data Analysis: Requirement-B [a]

CCM	Freq	$F_{i,j,k}$[b]	CT	$C_{i,j,k}$[c]	POT	$O_{i,j,k}$[d]	CT/F[e]	V(1:1:1)[f]
I.1	29	16.7%	326.5	29.5%	312.5	66.0%	11.3	74.2
I.2	26	14.9%	78.5	7.1%	220.6	46.6%	3.0	49.5
I.3	27	15.5%	111.0	10.0%	222.4	47.0%	4.1	50.5
I.4	19	10.9%	116.0	10.5%	283.3	60.0%	6.1	61.7
I.5	19	10.9%	151.0	13.6%	296.5	62.7%	8.0	65.1
I.6	16	9.2%	110.0	9.9%	238.2	50.3%	6.9	52.1
I.7	14	8.1%	56.0	5.1%	172.9	36.5%	4.0	37.8
I.8	24	13.8%	159.0	14.3%	310.8	65.7%	6.6	68.6
Tot	174	100.0%	1108.0	100.0%	473.2[g]	54.3%[h]	6.4[i]	57.4[j]
P.1	31	17.2%	308.5	26.1%	248.8	52.6%	10.0	61.2
P.2	53	29.4%	424.5	35.9%	274.1	57.9%	8.0	74.3
P.3	32	17.8%	92.0	7.8%	243.5	51.5%	2.9	55.0
P.4	7	3.9%	54.0	4.6%	265.3	56.1%	7.7	56.4
P.5	12	6.7%	33.5	2.9%	270.5	57.2%	2.8	57.6
P.6	6	3.3%	45.0	3.8%	232.9	49.2%	7.5	49.5
P.7	19	10.6%	111.0	9.4%	281.1	59.4%	5.8	61.1
P.8	13	7.2%	24.0	2.0%	165.3	34.9%	1.9	35.7
P.9	7	3.9%	89.0	7.5%	178.1	37.6%	12.7	38.6
Tot	180	100.0%	1181.5	100.0%	473.2	50.7%	6.6	54.4
B.1	28	16.2%	40.5	3.6%	231.7	49.0%	1.5	51.7
B.2	58	33.5%	280.0	24.7%	250.1	52.8%	4.8	67.3
B.3	78	45.1%	716.5	63.2%	256.3	54.2%	9.2	94.7
B.4	9	5.2%	96.5	8.5%	232.6	49.1%	10.7	50.2
Tot	173	100.0%	1133.5	100.0%	473.2	51.3%	6.6	66.0

[a] I_i: identification of common-cause error mode, P_j: pattern recognition of common-cause error mode, B_k: behavior domain of common-cause error mode.

[b] $F_{i,j,k}$: portion(%) of total frequency.

[c] $C_{i,j,k}$: portion(%) of correction time.

[d] $O_{i,j,k}$: average point of occurrence in time from total 100 % completion time.

[e] CT/F: correction time per frequency(unit: time in min.).

[f] geometric vector evaluation value with 1:1:1 rating.

[g] total 100% completion time.

[h] average percentage of $O_{i,j,k}$.

[i] average time of CT/F.

[j] average vector value in V(1:1:1).

Table 5.9: Common-Cause Error Mode - Data Analysis: Expert Level 1 [a]

CCM	Freq	$F_{i,j,k}$[b]	CT	$C_{i,j,k}$[c]	POT	$O_{i,j,k}$[d]	CT/F[e]	V(1:1:1)[f]
I.1	27	14.5%	403.0	24.4%	469.7	71.5%	14.9	76.9
I.2	29	15.6%	221.0	13.4%	272.7	41.5%	7.6	46.3
I.3	46	24.7%	146.5	8.9%	299.0	45.5%	3.2	52.5
I.4	11	5.9%	54.0	3.3%	301.0	45.8%	4.9	46.3
I.5	22	11.8%	367.0	22.2%	422.6	64.3%	16.7	69.0
I.6	21	11.3%	243.0	14.7%	292.2	44.5%	11.6	48.2
I.7	10	5.4%	17.5	1.1%	206.9	31.5%	1.8	32.0
I.8	20	10.8%	199.0	12.0%	468.7	71.3%	10.0	73.1
Tot	186	100.0%	1651.0	100.0%	657.4[g]	52.0%[h]	8.9[i]	55.5[j]
P.1	33	17.4%	368.0	21.2%	313.5	47.7%	11.1	55.0
P.2	72	37.9%	866.0	49.8%	364.1	55.4%	12.0	83.6
P.3	35	18.4%	93.0	5.3%	302.2	46.0%	2.7	49.8
P.4	10	5.3%	101.0	5.8%	336.4	51.2%	10.1	51.8
P.5	10	5.3%	54.0	3.1%	219.2	33.3%	5.4	33.9
P.6	2	1.0%	31.0	1.8%	313.5	47.7%	15.5	47.7
P.7	9	4.7%	101.0	5.8%	434.1	66.0%	11.2	66.5
P.8	12	6.3%	69.0	4.0%	303.1	46.1%	5.8	46.7
P.9	7	3.7%	55.0	3.2%	268.6	40.9%	7.9	41.2
Tot	190	100.0%	1738.0	100.0%	657.4	48.3%	9.2	52.9
B.1	29	15.8%	54.5	3.3%	310.3	47.2%	1.9	49.9
B.2	70	38.3%	518.5	31.5%	325.4	49.5%	7.4	70.0
B.3	81	44.3%	1041.0	63.1%	349.2	53.1%	12.9	93.6
B.4	3	1.6%	35.0	2.1%	394.3	60.0%	11.7	60.0
Tot	183	100.0%	1649.0	100.0%	657.4	52.5%	9.0	68.4

[a]I_i: identification of common-cause error mode, P_j: pattern recognition of common-cause error mode, B_k: behavior domain of common-cause error mode.

[b]$F_{i,j,k}$: portion(%) of total frequency.

[c]$C_{i,j,k}$: portion(%) of correction time.

[d]$O_{i,j,k}$: average point of occurrence in time from total 100 % completion time.

[e]CT/F: correction time per frequency(unit: time in min.).

[f]geometric vector evaluation value with 1:1:1 rating.

[g]total 100% completion time.

[h]average percentage of $O_{i,j,k}$.

[i]average time of CT/F.

[j]average vector value in V(1:1:1).

Table 5.10: Common-Cause Error Mode - Data Analysis: Expert Level 2 [a]

CCM	Freq	$F_{i,j,k}$ [b]	CT	$C_{i,j,k}$ [c]	POT	$O_{i,j,k}$ [d]	CT/F [e]	V(1:1:1) [f]
I.1	27	17.5%	294.0	29.0%	269.7	69.4%	10.9	77.2
I.2	26	16.9%	96.0	9.5%	162.7	41.9%	3.7	46.1
I.3	20	13.0%	77.5	7.6%	162.3	41.8%	3.9	44.4
I.4	19	12.3%	135.0	13.3%	238.0	61.2%	7.1	63.9
I.5	16	10.4%	75.0	7.4%	218.3	56.2%	4.7	57.6
I.6	14	9.1%	111.0	11.0%	178.6	46.0%	7.9	48.1
I.7	14	9.1%	54.5	5.4%	128.5	33.1%	3.9	34.7
I.8	18	11.7%	170.0	16.8%	249.6	64.2%	9.4	67.4
Tot	154	100.0%	1013.0	100.0%	388.6 [g]	51.7% [h]	6.6 [i]	54.9 [j]
P.1	22	13.7%	215.5	21.2%	194.6	50.1%	9.8	56.1
P.2	46	28.7%	368.0	36.3%	220.6	56.8%	8.0	73.2
P.3	28	17.5%	75.5	7.4%	195.2	50.2%	2.7	53.7
P.4	7	4.4%	39.5	3.9%	143.2	36.9%	5.6	37.3
P.5	14	8.8%	29.0	2.9%	227.1	58.5%	2.1	59.2
P.6	7	4.4%	49.5	4.9%	188.9	48.6%	7.1	49.1
P.7	17	10.6%	115.0	11.3%	199.7	51.4%	6.8	53.7
P.8	10	6.3%	15.5	1.5%	136.1	35.0%	1.6	35.6
P.9	9	5.6%	108.0	10.6%	169.9	43.7%	12.0	45.3
Tot	160	100.0%	1015.5	100.0%	388.6	47.9%	6.4	51.5
B.1	26	16.9%	32.0	3.1%	182.7	47.0%	1.2	50.1
B.2	53	34.4%	235.5	22.8%	188.9	48.6%	4.4	63.8
B.3	66	42.9%	640.5	62.0%	210.7	54.2%	9.7	92.8
B.4	9	5.8%	125.5	12.1%	219.2	56.4%	13.9	58.0
Tot	154	100.0%	1033.5	100.0%	388.6	51.6%	6.7	66.2

[a] I_i: identification of common-cause error mode, P_j: pattern recognition of common-cause error mode, B_k: behavior domain of common-cause error mode.

[b] $F_{i,j,k}$: portion(%) of total frequency.

[c] $C_{i,j,k}$: portion(%) of correction time.

[d] $O_{i,j,k}$: average point of occurrence in time from total 100 % completion time.

[e] CT/F: correction time per frequency(unit: time in min.).

[f] geometric vector evaluation value with 1:1:1 rating.

[g] total 100% completion time.

[h] total 100% completion time.

[i] average time of CT/F.

[j] average vector value in V(1:1:1).

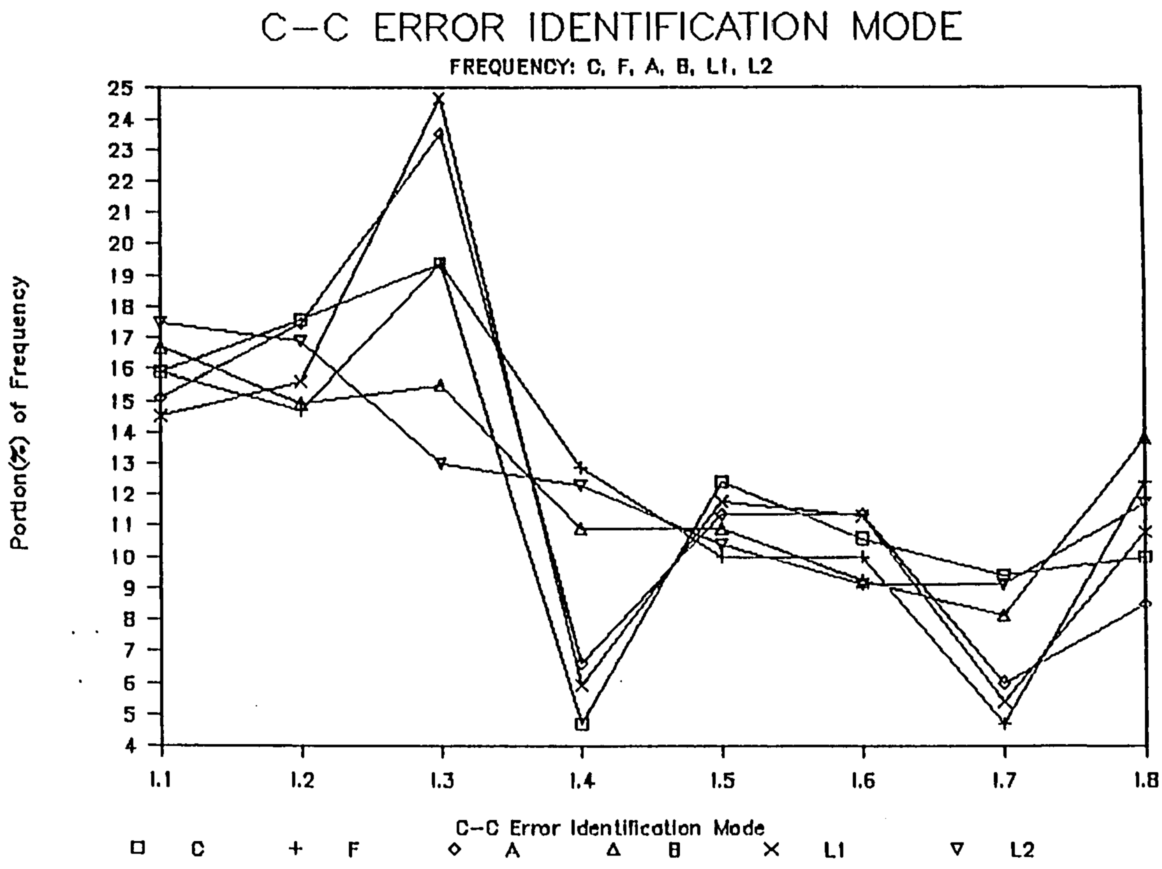


Figure 5.4: Portion of Frequency in Identification of Common-Cause Error Mode

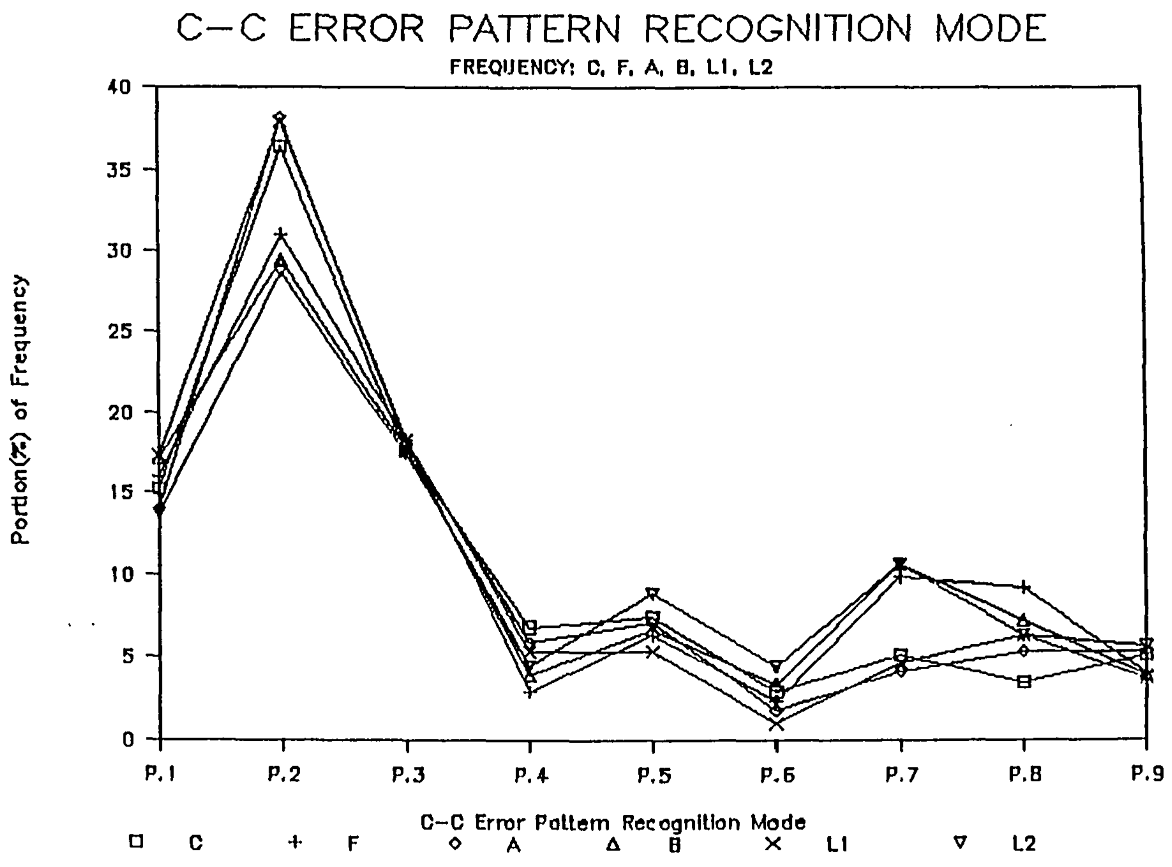


Figure 5.5: Portion of Frequency in Pattern Recognition of Common-Cause Error Mode

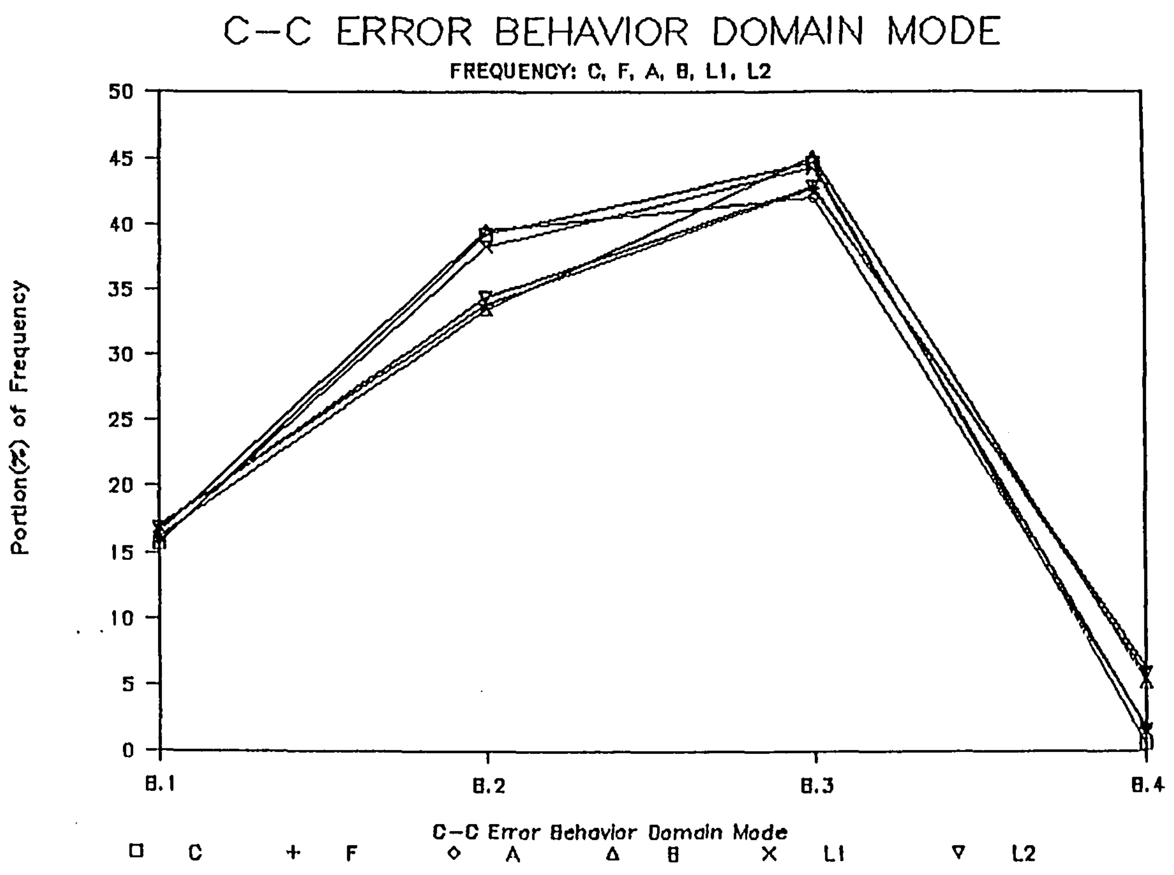


Figure 5.6: Portion of Frequency in Behavior Domain of Common-Cause Error Mode

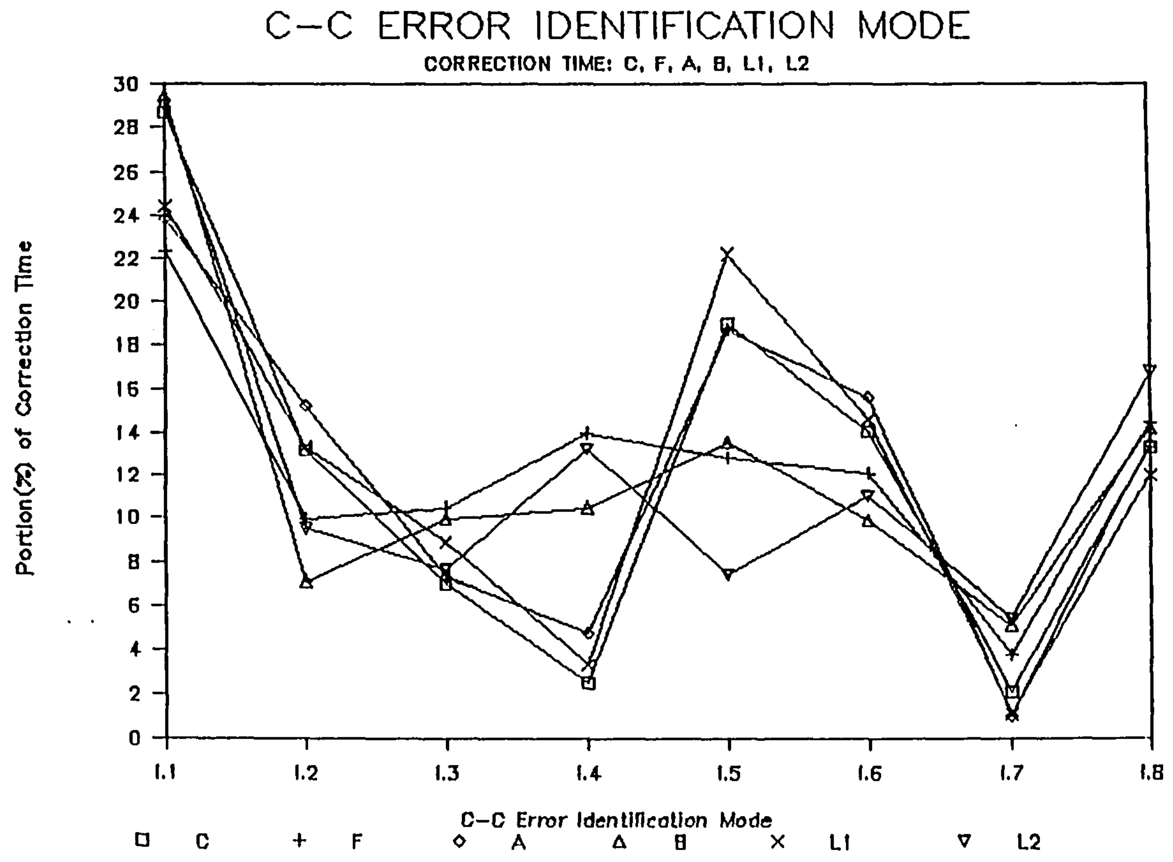


Figure 5.7: Portion of Correction Time in Identification of Common-Cause Error Mode

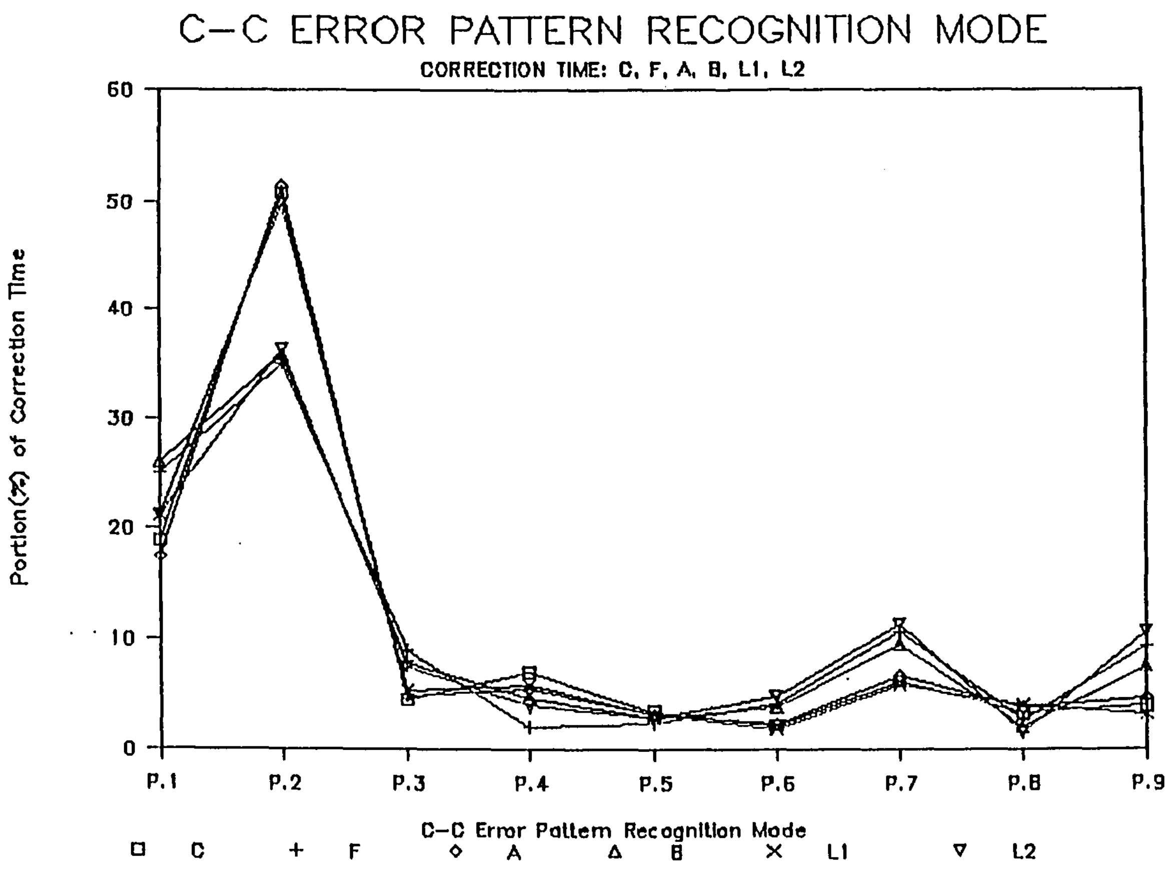


Figure 5.8: Portion of Correction Time in Pattern Recognition of Common-Cause Error Mode

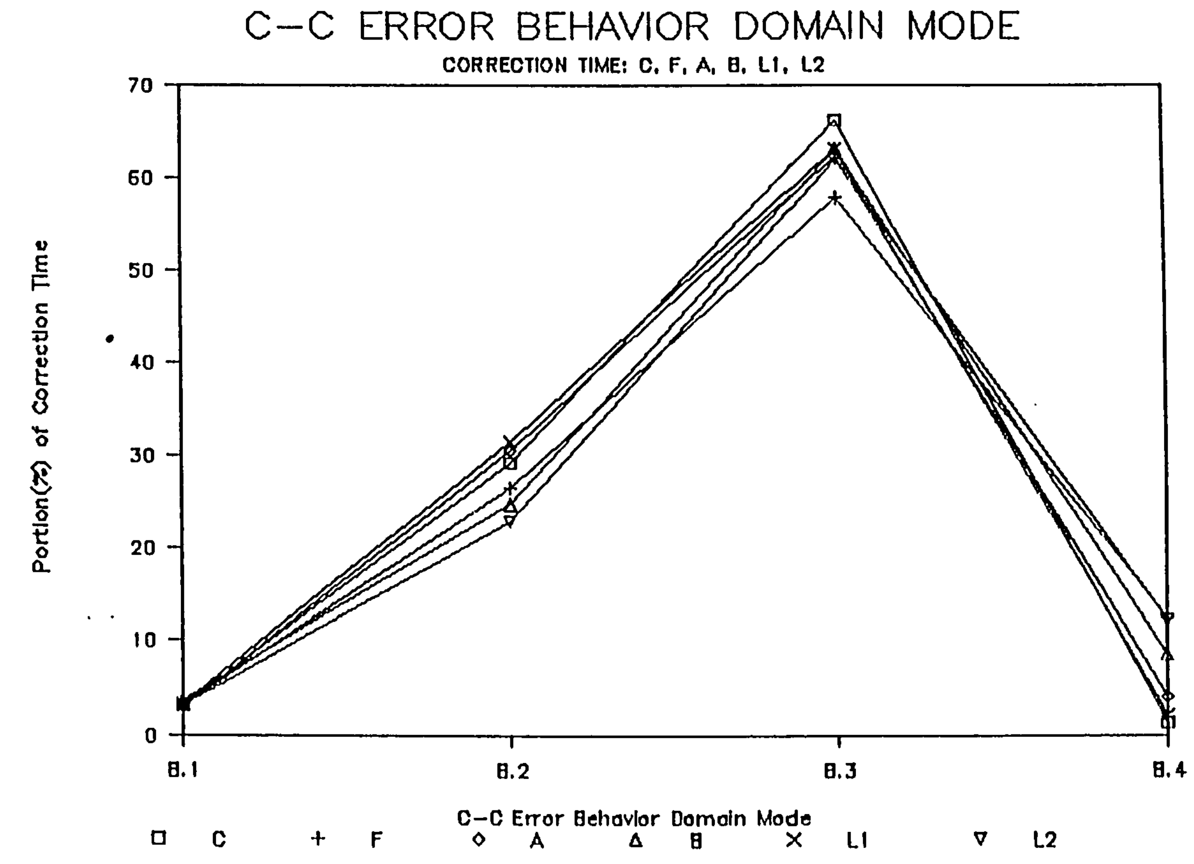


Figure 5.9: Portion of Correction Time in Behavior Domain of Common-Cause Error Mode

P.4, P.5 and P.6 in the pattern recognition mode and B.4 in behavior domain mode result in different relative proportions but the remaining common-cause error modes show a strong trend for comparison among the various proportional means.

Value of the common-cause function and simulated rating

Each value listed in the common-cause function parameters can be produced by three factors, $F_{i,j,k}$, $C_{i,j,k}$, $O_{i,j,k}$. For example, using a 1:1:1 weight rate simulation:

$$C((I_1(0.159, 0.262, 0.707), I_2(0.162, 0.119, 0.416), I_3(0.194, 0.084, 0.441),$$

$$I_4(0.088, 0.071, 0.509), I_5(0.112, 0.166, 0.613), I_6(0.103, 0.133, 0.450),$$

$$I_7(0.071, 0.027, 0.312), I_8(0.112, 0.139, 0.687)), (P_1(0.157, 0.212, 0.486),$$

$$P_2(0.337, 0.448, 0.559), P_3(0.180, 0.061, 0.476), P_4(0.049, 0.051, 0.479),$$

$$P_5(0.069, 0.030, 0.427), P_6(0.026, 0.029, 0.457), P_7(0.074, 0.078, 0.606),$$

$$P_8(0.063, 0.031, 0.397), P_9(0.046, 0.059, 0.379)), (B_1(0.163, 0.032, 0.471),$$

$$B_2(0.365, 0.281, 0.492), B_3(0.436, 0.627, 0.535), B_4(0.036, 0.060, 0.553))).$$

From these common-cause profiles, one can determine the major common-cause error mode in terms of error frequency, error correction time, and point of error occurrence in time. The common-cause function can be simulated with different weighting of variables' rating as in Table 5.11. The final evaluation value of these common-cause functions can be produced using the geometrical vector evaluation method.

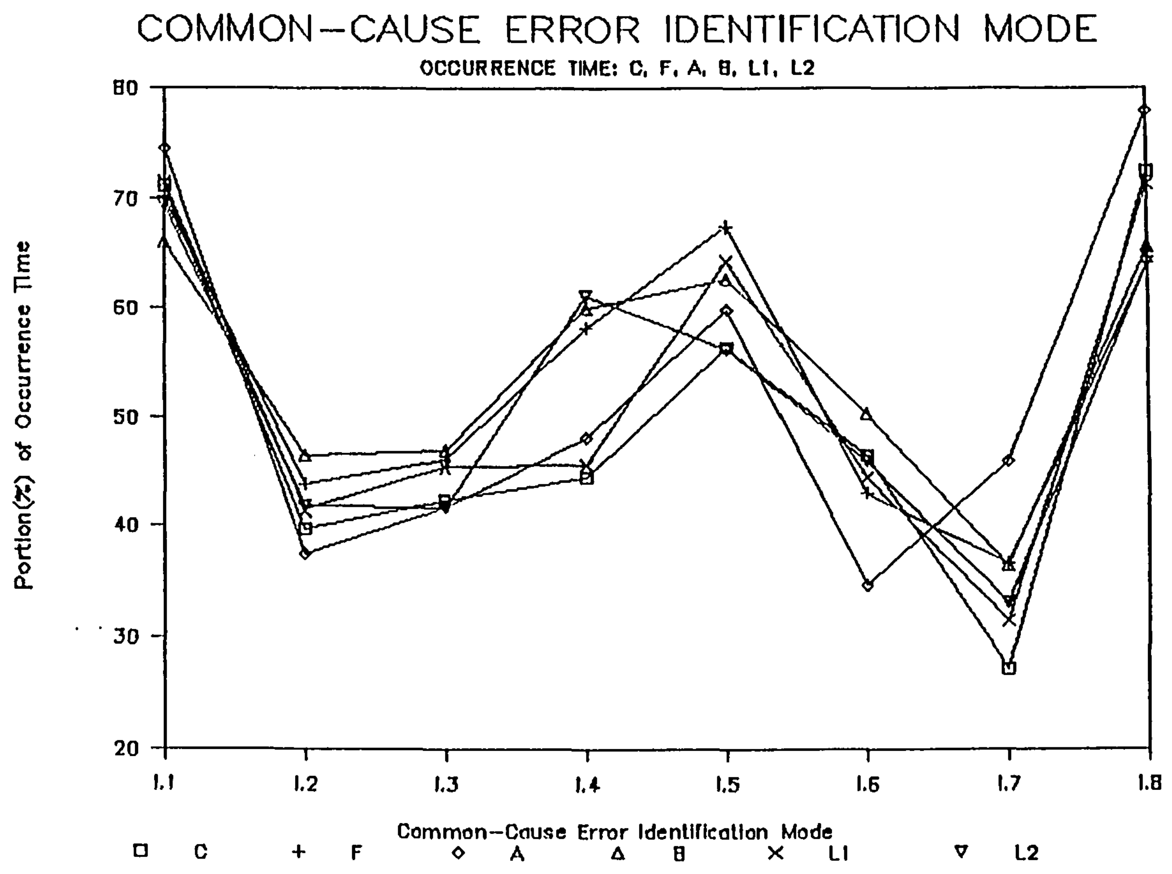


Figure 5.10: Portion of Occurrence Time in Identification of Common-Cause Error Mode

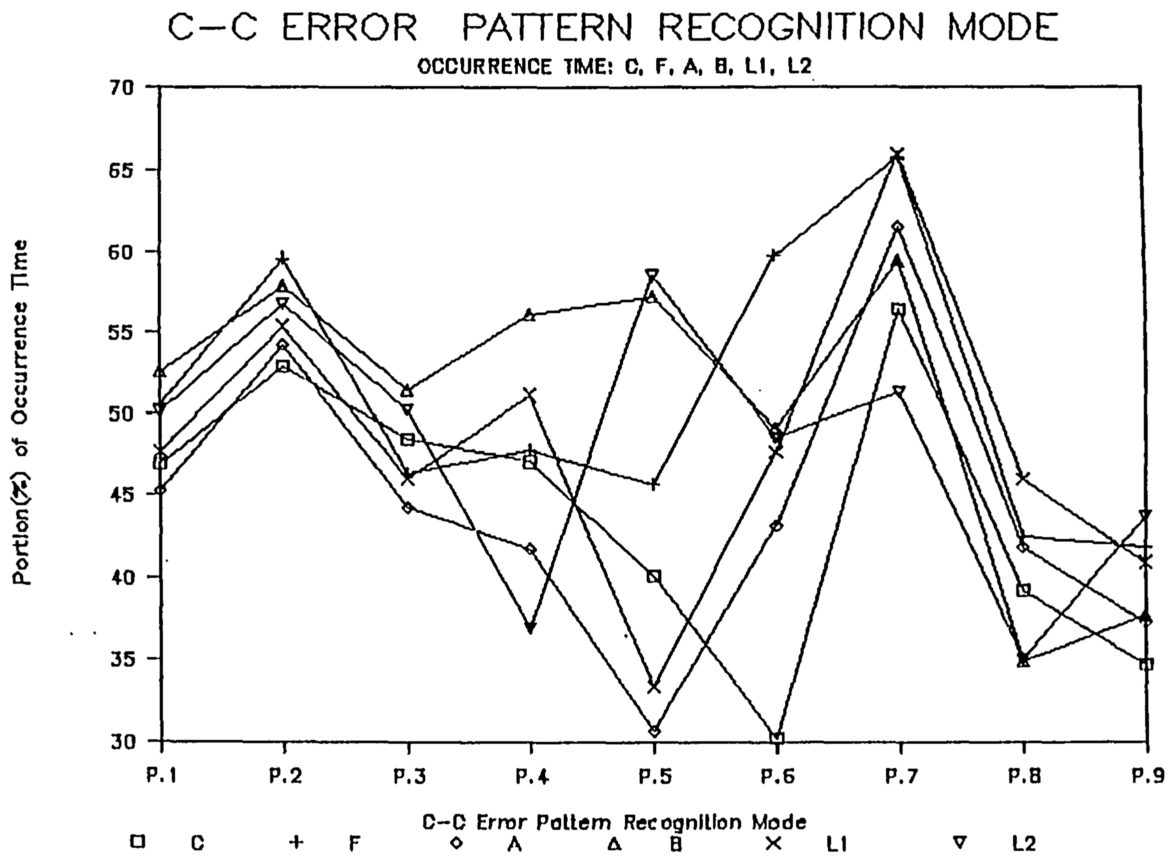


Figure 5.11: Portion of Occurrence Time in Pattern Recognition of Common-Cause Error Mode

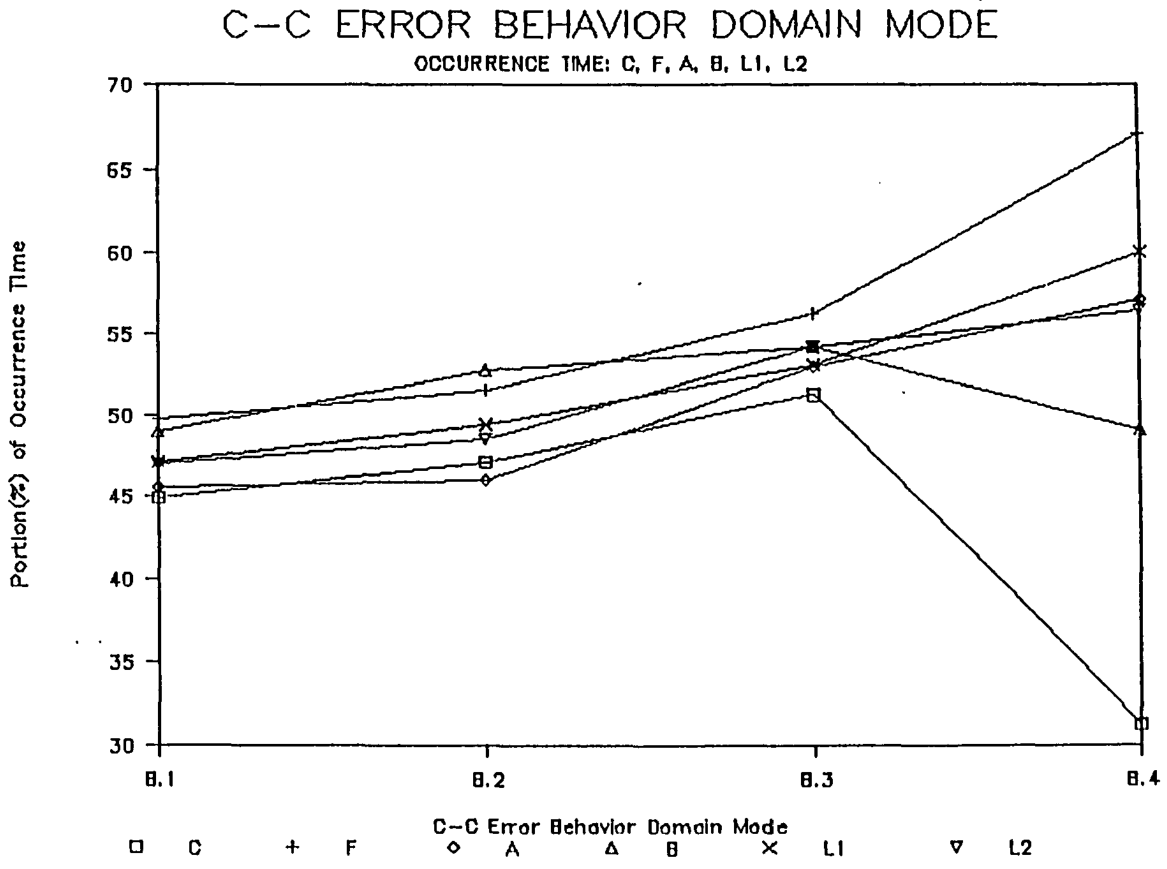


Figure 5.12: Portion of Occurrence Time in Behavior Domain of Common-Cause Error Mode

Table 5.11: Vector Evaluation with Rating Simulation [a]

CCM[b]	-	-	-	V1[c]	V2	V3	V4	V5	V6	V7	V8
-	F[d]	C[e]	O[f]	1:1:1	1:2:3	2:1:3	3:2:1	1:3:2	3:1:2	2:3:1	2:2:1
I.1	15.9	26.2	70.7	77.1	219.1	216.1	100.1	162.6	151.5	110.4	93.6
I.2	16.2	11.9	41.6	46.2	128.1	129.5	68.3	92.0	97.1	63.7	57.9
I.3	19.4	8.4	44.1	48.9	134.8	138.1	74.9	93.8	106.0	63.9	61.1
I.4	8.8	7.1	50.9	52.1	153.6	153.9	59.1	104.4	105.4	57.9	55.7
I.5	11.2	16.6	61.3	64.5	187.2	186.0	77.4	132.8	128.2	82.1	73.2
I.6	10.3	13.3	45.0	48.0	138.0	137.2	60.7	99.0	96.1	63.6	56.2
I.7	7.0	2.7	31.2	32.1	94.0	94.7	38.0	63.3	65.9	35.1	34.6
I.8	11.2	13.8	68.7	71.0	208.2	207.8	81.3	143.9	142.1	83.3	77.4
P.1	15.7	21.2	48.6	55.3	152.6	150.6	79.9	117.2	110.1	86.0	71.7
P.2	33.7	44.8	55.9	79.2	193.1	186.2	146.2	178.0	157.2	160.4	125.3
P.3	18.0	6.1	47.6	51.3	144.4	147.4	73.0	98.6	109.6	62.4	60.9
P.4	4.9	5.1	47.9	48.4	144.1	144.1	51.1	97.1	97.1	51.2	49.9
P.5	6.8	3.0	42.7	43.3	128.4	128.9	47.7	86.1	87.9	45.7	45.2
P.6	2.6	2.9	45.7	45.9	137.2	137.2	46.7	91.8	91.8	46.8	46.4
P.7	7.4	7.9	60.6	61.6	182.6	182.6	66.4	123.7	123.5	66.7	64.4
P.8	6.3	3.1	39.7	40.3	119.4	119.8	44.4	80.2	81.7	42.7	42.1
P.9	4.6	5.9	37.9	38.6	114.4	114.2	42.0	78.0	77.3	42.8	40.7
B.1	16.3	3.2	47.1	49.9	142.4	145.0	68.2	96.1	106.2	58.1	57.6
B.2	36.5	28.1	49.2	67.4	162.1	167.0	132.5	134.6	149.9	121.9	104.4
B.3	43.6	62.7	53.5	93.2	208.3	193.1	188.9	220.8	180.2	214.1	161.8
B.4	3.6	6.0	55.3	55.7	166.4	166.2	57.6	112.1	111.3	58.6	57.0

[a] I_i: identification of common-cause error mode, P_j: pattern recognition of common-cause error mode, B_k: behavior domain of common-cause error mode.

[b] CCM: common-cause error mode.

[c] V: rating weight for simulation.

[d] F: frequency of error occurrence(%).

[e] C: error correction time(%).

[f] O: point of error occurrence in time(% of final completion time).

Mapping geometrical vector evaluation in hexahedron contours

There are three configurations of hexahedron contours shown in Figures 5.13, 5.14, and 5.15, which present a combined severity profile of common-cause errors using each of the three factors of the common-cause function. Each common-cause mode can be evaluated by the calculation of a geometrical vector value from the geometric origin (F, C, O)=(0, 0, 0). Connections between major reasoning common-cause modes and minor reasoning modes can be recognized from this contour map. From the common-cause function $I_1(0.159, 0.262, 0.707)$, as given in Table 5.11, the vector evaluation with $\alpha : \beta : \gamma$=1:1:1 factors is derived as follows:

$$\sqrt{(1 \times 15.9^2 + 1 \times 26.2^2 + 1 \times 70.7^2)}$$
$$= \sqrt{(252.81 + 686.44 + 4998.49)}$$
$$= \sqrt{(5937.74)} = 77.1.$$

As an example of vector evaluation for the common-cause function with $\alpha : \beta : \gamma$=1:1:1 as weight ratings, using values from column V1 in Table 5.11, the overall common-cause function is,

$$C((I_1(77.1), I_2(46.2), I_3(48.9), I_4(52.1), I_5(64.5), I_6(48.0), I_7(32.1), I_8(71.0)),$$
$$(P_1(55.3), P_2(79.2), P_3(51.3), P_4(48.4), P_5(43.3), P_6(45.9), P_7(61.6), P_8(40.3), P_9(38.6)),$$
$$(B_1(49.9), B_2(67.4), B_3(93.2), B_4(55.7))).$$

By different emphasis or weighting on the evaluation factors, a different orientation stress for representing development cost or effort, frequency of error occurrence, error correction time, and point of error occurrence time, using evaluation by geometric

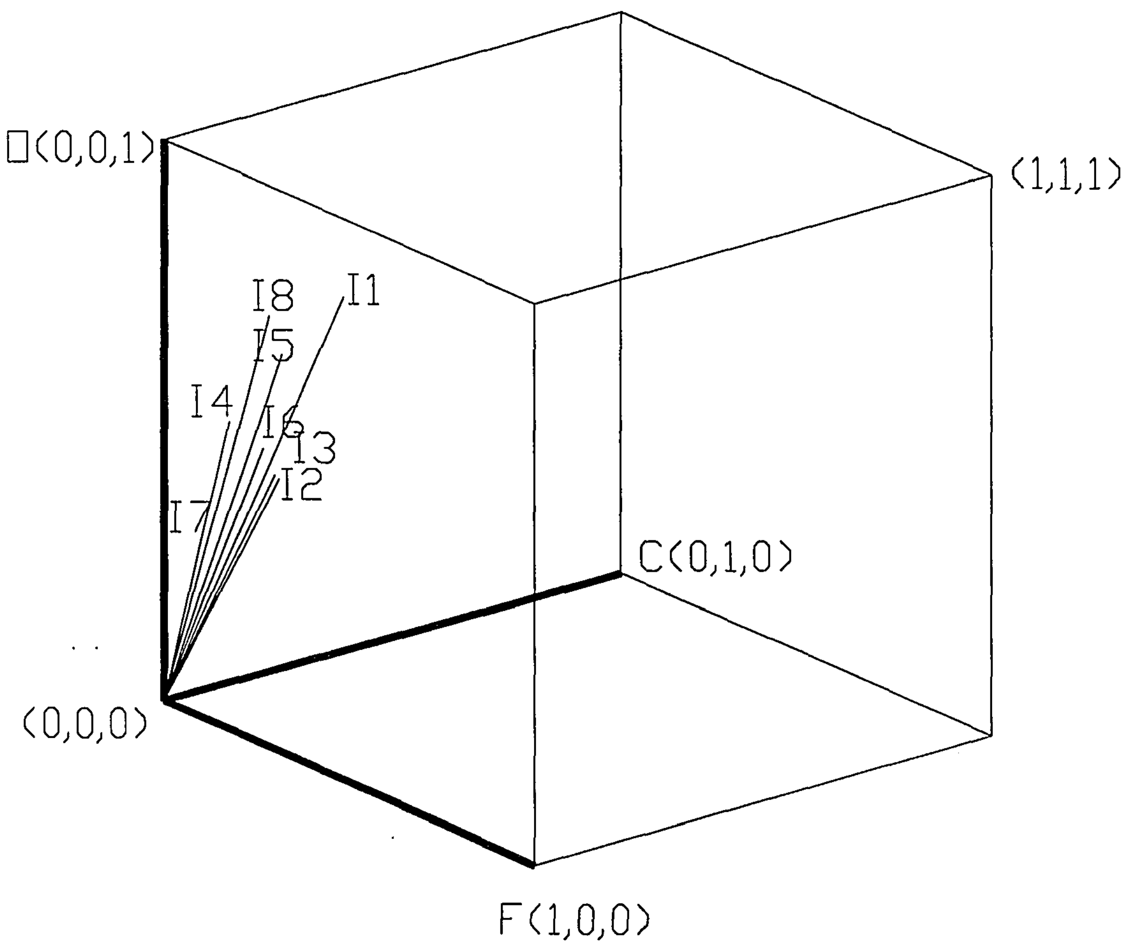


Figure 5.13: Identification of Common-Cause Error Mode: Geometric Configuration

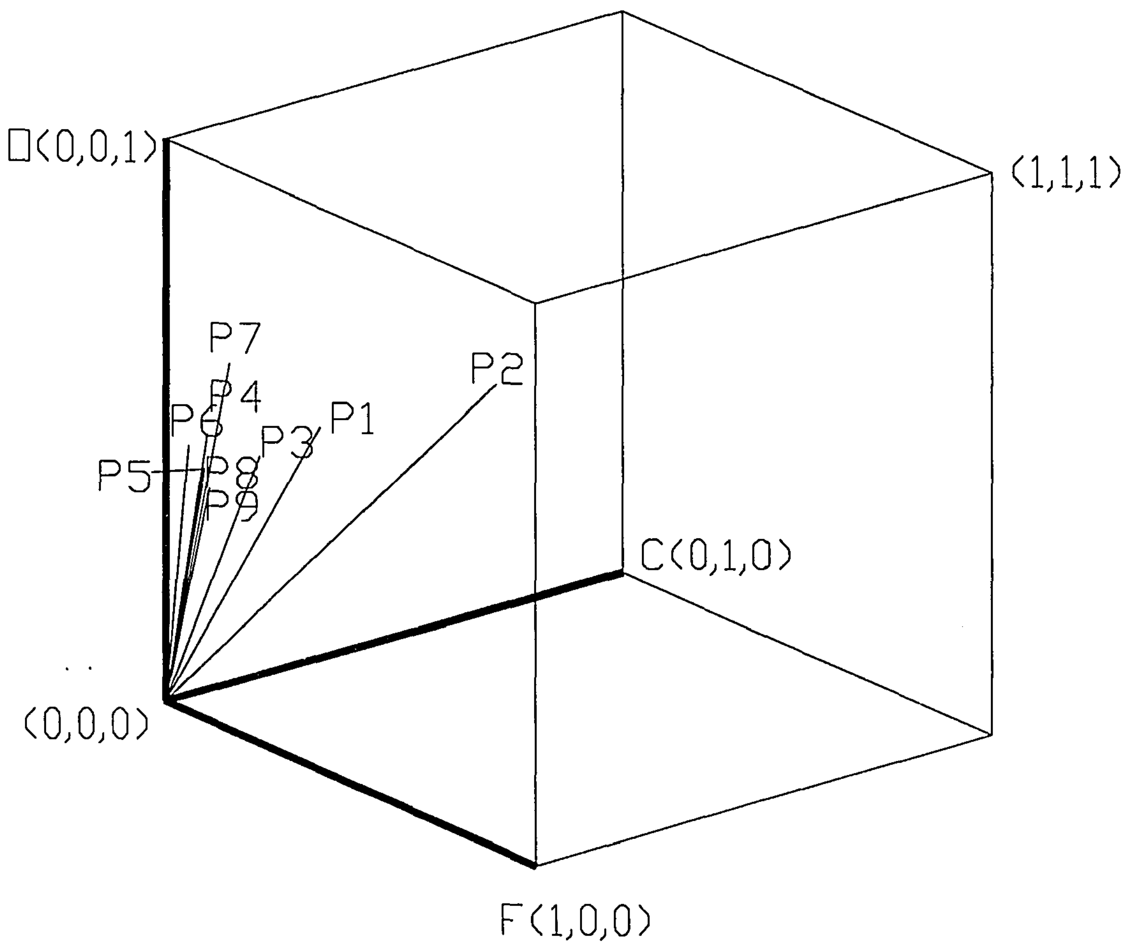


Figure 5.14: Pattern Recognition of Common-Cause Error Mode: Geometric Configuration

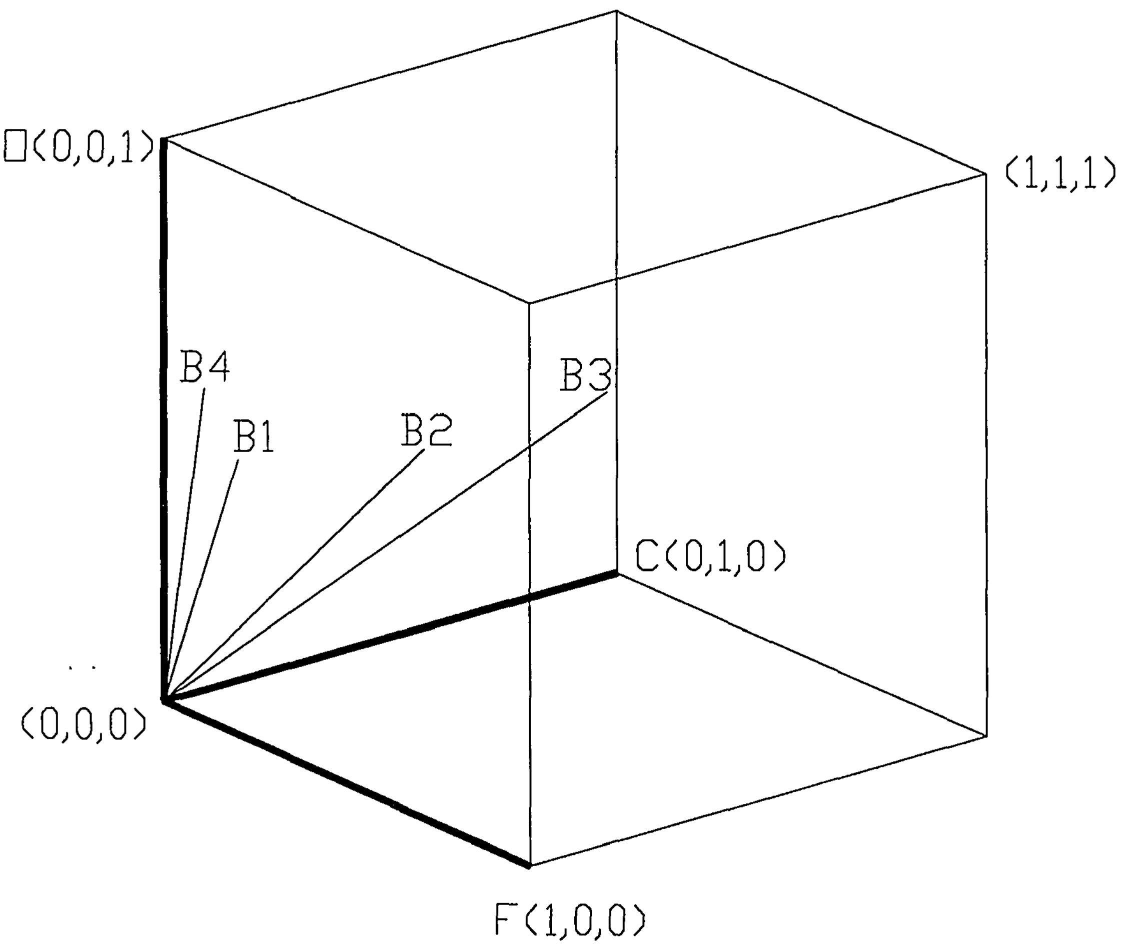


Figure 5.15: Behavior Domain of Common-Cause Error Mode: Geometric Configuration

vector can be calculated to produce varying shapes. Thus, the figure of hexahedron can be changed with different unit values on each of the three axes. Such simulation has shown trends of differences in identification modes among different ratings, the major reasoning common-cause error modes being I.1, I.8, and I.5. In pattern recognition of the common-cause error mode, the same trend results with major reasoning patterns, P.2, P.7, and P.1 in simulation V1, V2, V3, V5, V6, but different order results with P.2, P1, P.7 in V7 and V8. In simulation V4($\alpha : \beta : \gamma$=3:2:1), the major order of important reasoning pattern modes in common-cause error is P.2, P.1, and P.3. In the behavior domain common-cause error mode, the same result occurred with major reasoning .behavior domain B.3, B.2, appearing in every simulation except V2 which produced a different order of major reasoning behavior domain modes with B.3, B.4, and B.2.

Historical common-cause error recovery time zone

Points of common-cause error occurrence in time are shown in Figures 5.16, 5.17, and 5.18. Three time zones are shown; the initial time zone, the intermediate time zone, and the final time zone. Each level of recovery time zone affects the cost/effort of software development. In the final error recovery time zone, very expensive costs of development and error recovery occur. These involve I.1 and I.8 in the I_i mode, P.2 and P.7 in the P_j mode, and B.3 and B.4 in the B_k mode. In the intermediate error recovery time zone, it involves I.4 and I.5 in the I_i mode, P.1, P.3, P.4 and P.6 in the P_j mode, B.2 in the B_k mode. In the initial error recovery time zone, the most economical cost related error recovery time zone, it involves I.2, I.3, I.6 and I.7 in the I_i mode, P.5, P.8 and P.9 in the P_j mode, B.1 in the B_k mode.

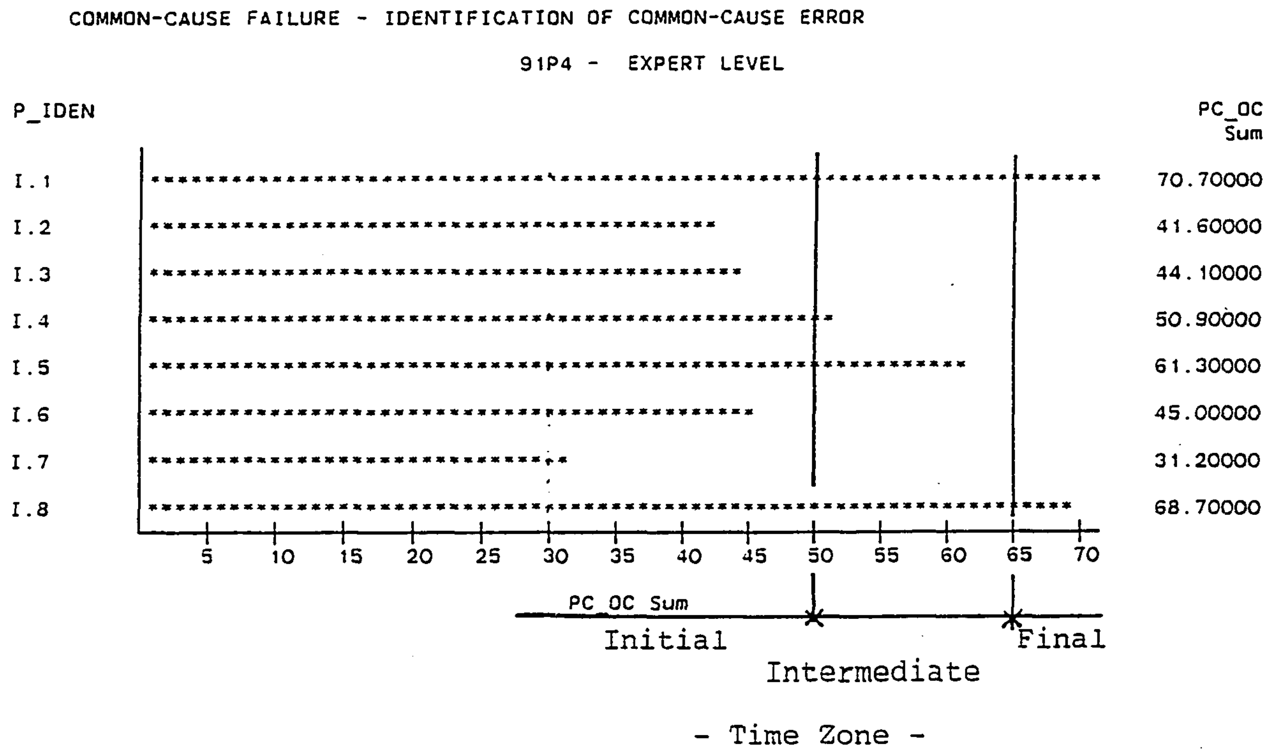


Figure 5.16: Identification of Common-Cause Error Mode: Recovery Time Zone (Units: portion (%) of occurrence time.)

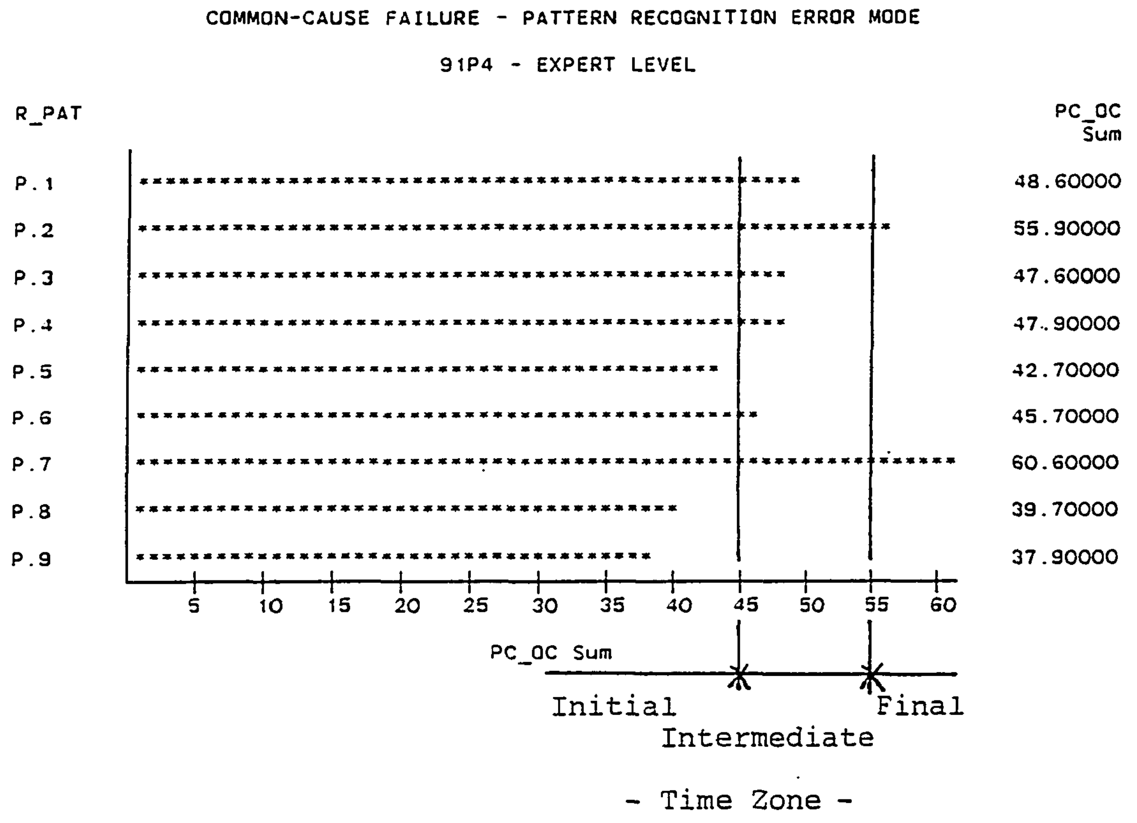


Figure 5.17: Pattern Recognition of Common-Cause Error Mode: Recovery Time Zone (Units: portion (%) of occurrence time.)

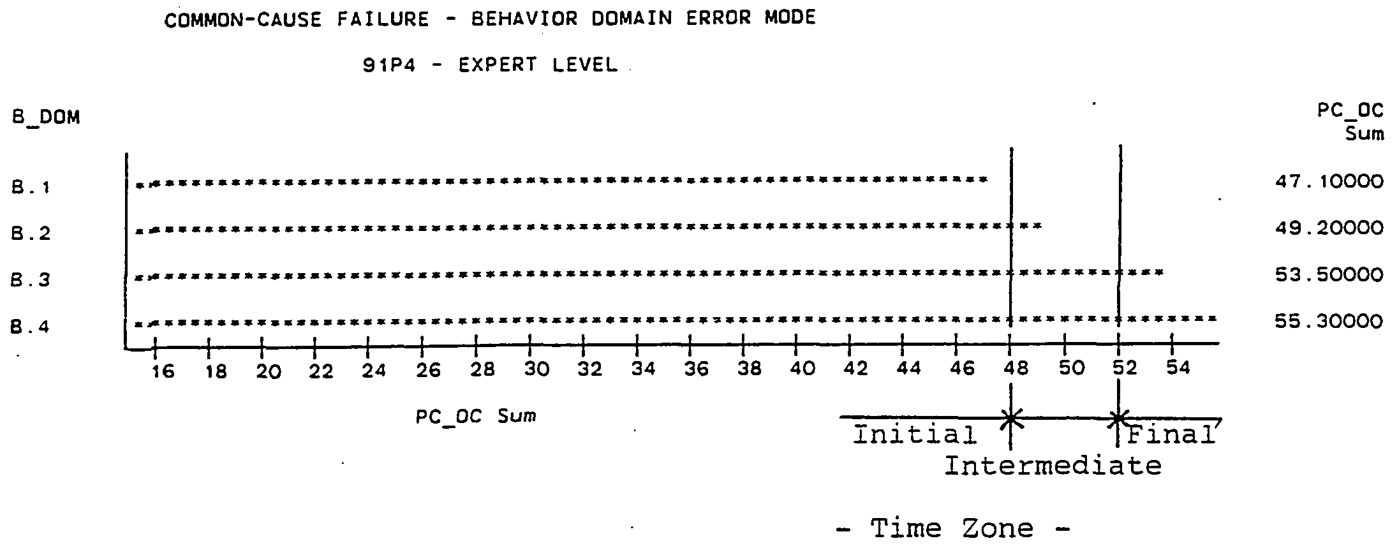


Figure 5.18: Behavior Domain of Common-Cause Error Mode: Recovery Time Zone (Units: portion (%) of occurrence time.)

Transition relationship diagram and grouping of major common-cause factors

Figure 5.19 shows the transition relationships among different common-cause function modes, and Table 5.12, Table 5.13, and Table 5.14 show the relative transition frequencies among common-cause error modes. Common-cause properties can be grouped according to their analogical characteristics with human behavioral aspects. There are four groups of common-cause factors, Group 1 of skill-based

Table 5.12: Frequency of Transition Load and Relationship between I_i and P_j [a] [b]

CCM	I.1	I.2	I.3	I.4	I.5	I.6	I.7	I.8	TOTAL	AVG
P.1:	18	4	5	5	1	11	11	0	55	6.9
P.2:	31	28	6	7	11	3	1	24	111	13.9
P.3:	2	12	33	1	0	4	4	7	63	7.9
P.4:	0	1	0	0	6	6	0	1	14	1.8
P.5:	0	3	8	4	1	2	1	0	19	2.4
P.6:	0	4	2	1	0	0	0	0	7	0.9
P.7:	0	0	0	8	17	0	0	0	25	3.1
P.8:	2	3	10	1	0	0	3	0	19	2.4
P.9:	0	0	0	0	0	6	4	2	12	1.5
TOTAL	53	55	64	27	36	32	24	34	325	.
AVG	5.9	6.1	7.1	3.0	4.0	3.6	2.7	3.8	.	.

[a] I_i: identification of common-cause error mode, P_j: pattern recognition of common-cause error mode.

[b] TOTAL: total frequency of error occurrence in each common-cause error mode, AVG: average frequency of error occurrence in each common-cause error mode.

behavior error domain, Group 2 of rule-based behavior error domain, Group 3 of knowledge-based behavior error domain, and Group 4 of model-based behavior error domain. The heavy lines indicated more frequent transition each other, that is, more strong relationship, than the light lines. The major transit relationship group is

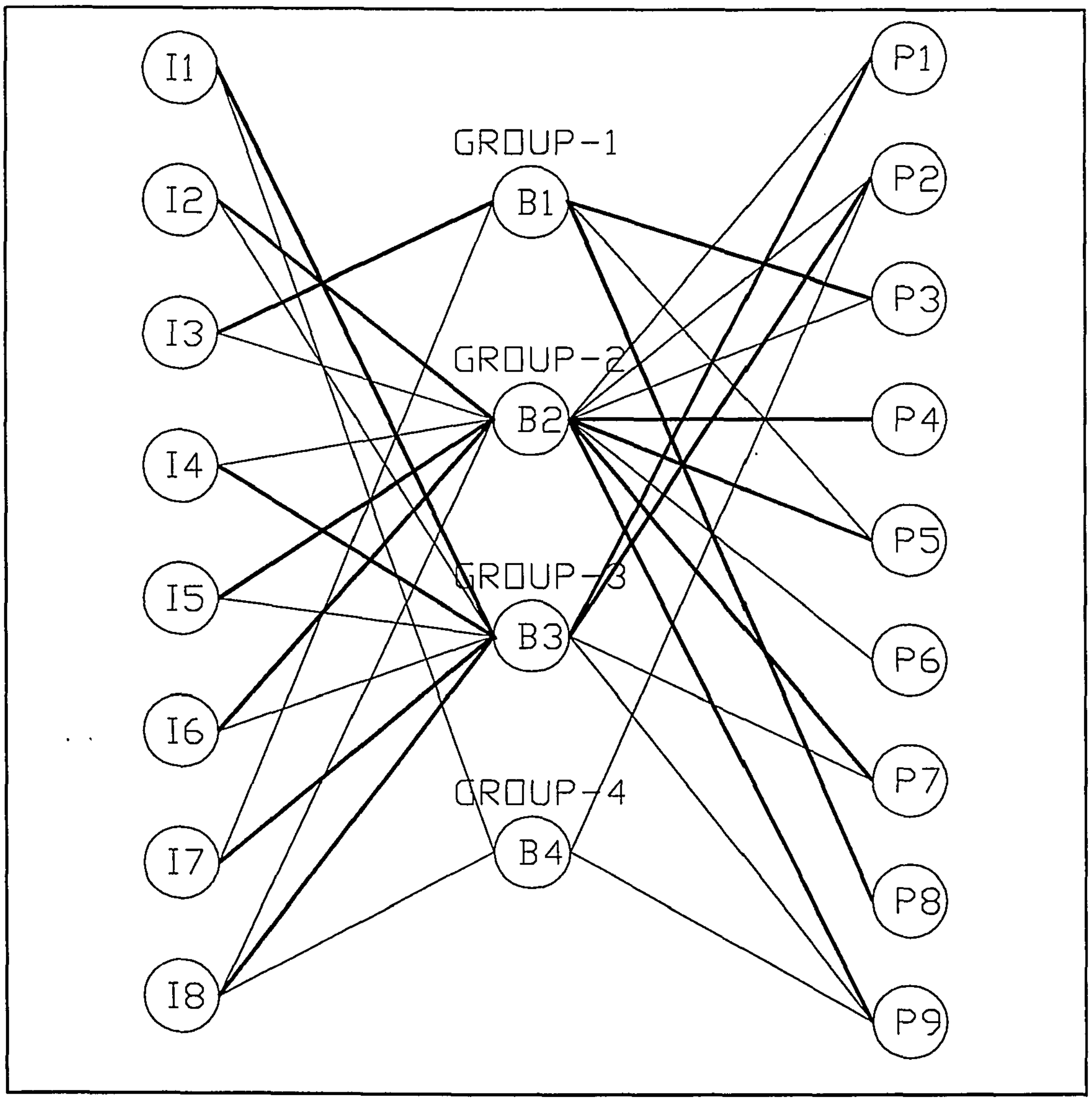


Figure 5.19: Transitions Relationship Diagram and Grouping of Common-cause Error Modes

Table 5.13: Frequency of Transition Load and Relationship between I_i and B_k [a] [b].

CCM	I.1	I.2	I.3	I.4	I.5	I.6	I.7	I.8	TOTAL	AVG
B.1:	0	9	36	3	0	0	7	0	55	6.9
B.2:	1	25	24	10	22	23	5	13	123	15.4
B.3:	48	21	5	13	12	9	11	20	139	17.4
B.4:	4	0	0	0	1	0	1	3	9	1.1
TOTAL	53	55	65	26	35	32	24	36	326	.
AVG	6.6	6.9	8.1	3.3	4.4	4.0	3.0	4.5	.	.

[a] I_i: identification of common-cause error mode, B_k: behavior domain of common-cause error mode.

[b] TOTAL: total frequency of error occurrence in each common-cause error mode, AVG: average frequency of error occurrence in each common-cause error mode.

I.3, P.3 and P.8 in B.1 Group 1; I.2, I.5, I.6, P.4, P.5, P.7 and P.9 in B.2 Group 2; and I.1, I.4, I.7, I.8, P.1 and P.2 in B.3 Group 3. The minor transit relationship group is I.7 and P.5 in B.1 Group 1; I.3, I.4, I.8, P.1, P.2, P.3 and P.6 in B.2 Group 2; I.2, I.5, I.6, P.7 and P.9 in B.3 Group 3; and I.1, I.8, P.2 and P.9 in B.4 Group 4. Each group has unique characteristics and error symptoms in various aspects of human behavior domain properties which were explained in the previous chapter.

Correlation and regression analysis

Correlation analysis measures the strength of the linear relationship between two variables such as frequency and correction time, correction time and point of occurrence in time, or frequency and point of occurrence in time. Table 5.15 indicates the extent to which these correlate with each other. A positive value of the Pearson correlation coefficient indicates more correlation between two variables. A negative value indicates less correlation. From Table 5.15, programming experience

Table 5.14: Frequency of Transition Load and Relationship between P_j and B_k [a] [b]

CCM	P.1	P.2	P.3	P.4	P.5	P.6	P.7	P.8	P.9	TOTAL	AVG
B.1:	0	0	34	0	7	0	0	15	0	56	6.2
B.2:	20	21	25	13	13	6	16	2	7	123	13.7
B.3:	35	87	4	2	0	1	9	2	4	144	16.0
B.4:	0	5	0	0	0	0	1	0	3	9	1.0
TOTAL	55	113	63	15	20	7	26	19	14	332	.
AVG	13.8	28.3	15.8	3.8	5.0	1.8	6.5	4.8	3.5	.	.

[a] P_j: pattern recognition of common-cause error mode, B_k: behavior domain of common-cause error mode.

[b] TOTAL: total frequency of error occurrence in each common-cause error mode, AVG: average frequency of error occurrence in each common-cause error mode.

had a negative influence in the overall correlation value. This means that a more experienced programmer had a better performance in the programming task with less frequent error, and less time taken in programming design and error correction. More spent time in the design phase resulted in a lower frequency of error occurrence during the computing phase.

The hypothesis in this experiment, that there were no significant differences in common-cause error properties among different categorical conditions such as specification requirements, programming languages, and subject expertise levels, was tested according to F statistics using the SAS ANOVA variance analysis test. Using factorial experimental analysis[1], there is no significant difference in frequency as a dependent variable (see Table 5.16) for three independent variables involving two levels for each variable. The probabilities values associated with the F value are 0.9770 for requirement, 0.3085 for expert level, and 0.9770 for programming language. No

[1] In a factorial design, the effects of different factors are considered simultaneously.

Table 5.15: Pearson Correlation Coefficients / Prob > | R | under H_0: Rho=0 / n = 10

.	Exp[a]	Freq[b]	Ct-T[c]	Com-T[d]	Sol-T[e]	Des-T[f]	Tot-T[g]
Exp	1.0000	-0.3741	-0.6512	-0.6656	-0.7483	-0.7947	-0.8030
.	0.0	0.287	0.041	0.036	0.013	0.006	0.005
Freq	-0.3741	1.0000	0.5295	0.6553	0.3971	0.1006	0.5184
.	0.287	0.0	0.116	0.040	0.256	0.782	0.125
CT-T	-0.6512	0.5295	1.0000	0.9090	0.8628	0.7019	0.9482
.	0.041	0.116	0.0	0.000	0.001	0.024	0.000
Com-T	-0.6656	0.6553	0.9090	1.0000	0.7409	0.5492	0.9400
.	0.036	0.040	0.000	0.0	0.014	0.100	0.000
Sol-T	-0.7483	0.3971	0.8628	0.7409	1.0000	0.7460	0.8728
.	0.013	0.256	0.001	0.014	0.0	0.013	0.001
Des-T	-0.7947	0.1006	0.7019	0.5492	0.7460	1.0000	0.7938
.	0.006	0.782	0.024	0.100	0.013	0.0	0.006
Tot-T	-0.8030	0.5184	0.9482	0.9400	0.8728	0.7938	1.0000
.	0.005	0.125	0.000	0.000	0.001	0.006	0.0

[a]Exp: programming experience.
[b]Freq: frequency of common-cause error mode.
[c]Ct-T: correction time of error.
[d]Com-T: computing time of program.
[e]Sol-T: problem solving time.
[f]Des-T: program design time.
[g]Tot-T: total spent time.

Table 5.16: ANOVA Test for Variance Analysis (Model: Frequency = Requirement Level Language; Dependent Variable: Frequency)

Source	D.F[a]	S.S[b]	M.S[c]	F[d]	$Pr > F$[e]
Model	3	137.1000	45.7000	0.41	0.7499
Error	6	663.8000	110.6333	.	.
Corrected total	9	800.9000	.	.	.
.	R.S[f]	C.V[g]	Root MSE[h]	.	Freq Mean
.	0.1712	32.7671	10.5182	.	32.1000
Source	D.F	Anova S.S	M.S	F	$Pr > F$
Requirement	1	0.1000	0.1000	0.00	0.9770
Subject Level	1	136.9000	136.9000	1.24	0.3085
Language	1	0.1000	0.1000	0.00	0.9770

[a]D.F: degree of freedom.
[b]S.S: the sum of squares.
[c]M.S: mean square.
[d]F: the F value for testing hypothesis that the group means for that effect are equal.
[e]$Pr > F$: the significant probability value associated with the F value.
[f]R.Square: measures how much variation in the dependent variable.
[g]C.V: coefficient of variation.
[h]Root MSE: estimates the standard deviation of the dependent variable.

Table 5.17: ANOVA Test for Variance Analysis (Model: CorrectionTime = Requirement Level Language; Dependent Variable: CorrectionTime)

Source	D.F[a]	S.S[b]	M.S[c]	F[d]	$Pr > F$[e]
Model	3	1509.1000	503.0333	0.76	0.5555
Error	6	3963.5000	660.5833	.	.
Corrected total	9	5472.6000	.	.	.
.	R.S[f]	C.V[g]	Root MSE[h]	.	Freq Mean
.	0.2758	56.7369	25.7018	.	45.3000
Source	D.F	Anova S.S	M.S	F	$Pr > F$
Requirement	1	476.1000	476.1000	0.72	0.4285
Subject Level	1	980.1000	980.1000	1.48	0.2689
Language	1	52.9000	52.9000	0.08	0.7867

[a]D.F: degree of freedom.

[b]S.S: the sum of squares.

[c]M.S: mean square.

[d]F: the F value for testing hypothesis that the group means for that effect are equal.

[e]$Pr > F$: the significant probability value associated with the F value.

[f]R.Square: measures how much variation in the dependent variable.

[g]C.V: coefficient of variation.

[h]Root MSE: estimates the standard deviation of the dependent variable.

significant difference occurred for correction time as a dependent variable associated with the F value probabilities, 0.4285 for requirement, 0.2689 for expert level, and 0.7867 for language (Table 5.17). There is only a significant difference in the expert level with computing time as a dependent variable. The significant probability values associated with F value are 0.3705 for requirement, 0.0400 for expert level, and 0.3617 for programming language (Table 5.18). Regression analysis typically is the

Table 5.18: ANOVA Test for Variance Analysis (Model: ComputingTime = Requirement Level Language; Dependent Variable: ComputingTime)

Source	D.F[a]	S.S[b]	M.S[c]	F[d]	$Pr > F$[e]
Model	3	231240.4000	77080.1333	2.91	0.1230
Error	6	158881.6000	26480.2666	.	.
Corrected total	9	390122.0000	.	.	.
.	R.S[f]	C.V[g]	Root MSE[h]	.	Freq Mean
.	0.5927	31.1143	162.7276	.	523.0000
Source	D.F	Anova S.S	M.S	F	$Pr > F$
Requirement	1	24800.4000	24800.4000	0.94	0.3705
Subject Level	1	180633.6000	180633.6000	6.82	0.0400
Language	1	25806.4000	25806.4000	0.97	0.3617

[a]D.F: degree of freedom.
[b]S.S: the sum of squares.
[c]M.S: mean square.
[d]F: the F value for testing hypothesis that the group means for that effect are equal.
[e]$Pr > F$: the significant probability value associated with the F value.
[f]R.Square: measures how much variation in the dependent variable.
[g]C.V: coefficient of variation.
[h]Root MSE: estimates the standard deviation of the dependent variable.

analysis of the relationship between one dependent variable and a set of independent variables to find out how well one can predict values of the dependent variable using least-squares estimates and error sum of squares based on the independent variables.

For an estimate of linear regression equation of the straight line that best fits the

Table 5.19: Regression Analysis (Dependent Variable: Frequency)

Source	D.F[a]	P.E[b]	S.E[c]	T[d]	$Pr > \|T\|$[e]
Intercept	1	22.1197	6.2585	3.53	0.0077
Correction Time	1	0.0391	0.0221	1.77	0.1155
Intercept	1	16.5718	6.7649	2.45	0.0400
Computing Time	1	0.0297	0.0121	2.45	0.0397
Intercept	1	30.5530	6.2605	4.88	0.0012
Design Time	1	0.0091	0.0318	0.29	0.7822

[a]D.F: degree of freedom.
[b]P.E: parameter estimate.
[c]S.E: standard error.
[d]T: T for H_0: parameter=0, the t test that parameter is zero.
[e]$Pr > |T|$: the probability that a t statistic would obtain a greater absolute value than that observed given that the true parameter is zero.

points between two variables involving frequency, correction time, computing time, and design time, the Table 5.19 has the coefficients and intercepts for the linear regrèssion equation describing frequency as a dependent variable:

$$F = 0.0391 \cdot C_t + 22.1197,$$

where F: frequency, C_t: correction time(Units: time in min.);

$$F = 0.0297 \cdot M_t + 16.5718,$$

where F: frequency, M_t: computing time(Units: time in min.);

$$F = 0.0091 \cdot D_t + 30.5530,$$

where F: frquency, D_t: design time(Units: time in min.).

From the Table 5.20, the linear regression line of design time as a dependent variable (correlation coefficient: 0.7019) is:

$$D_t = 0.5727 \cdot C_t + 23.7929,$$

where D_t: design time, C_t: correction time(Units: time in min.).

The acceptance confidence probabilities for the best fit regression line between the variables are significantly enough with t test statistics except design time.

Table 5.20: Regression Analysis (Dependent Variable: Design Time) [a]

Source	D.F[b]	P.E[c]	S.E[d]	T[e]	$Pr > \|T\|$[f]
Intercept	1	23.7929	58.0672	0.410	0.6927
Correction Time	1	0.5727	0.2055	2.787	0.0237

[a]DF: Degree of Freedom; PE: Parameter Estimate ; SE: Standard Error; T: T for H_0: parameter=0, the t test that parameter is zero; $Pr > |T|$: the probability that a t statistic would obtain a greater absolute value than that observed given that the true parameter is zero.

[b]D.F: degree of freedom.

[c]P.E: parameter estimate.

[d]S.E: standard error.

[e]T: T for H_0: parameter=0, the t test that parameter is zero.

[f]$Pr > |T|$: the probability that a t statistic would obtain a greater absolute value than that observed given that the true parameter is zero.

General observations and causal factors of common-cause error domain in human-software interaction

General observations and some symptoms of common-cause error were discovered during the experiment.

(1) Pre-existing knowledge was a major diagnostic symptom for completing the programming task. Subjects solved the problem, designed the requirements, and com-

puted using incorrect or mis-informed knowledge and methodologies. These symptoms had a lower frequency but required the longest correction time.

(2) Human memory, recognition, and availability were associated with some effects in the rule-based behavior domain. Logical, functional, syntax, design, and complexity error problems were examples of these symptoms which occurred with intermediate frequency and required a moderate amount of correction time. Within this category, the symptoms were related to pattern matching, stereotypical recognition, subject working memory and availability to take more logical rule-based problems.

(3) Human attention and perceptual ability can be affected by subject sensory-motor variability, recent physical and psychological events, and external environments. These symptoms constituted the minor reasons for common-cause error domain with greater frequency but the least correction time. They were identified as clerical, semantic, syntax, and some operation errors associated with skill-based behavior domain.

(4) Incomplete of knowledge was a major common-cause in the area of system operation, programming language, design method, and requirements specification. This causal factor was associated with the knowledge-based and rule-based behavior domain groups.

(5) Uncertainty of information was associated with knowledge-based, model-based, and skill-based behavior domain groups. This was a causal factor in the following areas: understanding and design of requirements specification, knowledge background of hardware and operating system, internal and external situation of environment, conditional factors in environment of system and subject.

(6) There was no significant difference in common-cause error properties between the

two levels among three of the subject factors: C and Fortran for programming language, A (dynamic programming assignment) and B (inventory control system) for requirements specification, and Level 1 (less programming expertise level) and Level 2 (more programming expertise level) for programming expertise level, the exception bein computing time in level of expertise.

(7) The major common-cause error modes arose from system design and requirement error, output and output formatting error, and program logic error. Design deficiency, logical formulation of the problem, and knowledge deficiency were major categories in pattern recognition in the common-cause error mode. The knowledge-based behavior domain and rule-based behavior domain were significantly important factors in common-cause error behavior domain.

(8) The knowledge-based behavior error domain was associated with the most significant error mode group in each of the common-cause function factors which involved identification and pattern recognition error modes. This is respectively requested the error causal prevention for knowledge-based behavior error domain including the following symptom factors: task identification, domain principle, object orientation, concurrent and intelligent design, integration and optimization method. Characteristics of these causal factors are human variability, selectivity, adaptation, working memory limitation, errors in a causal structure, availability, matching bias revisited, need for human decision making, incomplete knowledge, and uncertainty of information.

(9) Frequency and correction time in each common-cause error mode have a more consistent trend than point of occurrence in time among different error modes and over different task criteria.

Common-Cause Error Control Mechanism and Prevention

It is said that human-software interaction is more difficult to apply at higher levels, simply because great system complexity and flexibility imply more choices for system designers. It takes longer and is more difficult to analyze the system. Since there is a trend toward more sophisticated technology where the human is a programmer and a monitor of system behavior rather than an active controller, more human factors efforts and system improvements will be directed toward problems at this level.

With the analysis of experimental data, characteristics and properties of common-cause human error can be defined tentatively in the human behavior domain in human-software interaction, and the human error control mechanism can be re-designed.

Error control mechanism and environment

Figure 5.20 schematically represents a feed back process from a common-cause model to an error control scheme in human-software interaction. The experimental model for defining the common-cause human domain error in identification, pattern recognition, and behavior domain of common-cause proceeds to a 'black box' with the result of common-cause analysis. Then, with the representation of common-cause error analysis, knowledge processing and information processing of human-software interaction, it provides all information and guide lines for the new intelligent design which will be supported by concurrent design orientation and a model-based design method. Figure 5.21 shows environmental phenomena and influences in human-software interaction. This schematic frame-work explains how major common-cause

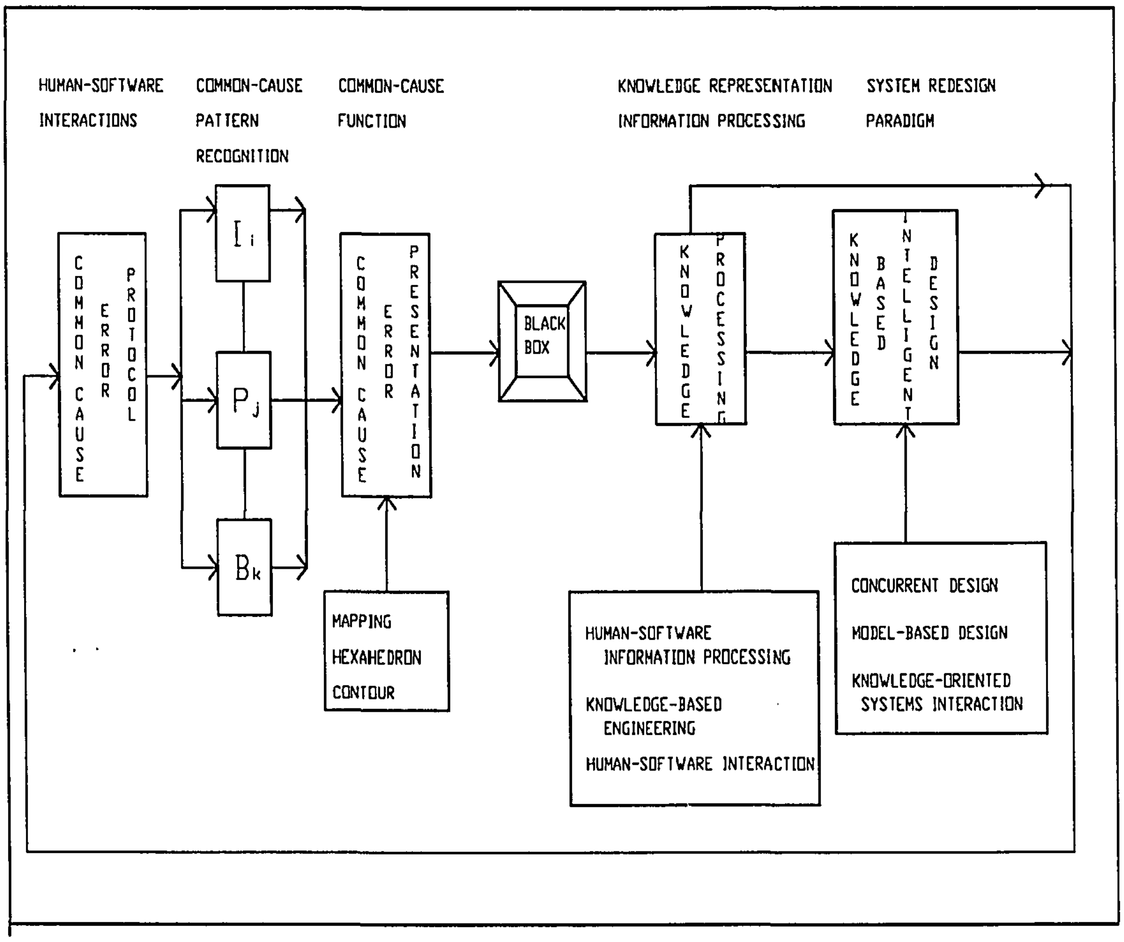

HUMAN-SOFTWARE INTERACTIONS
COMMON-CAUSE PATTERN RECOGNITION
COMMON-CAUSE FUNCTION
KNOWLEDGE REPRESENTATION INFORMATION PROCESSING
SYSTEM REDESIGN PARADIGM
COMMON CAUSE ERROR PROTOCOL
I_i
P_j
B_k
COMMON CAUSE ERROR PRESENTATION
BLACK BOX
KNOWLEDGE PROCESSING
KNOWLEDGE BASED INTELLIGENT DESIGN
MAPPING HEXAHEDRON CONTOUR
HUMAN-SOFTWARE INFORMATION PROCESSING
KNOWLEDGE-BASED ENGINEERING
HUMAN-SOFTWARE INTERACTION
CONCURRENT DESIGN
MODEL-BASED DESIGN
KNOWLEDGE-ORIENTED SYSTEMS INTERACTION

Figure 5.20: Common-Cause Error Control Mechanism

error domains relate to human-software interaction and system processing. In this error control environment, the system environment of human-software interaction including social problems, knowledge back ground, information system, situation motivation, and physical climate affect the human-software function. These factors are directly or indirectly, involved with system goal formation, knowledge orientation, human-software information processing, psychological mechanisms and physiological functioning for software task output.

Allocation of function and system interaction

Allocation of function is the process whereby the designer decides which tasks or functions should be allocated to the software subsystem and which to the human subsystem. The reliability of software can be improved less expensively than can the reliability of the human simply by putting extra components in parallel. Software can be changed fairly easily. The human was allocated some functions in older systems to permit flexibility for changes. Then, this flexibility could be achieved through software modification, making it practical for the designer to allocate even more functions to the software system. Therefore, the major decision in allocation of function involves checking that the human is left with a reasonable set of tasks. These tasks should neither overload nor under-load, considering the operators' capabilities.

In order to accomplish system goals in human-software interaction, designers must proceed systematically with seven relevant questions. These are:

(1) What system inputs and outputs must be provided to satisfy goals in human-software interaction?

(2) What operations functions are required to produce system outputs?

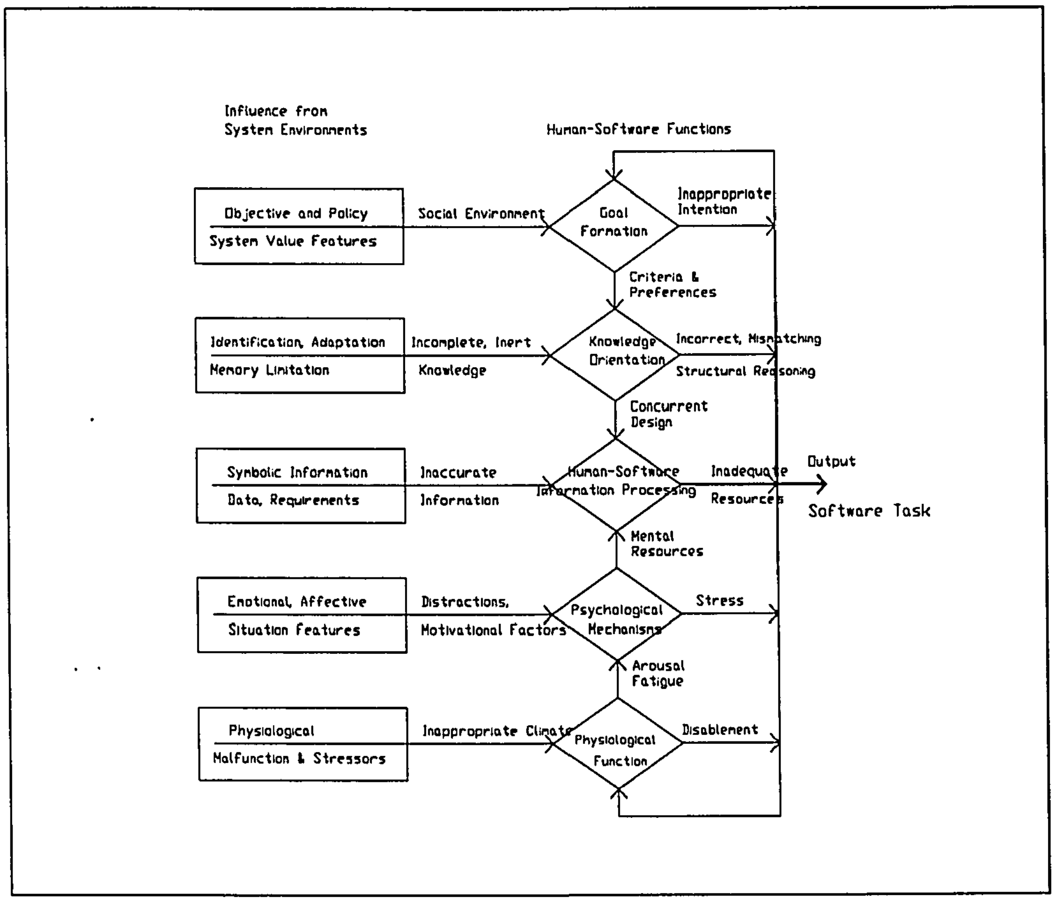


Figure 5.21: Common-Cause Error Control Environment and Human-Software Interaction

(3) What functions should the human perform within the human-software system?

(4) What are the training and skill requirements of human subjects?

(5) Are the tasks demanded by the system compatible with human capabilities?

(6) What interfaces does the human need to perform the job between the human and software systems?

(7) Does the human help or hurt software operation systems and vice versa?

Design analysis in human-software interaction

Common-cause design error patterns are due to inadequate design by the program designer. The three types of errors are the failure to implement human needs in the design, assigning an inappropriate function to a person, and failure to ensure the effectiveness of human-software interaction. Factors such as too much hastiness in the design effort, inclination of the designer to a particular design method and poor analysis of the requirement specifications needs are the causes of design errors.

Design principles for improving software task productivity in human-software interaction are as follows:

(1) Provide feedback error control mechanism with considerations of their environment;

(2) Be consistent in its system design and task completion;

(3) Minimize human memory demands by the information from human-software information processing;

(4) Keep the program simple, and not too much complexity;

(5) Match the program to software users' skill level;

(6) Sustain human, users or operators, orientation.

Knowledge-based human-software interaction and prevention

Interface software that can adapt to the current operator and the current context is a long-term research goal of the adaptive interface project. An adaptive human-software interface needs to include a knowledge-base that encompasses four domains; knowledge of the current human operator, knowledge of the human-software interaction scheme, knowledge of the operation task, and knowledge of the underlying human-software interaction system.

There are at least three major factors underlying the inadequacy of HSI[2] technology [80].

(1) Interface software is generally not viewed as part of the system but rather as a software package between the system and the operator [77].

(2) The design of effective interfaces is a difficult problem with sparse theoretical foundations [67].

(3) Software engineering principles are generally not given significant consideration in designing interfaces. Human operator specifications using the information hiding principle[83] in an abstract interface [61] need to be incorporated in the design of human-software interaction.

Advantages and disadvantages to adaptive human-software interfaces include:

Advantages:

(1) A system that dynamically allocates operations must be able to adapt to individual operators. It is imperative to have information specific to the current human operator for an optimal allocation process.

(2) Many times operators may not have the necessary information or expertise to

[2]HSI - Human-Software Interfaces.

modify their behavior.

(3) An adaptive human-software interface system increases operator proficiency with a new system and prevents frustration with an overly simple system.

Disadvantages:

(1) The operator may not be able to develop a coherent model of the human-software system if the system is frequently changing.

(2) The loss of control or the feeling of loss of control that the operator may experience.

(3) An adaptive interfaces also has an increase in implementation complexities and costs.

Control of common-cause factors of incompleteness and uncertainty

There are approaches and requirements for controlling the common-cause factors of incompleteness and uncertainty in aspects of the knowledge-based system including fuzzy set application. Software engineers are faced with information and knowledge simultaneously incomplete and uncertainties in human-software interaction. Since the initial phase of software system development, it became evident that these reasoning factors could not be neglected because they are strongly related to the way in which the common-cause error problem is controlled by a software system designer. There are two aspects of data from a common-cause error experiment in human-software interaction: incompleteness of information/knowledge, and uncertainty of information/knowlwdge [79].

(1) Incompleteness of information/knowledge was dealt with using some theories and techniques such as non-monotonic logics [66], truth maintenance system [64],

and reason maintenance [15]. Some causal factors of common-cause domain errors arise from incompleteness of information/knowledge in the area of: program source language, operating software system, background knowledge for requirements, and design methodology.

(2) Uncertainty of information/knowledge has been studied using techniques based on probability, subjective probability, evidence theory, fuzzy sets and possibility theory [23] [122]. Some causal factors of common-cause domain errors arise from uncertainty of information/knowledge in the areas of requirement of specifications, hardware systems, and environmental factors.

A set of requirements is needed in order to have a plausible technique of coping with uncertainty of information/knowledge. A list of requirements has been formulated as follows [87]:

(1) An inference should not depend on any assumptions about the probability distributions of the propositions,

(2) It should be possible to assert common relationships between propositions when the relationships are indeed known,

(3) It should be possible to posit information about any set of propositions and observe the consequences for the whole system,

(4) If the information provided to the system is inconsistent, this fact should be made obvious along with some notion of alternative ways that the information could be made consistent.

The list of requirements has been extended [66] and arranged into three categories bearing in mind distinct layers of the system, namely representation, inference, and control. The major requirements consider the following facts.

(1) The inference mechanisms should be logically tied to mechanisms initialized previously for knowledge acquisition. Thus, the knowledge-base is consistent and preserves properties within the framework of a specified formalism. The same formalism should form a basis for inference layer.

(2) A performance of the knowledge-base in the sense of its consistency and completeness should be taken into account by any inference procedure. The procedure should return not only a result of inference but also indicate the degree of its precision.

Improving software productivity

There is a general comment for software productivity improvement from the experiment in human-software interaction.

(1) Getting the best strategies from programmer: staffing, facilities, project goals, and management;

(2) Making policy more efficient: operating systems, environmental conditions, hardware work stations, office automation;

(3) Training for the intelligent and concurrent design methodologies;

(4) Consulting for appropriate requirements and matching to appropriate specification;

(5) Eliminate factors: biased orientations, pre-existing knowledge/information, automated documentation, quality assurance automated programming;

(6) Eliminate rework: front-end aids, knowledge-based software task assistant, information hiding, modern programming practices, incremental development;

(7) Building simpler products: process models, rapid prototype

(8) Reuse components: component libraries, application generators, fourth-generation

languages, feedback function from post project.

CHAPTER 6. CONCLUSION

Summary

The overall objectives of this research were to develop a cognitive paradigm including a new model of common cause human-domain error and a common cause error function to define internal common cause human-domain errors and also to determine how to control and prevent common cause errors in human-software interaction.

A laboratory experiment was performed to analyze the common causes of human error in software development and to identify software design factors contributing to the common cause effects in common cause failure redundancy. Three pilot projects with 46 subjects representing three skill levels were used to establish the design for a cognitive experiment. Following this study, a main experiment using ten programming experts was conducted in order to define a new cognitive paradigm, in the aspects of identification, pattern recognition, and behavior domain for internal human domain common-cause errors.

Main experimental results consisted of a 32.1 average (9.4 standard deviation) total common-cause error frequency, and 255.3 minutes average total error correction time during 523 minutes total computing time per each version of software development. Time spent in understanding and problem solving was 109 minutes, and design time for programming was 170 minutes. In the five categories of subject evaluation factors, average rating from experts' responses are (a) programming experience (23%), (b) knowledge background (21%), (c) intelligence (23%), (d) experiment attitude (18%), (e) work environmental conditions (15%). With the error occurrence frequency factor, the major reasoning categories in each common-cause error mode are: in the identification mode, I.3 (19.4%), I.2 (16.2%), and I.1 (15.9%); in the pattern recognition mode, P.2 (33.7%), P.3 (18.0%), and P.1 (15.7%); in the behavior domain mode, B.3 (43.6%) and B.2 (36.5%). When the error correction time factor is applied, I.1 (26.2%), I.5 (16.6%), and I.8 (13.9%) in the I_i mode; P.2 (44.8%) and P.1 (21.2%) in the P_j mode; and B.3 (62.7%) and B.2 (28.1%) in the B_k mode.

Each value listed in the common-cause function parameters can be produced by three factors, $F_{i,j,k}$, $C_{i,j,k}$, $O_{i,j,k}$. Such simulation has shown trends of differences in identification modes among different ratings, the major reasoning common-cause error modes being I.1, I.8, and I.5. In pattern recognition of the common-cause error mode, the same trend results with major reasoning patterns, P.2, P.7, and P.1 in simulation V1, V2, V3, V5, V6, but different order results with P.2, P1, P.7 in V7 and V8.

Each level of recovery time zone affects the cost/effort of software development. In the final error recovery time zone, very expensive costs of development and error recovery occur. These involve I.1 and I.8 in the I_i mode, P.2 and P.7 in the P_j mode, and B.3 and B.4 in the B_k mode. In the intermediate error recovery time zone, it involves I.4 and I.5 in the I_i mode, P.1, P.3, P.4 and P.6 in the P_j mode, B.2 in the B_k mode. In the initial error recovery time zone, the most economical cost related error recovery time zone, it involves I.2, I.3, I.6 and I.7 in the I_i mode, P.5, P.8 and P.9 in the P_j mode, B.1 in the B_k mode. The major transit relationship group is

I.3, P.3 and P.8 in B.1 Group 1; I.2, I.5, I.6, P.4, P.5, P.7 and P.9 in B.2 Group 2; and I.1, I.4, I.7, I.8, P.1 and P.2 in B.3 Group 3 from the transit relation diagram analysis.

From correlation analysis, more experienced programmers had better performance in programming tasks with less frequent error, and less amount of time in programming design and error correction. More spent time in design phase resulted in a lower frequency of error occurrence during the computing phase. There was no significant difference in subject task performances among the three categorical factors: requirements, languages, and expert levels, except in computing time for both subject expertise levels using SAS ANOVA variance analysis. Also linear regression lines provided for the best fits estimate points between two variables.

Finally, the characteristics and the properties of common-cause failure modes in human-software interaction were determined by the analysis of experimental data collected on the ten expert subjects and compared with data from each of the categorical conditions in various aspects of the human-software information processing scheme, knowledge-based engineering approach, and concurrent/intelligent design concepts. Some observations and symptoms were analyzed from the results of the common-cause error domain in human-software interaction. First, human mind-robustness based on his/her knowledge obtained before was a major diagnostic symptom for completing the task as related to the model-based and knowledge-based behavior domain category. Secondly, human memory, recognition, and availability were associated with some of the effects in the rule-based behavior domain. Third, human attention and perceptual ability could be affected by subject sensory-motor variability, recent physical and psychological events, and external environments. Fourth, incomplete of knowledge was a major common-cause in the area of system operation, programming language, design method, and requirements specification. Fifth, uncertainty of information was associated with knowledge-based, model-based, and skill-based behavior domain groups. Sixth, there was no significant difference in common-cause error properties between the two levels among three of the subject factors: languages, requirements, and expert levels. Seventh, the major common-cause error modes arose from system design and requirement error, output and output formatting error, and program logic error. Design deficiency, logical formulation of the problem, and knowledge deficiency were major categories in pattern recognition in the common-cause error mode. The knowledge-based behavior domain and rule-based behavior domain were significantly important factors in common-cause error behavior domain. Eighth, the knowledge-based behavior error domain was associated with the most significant error mode group in each of the common-cause function factors which involved identification and pattern recognition error modes. This is respectively requested the error causal prevention for knowledge-based behavior error domain. Ninth, frequency and correction time have a more consistent trend than point of occurrence in time among different error modes and over different task criteria.

Limitations and assumptions of this experiment are as follows:

(1) There was an assumption that all subject should be randomly selected.

(2) The software development project had some limitations with its scale: (a) no. of subjects (ten programming experts); (b) program assignment size (300-400 lines).

(3) There were limited levels for independent variables: (a) two programming expert levels (level-1, level-2); (b) two programming languages (C, Fortran); (c) two requirements specification (A, B).

(4) There was an assumption of no difference in performance due to gender (8 males, 2 females), or national source of undergraduate education (3 U.S./American, 1 Turkish, 6 Asian/Indian).
(5) Hardware system limitation (only a VINCENT work station used), Software operating system limitation (only the VINCENT network ULTRIX was used).

Conclusions

Conclusions derived from this research are:
(1) Two major common-cause reasoning groups exist in human-software interaction: (a) a major group consisting of knowledge-based behavior related errors indicated by design and knowledge deficiencies; (b) another major group consisting of rule-based behavior related errors indicated by logical errors, functional deficiencies, and system complexity.
(2) In training education sessions, consideration should be given to common-cause reasoning characteristics to eliminate the common-cause human domain error in human-software interaction. These characteristics include: (a) human mind-robustness (pre-existing incorrect knowledge and information); (b) pattern recognition in human memory; (c) human attention and perceptual ability; (d) incompleteness of knowledge and information uncertainty.
(3) Design with intelligence and concurrence by the knowledge-based processing: (a) knowledge acquisitions; (b) knowledge representation; (c) knowledge utilization.

Future research should be directed toward:
(1) Studies of common-cause error in system operation.
(2) Studies of common-cause failure in communication network.

(3) Application of fuzzy set theory to pattern recognition.

(4) Knowledge-based application to system operation.

(5) Intelligent and concurrent design properties in software engineering.

(6) Application of quality assurance techniques in the design and testing of human-software interaction system.

The results and analytical procedures showed during this study were to analyze common-causes of software development related to human error and to identify software design factors contributing to common types of error occurring in human-software interaction. Therefore, this can be applied to improving reliability of software development and to providing guidelines for design of software development.

BIBLIOGRAPHY

[1] Alford, M. "SREM at the Age of Eight: the Distributed Computing Design System." *Computer,* vol. 18, no. 4, Apr. 1985, pp. 36-46.

[2] Allport, D. A., B. Antonis, and P. Reynolds. "On the Division of Attention: A Disproof of the Single-Channel Hypothesis." *Quarterly Journal of Experimental Psychology,* 1972, 24, pp. 225-235.

[3] American Nuclear Society. "PRA Procedures Guide." *NUREG/CR- 0777,* 1979.

[4] Anderson, Ronald T., Lewis Neri. *Reliability-Centered Maintenance.* Elsevier Applied Science, New York, 1990, pp. 317.

[5] Askren, W. B., T. L. Regulinski. "Quantifying Human Performance for Reliability Analysis of Systems." *Human Factors,* 11, 1969, pp. 393-396.

[6] Bailey, R. W. *Human Performance Engineering.* Prentice-Hall Inc., Englewood Cliffs, N.J., 1982.

[7] Beakley, George C., and E. G. Chilton. *Introduction to Engineering Design and Graphics.* Macmillan, New York, 1973, pp. 761-763.

[8] Beech, H. R., L. E. Burns, B. F. Sheffield. *A Behavioral Approach to the Management of Stress.* John Wiley & Sons, Chichester, 1982.

[9] Bell, B. J., A. D. Swain. "A Procedure for Conducting a Human Reliability Analysis for Nuclear Power Plants." *NUREG/CR- 2254,* 1981.

[10] Beam, Walter R., J. D. Palmer, A. P. Sage. "Systems Engineering for Software Productivity." *IEEE Transactions on Systems, Man, and Cybernetics,* Vol. SMC-17, No.2, March/April 1987.

[11] Boehm, B. W., TRW. "Improving Software Productivity." *Computer,* Sep. 1987, pp. 43-57.

[12] Boehm, B. W. *Characteristics of Software Quality.* North Holland, New York, 1978.

[13] Boehm, Barry. "Software Reliability - Measurement and Management." *Abr. Proc. AIAA Software Management Conf., Los Angeles Sec.,* June 1976.

[14] Boem, B. W. "Software Engineering." *IEEE Trans. Comput.,* Vol. C-25, No. 12, 1976, pp. 1226-1241.

[15] Bowen, A. L. "Modal Propositional Semantics for Reason Maintenance System." *Proceedings of the 9th Int. Conf. on Artificial Intelligence,* LA, CA, 1985.

[16] Bowen, T. P., G. B. Wigle, J. T. Tsai. "Specifications of Software Quality Attributes." *RADC-TR-85-37(3 vol.),* Feb. 1985.

[17] Broadbent, D. E. *Decision and Stress.* Academic Press, New York, 1971.

[18] Broadbent, D. E. "Application of Information Theory and Decision Theory to Human Perception and Reaction." *Cybernetics of the Nervous System,* Ed. by N. Wiener & J. P. Schade , Amsterdam, Elsevier, 1965.

[19] Broadbent, D. E. *Perception and Communication.* Pergamon, London, 1958.

[20] Brooks, Fredrick P. Jr.. *The Mythical Man-Month.* Addison-Wesley Publishing Co., Reading, Mass., 1975.

[21] Card, S. K., W. K. English and B. J. Burr. "Evaluation of Mouse, Rate-Controlled Isometric Joystick, Step Keys and Text Keys for text Selection on a CRT." *Ergonomics,* vol. 21, no. 8, 1978, pp. 601-613.

[22] Chapannis, A. *New Approaches to Safety in Industry.* Ed. by W. Johnson, InComTec, London, 1972.

[23] Cohen, P. R., and M. R. Grinberg. "A Framework for Heuristic Reasoning about Uncertainty." *Proceedings of the 8th Int. Joint Conf. on Artificial Intelligence,* Karlsruhe, W Germany, 1983, pp. 355-357.

[24] Conway, Richard. *A Primer on Disciplined Programming Using PL/1, PL/CS, and PL/CT.* Winthrop Publishers, Cambridge, Mass, 1978.

[25] Cooper, Thomas, Nancy Wogrin. *Rule-Based Programming with OPS5.* Morgan Kaufmann Publishers, Inc., San Mateo, CA, 1988, pp. 6-7.

[26] Curtis, B. "Human Factors in Software Development." *IEEE,* Cat. No. EHO 185-9, 1981.

[27] Davis, W. S. *Information Processing Systems.* Addison-Wesley Publishing Company, Manila, PH, 1981.

[28] Dhillon, B. S. *Human Reliability: with Human Factors.* Pergamon Press, New York, 1986, pp. 29, 44-60.

[29] Dhillon, B. S. "Fault Tree Analysis." *Mechanical Engineer's Handbook,* Ed. by Mayer P. Kutz, Chapter 20, John Wiley & Sons, New York, 1985.

[30] Dhillon, B. S. *Reliability Engineering in Systems Design and Operation.* Van Nestrand Reinhold Co., New York, 1983.

[31] Dhillon, B. S., C. Singh. *Engineering Reliability: New Technique and Application.* John Wiley & Sons, New York, 1981.

[32] Duncan, J. "Response Selection Rules in Spatial-Choice Reaction Tasks." Ed. by S. Dornic, *Attention and Performance VI,* Lawrence Erlbaum Associates, Hillsdale, N.J., 1977.

[33] Endres, Albert. "An Analysis of Errors and Their Causes in System Programs." *IEEE Transactions on Software Engineering,* June 1975, pp. 140-149.

[34] Esogbue, A. O., and R. C. Elder. "Fuzzy Sets and the Modeling of Physician Decision Process, Part II: Fuzzy Diagnosis Decision Models." *Fuzzy Sets and Systems,* vol 3, 1980, pp. 1-9.

[35] Fitts, P. M., M. I. Posner. *Human Performance.* Brooks/Cole Publishing Company, Belmont, CA, 1967.

[36] Fitts, P. M., C. M. Seeger. "S-R Compatibility: Spatial Characteristics of Stimulus and Response Codes." *Journal of Experimental Psychology,* 1953, 46, pp. 199-210.

[37] Fraassen, Bas C. Van. "Rational Belief and the Common-Cause Principle." *What? Where? When? Why?* Ed. by Robert McLaughlin, D. Reidel Publishing Co., Dordrecht, Holland, 1982, pp. 193-209.

[38] Glymour, Clark. "Causal Inference and Causal Explanation." *What? Where? When? Why?* Ed. by Robert McLaughlin, D. Reidel Publishing Co., Dordrecht, Holland, 1982, pp. 179-191.

[39] Green, A. E., A. J. Bourne. *Reliability Technology.* John Wiley & Sons, London, 1972, pp. 22.

[40] Halstead, Maurice. *Elements of Software Science.* Elsevier North-Holland Inc., New York, 1977.

[41] Harm, O. J., J. S. Lappin. "Probability, Compatibility; Speed and Accuracy." *Journal of Experimental Psychology,* 1973, 100, pp. 416-418.

[42] Harris, Bernard. "Stochastic Models for Common Failures." *Reliability and Quality Control,* Ed. by A. P. Basu, Elsevier Science Publishers B. V., North-Holland, 1986, pp. 185-200.

[43] Henley, Ernest. J., H. Kumamoto. *Designing for Reliability and Safety Control.* Prentice Hall, Inc., Englewood Cliffs, N.J., 1985, pp. 273-275.

[44] Herman, L. M., B. H. Kantowitz. "The Psychological Refractory Period Effect: Only Half the Double-Stimulation Story?" *Psychological Bulletin,* 1970, 73, pp. 74-88.

[45] Hick, W. E. "On the Rate of Gain of Information." *Quarterly Journal of Experimental Psychology,* Apr. 1952, pp. 11-26.

[46] Hill, H. C. *Information Processing and Computer Programming: An Introduction.* Melville Publishing Co., L.A., C.A., 1973.

[47] Hyman, R. "Stimulus Information as a Determinant of Reaction Time." *Journal of Experimental Psychology,* 1953, 45, pp. 188-196.

[48] Ingram, Rick E. *Information Processing Approaches to Clinical Psychology.* Academic Press, Inc., London, 1986

[49] Jones, T. C. *Programming Productivity.* McGraw-Hill, New York, 1986.

[50] Kahneman, D., P. Slovic, and A. Tversky. *Judgement under Uncertainty: Heuristics and Biases.* Cambridge University Press, 1982.

[51] Kantowitz, Barry. H. , and Robert D. Sorkin. *Human Factors: Understanding People-System Relationships.* John Wiley & Sons, New York, 1983, pp. 30-59.

[52] Kantowitz, Barry H. "Interfacing Human Information Processing and Engineering Psychology." *Human Performance and Productivity,* Vol.2: Information Processing and Decision Making, Ed. by W. C. Howell, E. A. Fleishman, Lawrence Erlbaum Associations, Publishers, Hillsdale, New Jersey, 1982, pp. 48.

[53] Kantowitz, Barry. H., and J. L. Knight. *Testing Tapping Timesharing: II. Auditory Secondary Task.* Acta Psychologica, 1976, 40, pp. 343-362.

[54] King, P. J., and E. H. Mamdani. "The Application of Fuzzy Control Systems to Industrial Processes." *Aotomatica,* 13, 1977, pp. 235-242.

[55] Kitahara, Yasusada. *Information Network System: Tele- communications in the 21st century.* Heinemann Educational Books, London, 1983.

[56] Knight, J. C., and N. G. Leveson. "An Experimental Evaluation of the Assumption of Independence in Multi-Version Programming." *IEEE Transactions on Software Engineering,* SE-12(1), pp. 96-109.

[57] Konz, S. "Quality Circles: Japanese Success Story." *Industrial Engineering,* 15, Oct. 1979, pp. 24-27.

[58] Koriat, A. and S. Lichtenstein. "Reasons for Confidence." *Journal of Experimental Psychology: Human Learning and Memory,* 6, 1980, pp. 107-117.

[59] Lee, K. W., F. A. Tillman, J. J. Higgins. "A Literature Survey of the Human Reliability Component in a Man-Machine System." *IEEE Trans. Reliab.,* Vol. 37, No. 1, Apr. 1988.

[60] Lehman, J. F. and J. G. Carbonell. *Learning the User's Language: A Step Towards Automated Creation of User Models.* Carnegie-Mellon University, Pittsburgh, PA, 1987.

[61] Liskov, B., and S. Zilles. "Specification Techniques for Data Abstractions." *IEEE Trans. Software Engr.,* vol. SE-1, no. 1, 1975, pp. 7-19.

[62] Lu, Stephen C. Y. "Knowledge Processing Technology for Simultaneous Engineering." *CIM Management,* Dec. 1990.

[63] Luchins, A. S., E. H. Luchins. "New Experimental Attempts at Preventing Mechanization in Problem Solving." *Journal of General Psychology,* 42, 1950, pp. 279-297.

[64] McAllester, D. A. *An Outlook on Truth Maintenance.* MIT Artificial Intelligence Laboratory, Cambridge, Mass., 1980.

[65] McCormick, E. J., M. S. Sanders. *Human Factors in Engineering and Design.* McGraw-Hill Book Comp., New York, 1982.

[66] McDermott, D. *Non-monotonic logic II: Non-monotonic modal theories.* J. Assn. Comp. Machinery 29, 1982, pp. 33-57.

[67] Meads, J. "Report on the SIGCHI Workshop on Planning for User Interface Standards." *SIGCHI Bull.,* vol. 17, no 2, 1985, pp. 11-16.

[68] Meister, D. "Reduction of Human Error." *Handbook of Industrial Engineering,* Ed. by G. Salvendy, John Wiley & Sons, New York, 1982, pp. 6.2.1-6.2.9.

[69] Meister, D. *Human Factors: Theory and Practice.* John Wiley & Sons, New York, 1976. pp. 11-56.

[70] Meister, D. *Human Factors: Theory and Practice.* Wiley, New York, 1971.

[71] Meister, D. "Human Factors in Reliability." *Reliability Handbook,* Ed. by W. G. Ireso, McGraw- Hill, New York, 1966, pp. 12.2-12.37.

[72] Meister, D. "The Problem of Human-initiated Failures." *Proceedings of the Eighth National Symposium on Reliability and Quality Control,* IEEE, New York, 1962, pp. 234-239.

[73] Melliar-Smith, P.M., B. Randell. "Software Reliability: The Role of Programmed Exception Handling." *Association for Computing Machinery Inc. SIGPLAN Notices,* Vol. 12, No. 3, March 1977, pp. 95-100.

[74] Miller, R. B. "A Method for Man-Machine Task Analysis." *Technical Report No. 10,* Wright-Air Development Center, U.S. Air Force Base, Ohio, June 1953, pp. 53-137.

[75] .Mitta, Deborah. "A Methodology for Quantifying Expert System Usability." *Human Factors,* 1991, 33(2), pp. 233-245.

[76] Musa, J. D., A. Iannino, K. Okumoto. *Software Reliability Measurement, Prediction, Application.* McGraw-Hill Book Com., New York, 1987, pp. 77-101.

[77] Morland, D.V.. "The Evaluation of Software Architecture." *Datamation,* Feb. 1985, pp. 123-132.

[78] Morse, A. "Some Principles for the Effective Display of Data." *Commun. ACM,* 1979, pp. 94-101.

[79] Nola, A. Di, S. Sessa, W. Pedrycz, and E. Sanchez. *Fuzzy Relation Equations and their Applications to Knowledge Engineering.* Kluwer Academic Publishers Group, Netherland, 1989, pp. 175-185.

[80] Norcio, Anthony F. and Jaki Stanley. "Adaptive Human-Computer Interfaces: A Literature Survey and Perspective." *IEEE Trans. on Systems, Man, and Cybernetics,* vol. 19, no. 2, Mar./Apr. 1989, pp. 399-408.

[81] Norman, D. A. "Categorization of Action Slips." *Psychological Review,* 88, 1981, pp. 1-15.

[82] Pages, A., M. Gondran. *System Reliability Evaluation and Prediction in Engineering.* Springer-Verlag, 1986.

[83] Parnas, D. L. "On the Criteria to be used in Decomposing Systems into Modules." *Commun. ACM,* vol. 15, 1972, pp. 1053-1058.

[84] Peters, G. "Human Error: Analysis and Control." *Journal of the ASSE,* Jan. 1966.

[85] Petersen, Dan. *Human-Error Reduction and Safety Management.* Garland STPM Press, New York, 1982.

[86] Pressman, Roger S. *Software Engineering.* McGraw-Hill Book Co., New York, 1987, pp. 191-201.

[87] Quinlan, J. R. "INFERNO: A Cautious Approach to Uncertain Inference." *Computer J.* 26, 1983, pp. 255-269.

[88] Raduchel, William, Otto Eckstein. "Economic Modeling Languages: The DRI Experience." *Human-Computer Interaction,* Ed. by G. Salvendy, Elsevier Science Publishers B.V., Amsterdam, 1984.

[89] Randell, B. "Fault Tolerant and System Structuring." *IEEE, Preceeding of 4th Jerusalem Cof. on Information Technology,* May 1984.

[90] Rasmussen, J., K. Duncan, J. Leplat. "Cognitive Control and Human Error: New Technology and Human Error." *New Technologies and Work a Wiley series,* 1987, pp. 53-61.

[91] Rasmussen, J. "What Can Be Learned From Human Error Reports?" *Changes in Working Life,* Ed. by K.D. Duncan, M.M. Gruneberg, and D. Wallis, John Wiley & Sons Ltd., New York, 1980, pp. 97-113.

[92] Reason, James. "Generic Error-Modelling System(GEMS): A Cognitive Framework for Locating Common Human Error Forms." *New Technology and Human Error,* Ed. by J. Rasmussen, K. Duncan and J. Leplat, John Wiley & Sons. Ltd, 1987, pp. 63-83.

[93] Reason, J. T. "Actions not as Planned: The Price of Atomization." *Aspects of Consciousness,* Ed. by G. Underwood and R. Stevens, Vol. 1, London Academic Press, 1979.

[94] Regulinski, T. L., W. B. Askren. "Stochastic Modeling of Human Performance Effectiveness Functions." *Proceeding of the Annual Reliability and Maintainability Symposium,* IEEE, New York, 1972, pp. 407-416.

[95] Regulinski, T. L., W. B. Askren. "Mathematical Modeling of Human Performance Reliability." *Proceeding of Annual Symposium on Reliability,* IEEE, New York, 1969, pp. 5-11.

[96] Rook, L. W. "Reduction of Human Error in Industrial Production." *Report No. SCTM 93-62(14),* Sandia Laboratories, Albuquerque, New Mexico, June 1962.

[97] Rouse, W. B. "Fuzzy Models of Human Problem Solving." *Advances in Fuzzy Set, Probability Theory and Applications,* Ed. by P. P. Wang, Plenum Press, New York, 1983.

[98] Rouse, W. B. *Systems Engineering Models of Human-Machine Interaction.* North Holland, New York, 1980.

[99] Rouse, W. B. "A Model of Human Decision Making in Fault Diagnosis Tasks that include Feedback and Redundancy." *IEEE Trans., Syst. Man Cyber.,* SMC-9(4), Apr. 1979.

[100] Rouse, W. B., and R. M. Hunt. "A Fuzzy Rule-Based Model of Human Problem Solving in Fault Diagnosis Tasks." *Proc. IFAC 8th Triennial World Congress,* Kyoto, Japan, 1981.

[101] Ross, D., and K. Schoman. "Structured Analysis for Requirements Definition." *IEEE Trans., Software Engineering,* vol. 3, no. 1, Jan. 1977, pp. 6-15.

[102] Salmon, Wesley C. " Causal Forks and Common Causes." *Scientific Explanation and the Causal Structure of the World,* Princeton U. Press, 1984.

[103] SAS Institute Inc. *SAS/STAT User's Guide.* Vol 1 & 2, SAS Institute Inc., Cary, NC, 1990, pp. 2-27, 209-244, 1351-1456.

[104] Shooman, M. L. *Software Engineering.* McGraw-Hill Book Comp., New York, 1983, pp. 296-403.

[105] Sievert, G. E., and T. A. Mizell. "Specification-Based Software Engineering with TAGS." *Computer,* vol. 18, no. 4, Apr. 1985, pp. 56-65.

[106] Sutton, Robert. *Modeling Human Operators in Control System Design.* John Wiley & Sons Inc., New York, 1990, pp. 141-183.

[107] Swain, A. D., H. E. Guttman. "Handbook of Human Reliability Analysis with Emphasis on Nuclear Power Plant Application." *NUREG/CR-1278,* 1980.

[108] Swain, A. D. "An Error-Cause Removal Program for Industry." *Human Factors,* 12, 1973, pp. 207-221.

[109] Swain, A. D. *Design Techniques for Improving Human Performance in Production.* Industrial & Commercial Techniques Ltd., Fleet Street, London, EC4, 1973, pp. 30-32.

[110] Swain, A. D. "A Method for Performing a Human-Factors Reliability Analysis." *Report SCR-685,* Sandia Corporation, Albuquerque, New Mexico, Aug. 1963.

[111] Teichroew, D., and E. Hershey. "PSL/PSA: A Computer Aided Technique for Structured Documentation and Analysis of Information Processing Systems." *IEEE Trans., Software Engineering,* vol. 3, no. 1, 1977, pp. 41-48.

[112] Thayer, Thomas A., Myronlipow, and E. C. Nelson. *Software Reliability.* North-Holland Publishing Co., New York, 1978.

[113] Tillman, F. A., C. L. Hwang, W. Kuo. *Optimization of Systems Reliability.* Marcel Dekker Inc., New York, 1980.

[114] Treisman, A. M. "Strategies and Models of Selective Attention." *Psychological Review,* 1969, 76, pp. 282-299.

[115] Walters, G. F., J. A. McCall. "Software Quality Matrices for Life-Cycle Cost-Reduction." *IEEE Trans. Reliab.,* Vol. R-28, No. 3, 1979, pp. 212-220.

[116] Watson, I. A. "Review of Common Cause Failure." *National Centre of Systems Reliability,* Report NCSR R27, 1981.

[117] Weinberg, G. M., E. L. Schulman. "Goals and Performance in Computer Programming." *Human Factors,* Vol. 16, No. 1, 1974, pp. 70-77.

[118] Weiberg, G. M. *The Psychology of Computer Programming.* Van Nostrand Reinhold, New York, 1971.

[119] Welford, H. T. *Skilled Performance: Perceptual and Motor Skills.* Glenview, Ill: Scott, Foresman, 1976.

[120] Woods, David D. "Modeling and Predicting Human Error." *Human Performance Models for Computer-Aided Engineering,* Ed. by J. I. Elkind, Academic Press, Inc., 1990.

[121] Youngs, Edward A. "Human Errors in Programming." *Human Factors in Software Development: COMPSAC81.* Ed. by Bill Curtis, L.A., C.A. 1981, pp. 383-392.

[122] Zadeh, L. A. "The Role of Fuzzy Logic in the Management of Uncertainty in Expert Systems." *Fuzzy Sets and Systems,* 11, 1983, pp. 199-227.

APPENDIX A. THE COMMON-CAUSE PRINCIPLE

Common Cause and Rational Belief

Wesley Salmon [102] and Bas C. Van Fraassen [37] have successively refined and elaborated Reichenbach's principle of the common cause, as part of a wide-ranging inquiry into statistical inference and explanation. In this section, the probabilistic concept of common cause, that is, the principle of the common cause, is derived.

Reichenbach's common cause principle says roughly that if there is a positive correlation between simultaneous, spatially separate events, then there is a third event in their common past which explains for their frequent joint occurrence. This is an empirical statement. It reminds one somewhat of certain traditional principles of metaphysics, such as that every event should have a cause. A scientific theory concerning those correlated events is not complete unless it exhibits, or implies that there is, such a common cause as a tactical maxim for scientific inquiry.

Extreme Bayesianism is the position that a rational person's epistemic state can be represented faithfully and without loss by means of a probability function; that any probability function at all can so represent some rational person; and that rational change of epistemic state consists in conditioning of that personal probability on the total evidence received.

If Reichenbach's principle can be explained as an empirical proposition, there are many probability functions that do not give it a high value. If one manages secondly his garden of beliefs in such a way that, whenever he has a certain degree of belief that two events are positively correlated, he gives at least that degree of belief to the proposition that they have a common cause, then either he gives probability one to that empirical proposition (the common cause principle) or else his belief change does not follow the pattern of conditioning on the total evidence. As Salmon has rightly emphasized, the principle of the common cause will appear as a powerful argument for scientific realism when it comes in any of these rational inference related forms.

The Principle of the Common Cause

Two events, A and B, are called statistically independent if $P(AB) = P(A)P(B)$. When the equality is replaced by the greater-than relation we may call them positively correlated. A third event C, using the conditional probability $P(-/C)$ may have a relationship with either of these notions:

if (1) $P(AB) > P(A)P(B)$

then there is an event C *such that*

(2) $P(AB/C) = P(A/C)P(B/C)$

(3) $P(AB/\bar{C}) = P(A/\bar{C})P(B/\bar{C})$

(4) $P(A/C) > P(A/\bar{C})$

(5) $P(B/C) > P(B/\bar{C})$

With the time element, the AB is an event which happens at a given time if and only if both A and B happen at that time. Suppose that put A_t for the (individual, non-generic) event is the occurrence of (generic) event A at time t. Suppose that has always occurred C in the intersection of the past cones of the occurrences of the A

and B. There are two relatively independent questions which may be raised. The first question is whether there is always an event C at a preceding time such that the above probabilistic relations hold. The second one is whether if C satisfies the stated conditions, it follows that C accounts for the correlation (can it reasonably be termed the cause?).

Statistical Dependence

The following relationships are important on examining statistical dependence:

(6) $P(AB) > P(A)P(B)$*: A and B are positively correlated;*

(7) $P(A/B) > P(A)$*: A has a positive dependence on B;*

(8) $P(A/B) > P(A/\bar{B})$*: B is positively relevant to A;*

(9) $P(A/BC) = P(A/C)$*: C screens off B from A.*

In each case, if the probability function P is replaced by the conditional probability $P_x = P(-/X)$, then the same terminology can be used with adding the rider *relative to X.* One can say easily how cognate terms such as *independent, negatively relevant,* and the like are used. Symmetric term, *A and B are,* is appropriate because the relationship is so clearly symmetric in A and B. It is important that there is no need to memorize the terms in (6)-(8), and their cognates, because the ones which are easily confused are actually equivalent (provided all the probabilities involved are well-defined). To get their this precise, let the letter, $\Re$, range over *positive linear relations* among numbers, defined by the properties [37]:

If $0 \leq x, y \leq 1$, *and* $0 < b$

then

(I) $x \Re y$ *iff* $bx \Re by$

(II) $x\Re y$ *iff* $(b+x)\Re(b+y)$

where $=, <, >, \leq, \geq$ *are all positive linear relations.*

LEMMA. *If* $\Re$ *is a positive linear relation and* $P(X), P(\bar{B}X)$ *are positive, then the following are mutually equivalent:*

(A) $P(AB/X)\Re P(A/X)P(B/X)$

(B) $P(A/BX)\Re P(A/X)$

(C) $P(A/BX)\Re P(A/\bar{B}X)$

Using this Lemma, there are restatements on the properties of the common cause in Reichenbach's principle in follows:

(10) If A *and* B *are positively correlated, then there is an event* C *such that*

(A) A *and* B *are independent relative to* C *and also relative to* $\bar{C}$,

(B) C *is positively relevant both to* A *and to* B.

(11) If B *is positively relevant to* A *then there is an event* C *such that*

(A) C, *and* $\bar{C}$, *screens off* B *from* A,

(B) Both A *and* B *have a positive dependence on* C.

APPENDIX B. EXPERIMENTAL MATERIAL AND REQUIREMENT SPECIFICATIONS

[Experiment Procedures: Subject Phase]

(1) Subject screen and interview: Subjects are screened and interviewed by the project supervisor according to their eligible capability for the experiment of human-software interactions.
(2) Subject life data collection: Subject life data are collected including personal data, computer programming background, experience, and any medical problem.

(3) Initialization session: In this session, initial information about the experiment is provided to subject with the general description of project, requirements of specification, and whole procedure of experiment and data collection.
(4) Educational and training session: Manual solving and mathematical validation about programming requirements are provided, and common cause error modes are taught about their definition, data collection method, and representation of their allocation.
(5) Program design: After understanding requirements, problems can be solved and the program is designed without encoding to computer. This is done in out of experiment station.
(6) Consultation session: A consultation session is provided for better understanding of requirements, system components, common cause error modes before program encoding to computer.

(7) Subject preliminary questionnaires: Just before start to program encoding, special conditions of subject's programming environments and design considerations are gathered from the subject.
(8) Program encoding to computer: The designed program is encoded to the computer using specified hardware work station/operating system.
(9) Representational interview session: In each 30-45 minutes, common cause error data can be collected. During the programming, Common cause errors are produced from human-software interactions and program failures by the verification of program. With correcting the error, occurrence time, correction time, and contents of the failures are recorded on data collection sheet. During programming, a subject is not interrupted in any way.
(10) Representation of common-cause errors: With the representational interview, common-cause error protocol can be classified to identification mode, and allocated to pattern recognition mode and behavior domain mode with representational interview for common-cause errors.

(11) Validation of data collection: Subject's task behavior is monitored by the supervisor using another simultaneous logging monitor, and that is taped to video recorder for their data validation.
(12) Continue to collect data for common-cause human errors until requirements are completed with a correct formatted output.
(13) After finishing the experiment, evaluate the experiment and predict a rating weight for subject performance evaluation.

91-P4-(Date/No.):______________________ HUMAN-SOFTWARE DATA COLLECTION
Programmer:91P4-_______ Starting Time:______________
Monitor :____________ Ending Time:______________

Oc No	Occur Time Act 1	Cont	Mode Code Ii	Pj	Bk	Description of Failure and Error	Correct. T(min)
--	------	------	----	----	----	----------------------------------	---------
						----------------------------------	
--	------	------	----	----	----	----------------------------------	---------
						----------------------------------	
--	------	------	----	----	----	----------------------------------	---------
						----------------------------------	
--	------	------	----	----	----	----------------------------------	---------
						----------------------------------	
--	------	------	----	----	----	----------------------------------	---------
						----------------------------------	
--	------	------	----	----	----	----------------------------------	---------
						----------------------------------	
--	------	------	----	----	----	----------------------------------	---------
						----------------------------------	
--	------	------	----	----	----	----------------------------------	---------
						----------------------------------	
--	------	------	----	----	----	----------------------------------	---------
						----------------------------------	
--	------	------	----	----	----	----------------------------------	---------
						----------------------------------	
--	------	------	----	----	----	----------------------------------	---------
						----------------------------------	

Serial #: 91-P4-___ Date: ________, 1991

.Name :_______________, ______________ .S.S.#:_____-___-______
(Last) (First)

.Address:(H)_________________________________ .Tel:____________
(O)_________________________________ .Tel:____________

[Subject Life Data Collection:]

A.Sex: M____ or F____ B.Age:______ C.Grade:________________

D. Nationality:______________ E. Major:______________________

I. Computer Programming Background & Experience:

(1) How many years have you computer programmed? : ________yrs

(2) When did you program using FORTRAN or C most currently?
__________month ago Date:________________

(3) What size of programming project did you get?_________lines

(4) What kind of courses for computer programming did you take?

: ____

(5) Computer types preference: ________________________
(:type of hardware, workstation)
(6) Computer languages (Which language is your best preference?
Please circle it):________________________________

(7) Software Packages: ________________________________

J. Typing ability: ___________pages/hour

K. Do you like(enjoy) a computer work or programming?

: (A B C D E)
more <-- --> less

L. Do you have any medical (physical or mental) problem?

If yes, describe:______________________________________

I hereby declare that I will honestly conduct to do my best in the experiment, and that the above is true statement.

________________________________ ________________
Signature Date

[PRELIMINARY QUESTIONNAIRES for Programming Experiment]

91P4-______ Name:______________________________ Date:___________

1. Conditions of subject environment:
   - a. Type of specific requirement:______________________________
   - b. Type of programming language:______________________________
   - c. Level of subject:____
   - d. Use of operating system:___________________________________
   - e. Use of hardware system:____________________________________

2. How much familiar(knowledgable) are you with this requirement?

   (strong, good, weak)

3. How much time did you spend to disign the program? _______hrs

4. What is your design method? ______________________________

```
5. What is your condition level? . Physical: [ A  B  C  D  E ]
                                         good <----- -----> bad
                                 . Mental    [ A  B  C  D  E ]
                                   (Psychological)
```

*6. Coding time to computer ____________minutes

*7. Mis-typing error during the edition:____________##

*8. Typing skill:___________pages/hr

*9. Special situation to subject:______________________________

____

** You will have a representational interview at each 30-45 minutes long. This session will be taken for the identification and the allocation of your common-cause errors.

Project 4-A. "HUMAN-SOFTWARE RELIABILITY EXPERIMENT"
- Optimal Sequence of Machine Replacement -

Project 4-A will involve the determination of an optimal sequence of machines to employ in providing a service for a number of years using FORTRAN or C language for the human-software reliability experiment. The development will start with a manual exercise and design the program to determine appropriate methods, then proceed with the development of FORTRAN or C program to implement the algorithm. During the programming, programmer's task behavior can be observed to find the common cause human error in human-software interaction using video camera monitor and recorder.

[Description of Problem and Requirement:]

The COST of buying a machine in the year of purchase and operating it until the year of retirement can be found thru a COST function as developed in STEP1. The COST of a sequence of machines is simply the sum of the costs of the individual machines that constitute that sequence. The COST functions employed in this assignment will be provided in a tabular form for a initial exercise and a program.

To find the optimal replacement schedule for a specified LIFE, one must consider the various replacement sequences, and select that with the lowest total cost. All costs are expressed in current (year 0) dollars, so that they may be added and compared.

During software development task, you and your observer should collect the data by observing programmer's task behavior, then the experimental data of the human-software reliability can be analyzed.

STEP1. COST Functions:

As a component of a program to find optimum replacement sequences for equipment, there is needed a function to give the total cost of a unit purchased in one specified year and retired in another. The current year is year 0 of the anticipated replacement schedule. All costs should be computed in terms of current dollars.

This program component will be developed in two steps:
  A function subprogram PRESVAL(AMT, YEAR, INT)
    where PRESVAL(REAL) = the PRESENT VALUE in dollars.
          AMT(REAL) = the amount in dollars at the future date.
          YEAR(INTEGER) = the number of years in the future that
                      the amount AMT is paid or received.
          INT(REAL) = the annual interest rate expressed as a
                      decimal fraction (for example, 12% interest
                      would be 0.12)
  A function subprogram COST(PURCHASE_YEAR, RETIRE_YEAR)
    where COST(REAL) = the total cost of the considered unit,
                      expressed in current dollars.

PURCHASE_YEAR(INTEGER) = the year number in which the considered purchase is to be made.
RETIRE_YEAR(INTEGER) = the year number in which the unit is to be retired

FORMULAE:

The PRESENT VALUE of a future amount may be derived from the compound interest formula:

```
FUTURE_VALUE = PRESENT_VALUE * (1 + INTEREST) ** NUM_OF_YEARS
```

The COST function should take into account the following items:
The equipment is purchased for a purchase price (to be asked on the screen), which is paid at the year of purchase.
When the equipment is retired, it has a salvage value which will be received at the year of retirement. This salvage value may be computed as:

```
SALVAGE_VALUE = PURCH_PRICE * (0.8 ** AGE)
```

During the unit's productive life, there will be an operating cost to be paid each year of service. (For computational purposes, assume that this is paid at the start of each year.) This operating cost increases with the age of the unit, and may be computed as:

```
OPER_COST = $1200.00 + $500 * AGE
```

The AGE of the unit is measured from the year of purchase.
The interest rate to be used will be provided on the screen.

STEP2. Optimizing Solution:

Develop a computer solution to this program, by written an optimizing subroutine. The subroutine is to have five arguments:

```
SUBROUTINE FINDOPT(LIFE, COST, UNIT, LOWCOST, LASTPUR)
```

where

LIFE = an integer variable of the number of years for which service is required. Your subroutine's algorithm will compute an optimum sequence of machines that last this long.

COST = this is not an ordinary variable, but the name of a function upon which your subroutine will call. FORTRAN allows a program to pass the name of a subprogram as an argument to another subprogram. The subprogram argument must be declared EXTERNAL in the subprogram which receive it, and this receiving subprogram may then invoke the passed subprogram under the name of the dummy argument. The algorithm to be employed will call this function to find the cost to use each machine of a series. COST is a real function, invoked for a machine purchased in year J and kept until year K as: COST(J,K).

UNIT = the unit number for output from the subroutine. This allows the main program to control where output will appear (screen of file). UNIT is an integer variable.

LOWCOST = a one-dimensional real array into which the subroutine will put the minimum cost to provide service for any number of years up to the LIFE value. LOWCOST(J) = the minimum cost of providing machines from year 0 to the start of year J. This array has a zeroth element that your subroutine should set to zero. (The optimum cost to provide a machine for 0 years is zero.) This value will be used in the algorithm.

LASTPUR = a one dimensional integer array into which the subroutine will put the year number when the last machine in an optimal sequence is to be purchased. LASTPUR(J) = the year of last purchase for the optimal sequence of machines lasting J years. This array lets you trace backwards the optimal sequence of machines.

STEP3. Main Program & Output:

Provide each question for given value for specified situation (Input values for Purchasing price, Annual interest, and Length of sequence year for simulation).
Make a main program to get a optimal output for given years with correct output formats.

[Submit the following outputs:]
(1) Program list(:.FOR or .C) including main and three subprograms.
(2) Program output(:.OUT) with the same correct formats of handout.
(3) Flowchart and raw hand-writing code for program design.
(4) Raw data collection sheets of programming task experiment.
(5) Statistical data analysis of your experimental data using given analysis form. (Mean, Variance, Percent of frequency & correction time for each mode, Regression analysis with frequency and correction time.) - by supervisor

---

[Manual Exercise for Computation:]
Given the following COST function, find the lowest total cost for a LIFE of 3 years. Tabulate your calculations for the alternatives below. Choose the sequence that results in the lowest total cost.

Sequence:	1+1+1	1+2	2+1	3
COST of Machines in Sequence (PUR,RET)	(0,1)_______	(0,1)_______	(0,2)_______	(0,3)_______
	(1,2)_______	(1,3)_______	(2,3)_______	
	(2,3)_______			
Total Cost:	_______	_______	_______	_______

[OUTPUT 1: COST Function:]

PURCH YEAR	RETIRE YEAR 1	2	3	4	5	6
0	4477.78	6934.57	8316.60	9109.16	9570.01	9840.45
1		2487.65	3852.54	4620.33	5060.64	5316.67
2			1382.03	2140.30	2566.85	2811.47
3				767.79	1189.05	1426.03
4					426.55	660.59
5						236.97

As the number of years grows, the number of alternate machine sequences becomes quite large. To provide a better strategy for the search, find the optimum as a member of a series -- if the best choices for all prior LIFEs are known, then the search for the currently-desired life may involve a much smaller number of alternatives. For example, in studying a LIFE of 6 years, one need not investigate every sequence in which the final machine is purchased in year 4, since one has already found the best way to provide service for the first four years.
Using this series approach, find the lowest costs for LIFE values up to 5 years. Record your analysis in the following table:

```
                                         ALT     LOWEST   LAST
LIFE                                     COST    COST     PURCH
1:                   The ONLY way from 0 to 1: _______      0

2:  Alternative:                COST(0,2) _______
    Alternative: LOWCOST(1) + COST(1,2) _______
                 The lowest-cost way from 0 to 2: _______  ___

3:  Alternative:                COST(0,3) _______
    Alternative: LOWCOST(1) + COST(1,3) _______
    Alternative: LOWCOST(2) + COST(2,3) _______
                 The lowest-cost way from 0 to 3: _______  ___
          ( Does this agree with your answer from Ex.1? )

4:  Alternative:                COST(0,4) _______
    Alternative: LOWCOST(1) + COST(1,4) _______
    Alternative: LOWCOST(2) + COST(2,4) _______
    Alternative: LOWCOST(3) + COST(3,4) _______
                 The lowest-cost way from 0 to 4: _______  ___

5:  Alternative:                COST(0,5) _______
    Alternative: LOWCOST(1) + COST(1,5) _______
    Alternative: LOWCOST(2) + COST(2,5) _______
    Alternative: LOWCOST(3) + COST(3,5) _______
    Alternative: LOWCOST(4) + COST(4,5) _______
                 The lowest-cost way from 0 to 5: _______  ___
```

What is the sequence of machines that will achieve this lowest cost for a LIFE of 5 years? ______________________________

Project 4-B. "HUMAN-SOFTWARE RELIABILITY EXPERIMENT"
- Optimal Inventory System and Simulation -

Project 4-B will be produced an optimal inventory policy to determine the order quantity and the reorder point for minimum inventory cost. This project will be conducted using programming language for human-software interaction and reliability experiment. Development will start with a manual exercise and design the program to determine appropriate methods, then proceed with the development of FORTRAN or C program to implement the algorithm. During the programming, programmer's task behavior can be observed to find the common cause human error in human-software interaction.

[Description of Problem and Requirement:].

Project4-B will involve the analysis of an inventory control problem and the analysis of an experiment of human-software reliability in human-software interaction systems. You will simulate the performance of an inventory management procedure under random demands, selecting the management parameters for optimum (that is, lowest cost) control. Project will be observed by supervisor using video camera monitor and recorder.

Situation in inventory system:

The inventory quantity may be positive, representing items in stock, or negative, representing unfilled orders.
The inventory control strategy is to order the REORDER QUANTITY whenever the STARTING INVENTORY for the day is at or below the REORDER POINT. This order will be delivered overnight, and will be a part of the next day's STARTING INVENTORY.

Information of external environment:

Case (Jan. 1991):

```
 MINIMUM DEMAND =  10 UNITS/DAY
 MAXIMUM DEMAND =  20 UNITS/DAY
 BEGIN INVENTORY  =  20 UNITS
 SAFETY INVENTORY LEVEL = 10 UNITS
 ORDER COST     = $100/ORDER
 HOLDING COST   =  $1/UNIT/DAY
 SHORTAGE COST = $10/UNIT/DAY
```

Three 'costs' are associated with the management function:

The ORDER COST is a fixed cost of processing an order for additional inventory, and is PER ORDER. (This is NOT the cost of the inventory itself, but the processing costs.)

The HOLDING COST is the cost of holding goods, including the costs of storage and of capital being tied up in this inventory. It is proportional to the (positive) inventory on hand.

The SHORTAGE COST is the cost of NOT being able to fill a customer's order promptly. This 'cost' is difficult to measure, being largely a loss of future business from dissatisfied customers, but a strategy that ignores this cost in its computations will invariably make this true cost large in the efforts to minimize the others. It will be approximated as proportional to each days STARTING SHORTAGE (If we can tell the customer that the desired items have already been ordered and will be in tomorrow morning, there will be no dissatisfaction.)

'EOQ' is a inventory model to determine the particular lot size that will result in the lowest value for total cost with given demand(:D), holding cost(:H), order cost(:P).

```
        Qo = SQRT( (2*P*D) / H )
```

STEP (1) Develop a subroutine for daily randum demand using randum number generator within maximum demand and minimum demand.
STEP (2) Produce a main program to solve the situation of handout with the simulation results including a order quantity and a reorder point. Make a very user-interactive program for input/output. Get a output with similar format of example.
STEP (3) Produce an alternative decision(solution) if demand will increase 20% and all of the costs will increase 10% at the next year, Jan. 1992.
STEP (4) Develop a subprogram to fine Qo with 'EOQ' model. Compare the result with previous model.

[Tasks of inventory control:]

1) Complete the formulae in each cell(####) of inventory system with the same format of handout.
2) Simulate your inventory control system by the controllable inputs (more than 5 runs in each set of controllable variables (Reorder point, fixed order quantity) and keep each total cost for the calculation of normalized cost) with multi-runs.
3) Analyze and decide your optimal inventory strategy to minimize the total inventory cost.
4) Develop your own EOQ model for order quantity instead of fixed order quantity, and simulate with this situation.
5) Compare with these two situations for your optimal inventory policy.

[Submit the following outputs:]

(1) Program list(:.FOR or .C) including main and three subprograms.
(2) Program output(:.OUT) with a similar correct formats of handout.
(3) Flowchart and raw hand-writing code for program design.
(4) Data collection sheets of programming task experiment.
(5) Statistical data analysis of your experimental data using given analysis form. (Mean, Variance, Percent of frequency & correction time for each mode, Regression analysis with frequency and correction time.) - by supervisor

```
Subj##: 91P4-B______   Name:_________________________   IMSE 91-PROJ4-B

-------- SIMULATION INPUTS ---------------- SIMULATION OUTPUT--------
EXTERNAL ENVIRONMENT:                 | ONE RUN:
 MIN DEMAND=         0 UNITS/DAY      |  AVERAGE DEMAND=   ####.## UNITS/DAY
 MAX DEMAND=         0 UNITS/DAY      |  TOT ORDER COST=   ####.## $ / MONTH
 BEGIN INV =         0 UNITS          |  TOT HOLD COST =   ####.## $ / MONTH
 SAFETY INV=         0 UNITS          |  TOT SHORT COST=   ####.## $ / MONTH
 ORDER COST=     00.00 $/ORDER        |  TOTAL COST    =   ####.## $ / MONTH
 HOLD COST =     00.00 $/UNIT/DAY     |
 SHORT COST=     00.00 $/UNIT/DAY     |----- MULTI-RUN ANALYSIS: -----------
CONTROLLABLE INPUTS:                  |  RUN COUNT     =          # RUNS
 REORDERED PT=  #### UNITS            | NORMALIZED COST(NCOST) is based on
 ORDER QUANT =  #### UNITS            |   average daily demand for month.
------------------------------        |  NORM. COST: ($/UNIT/DAY)
MULTI-RUN/INPUT SET:                  |  NCOST MEAN: ($/UNIT/DAY)
 MIN RUNS    =       5 RUNS           |  NCOST STD DEV:($/UNIT/DAY)
------------ INVENTORY SIMULATION ----------------------------------
DAY   INVENT   DEMAND    ORDER   ORDCOST   HOLDCOST   SHRTCOST
  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    ####     ####      ####   ####.##   ####.##    ####.##
 29    ####     ####      ####   ####.##   ####.##    ####.##
 30    ####     ####      ####   ####.##   ####.##    ####.##
____________________________________________________________________
Total:
```

[Subject Calibration Factors & Evaluations:]

Subject #:________ Name:____

Date:_________________ Experiment Schedule:_________________________

		Score:
1. Programming experience:		________/5
. Programming experienced years:	________/5	
. Recurrence of programming:	________/5	
. Project scale involved:	________/5	
2. Knowledge background:		________/5
. Knowledge of programming language:	________/5	
. Familiarity with hardware:	________/5	
. Familiarity with operating system:	________/5	
3. Intelligence:		________/5
. Problem solving ability:	________/5	
. Creativity of entire approach:	________/5	
. Requirement understandability:	________/5	
. Recognition of project process:	________/5	
4. Experiment attitude:		________/5
. Concentration to task:	________/5	
. Commitment to regulation:	________/5	
. Preparation effort to task:	________/5	
5. Work environmental conditions to subject:		________/5
. Entire condition of work station:	________/5	
. Noise, temperature, humidity, etc.:	________/5	
. Subject's physical conditions:	________/5	
. Extra mental, psychological stress:	________/5	
Total score:		________/25
Average score:		________/5

[Rating Analysis of Subject Calibration Factors & Evaluations:]

Name:______________________________ S.S #:____________________
(Last) (First)

* 1____2____3____4____5
poor<----- -------> strong related

1. Programming experience:		1__2__3__4__5
. Programming experienced years:	1__2__3____	
. Recurrence of programming:	1__2__3____	
. Project scale involved:	1__2__3____	
2. Knowledge background:		1__2__3__4__5
. Knowledge of programming language:	1__2__3____	
. Familiarity with hardware:	1__2__3____	
. Familiarity with operating system:	1__2__3____	
. Educational background of requirement:	1__2__3____	
3. Intelligence:		1__2__3__4__5
. Problem solving ability:	1__2__3____	
. Creativity of entire approach:	1__2__3____	
. Requirement understandability:	1__2__3____	
. Recognition of project process:	1__2__3____	
4. Experiment attitude:		1__2__3__4__5
. Concentration to task:	1__2__3____	
. Commitment to regulation:	1__2__3____	
. Preparation effort to task:	1__2__3____	
5. Work environmental conditions to subject:		1__2__3__4__5
. Entire condition of work station:	1__2__3____	
. Noise, temperature, humidity, etc.:	1__2__3____	
. Subject's physical conditions:	1__2__3____	
. Extra mental, psychological stress:	1__2__3____	