AN ANALYSIS OF THE EFFECT OF ENVIRONMENTAL AND SYSTEMS COMPLEXITY ON INFORMATION SYSTEMS FAILURES

Xiaoni Zhang

Dissertation Prepared for the Degree of

DOCTOR OF PHILOSOPHY

UNIVERSITY OF NORTH TEXAS

August 2001

Zhang, Xiaoni, An Analysis of the Effect of Environmental and Systems Complexity on Information Systems Failures. Doctor of Philosophy (Business Computer Information Systems), August 2001, 116 pages, 18 reference pages.

Companies have invested large amounts of money on information systems development. Unfortunately, not all information systems developments are successful. Software project failure is frequent and lamentable. Surveys and statistical analysis results underscore the severity and scope of software project failure. Limited research relates software structure to information systems failures. Systematic study of failure provides insights into the causes of IS failure. More importantly, it contributes to better monitoring and control of projects and enhancing the likelihood of the success of management information systems. The underlining theories and literature that contribute to the construction of theoretical framework come from general systems theory, complexity theory, and failure studies.

100 COBOL programs from a single company are used in the analysis. The program log clearly documents the date, time, and the reasons for changes to the programs. In this study the relationships among the variables of business requirements change, software complexity, program size and the error rate in each phase of software development life cycle are tested. Interpretations of the hypotheses testing are provided as well.

The data shows that analysis error and design error occur more often than programming error. Measurement criteria need to be developed at each stage of the software development cycle, especially in the early stage. The quality and reliability of software can be improved continuously. The findings from this study suggest that it is imperative to develop an adaptive system that can cope with the changes to the business environment. Further, management needs to focus on processes that improve the quality of the system design stage.

ACKNOWLEDGMENTS

I am grateful to my committee members for their comments and suggestions. I have benefited greatly from their advice. I would like to thank Dr. John Windsor for his guidance. I would like to thank Dr. Robert Pavur for generously investing his time with me and appreciate Dr. Melinda Cline for her counseling. It has been a pleasure working with my committee members.

I thank Baojian Xiang, my husband and friend. His support and encouragement has seen me through tumultuous times. This dissertation would not have been possible without my parents, Jianghe Zhang and Li Li. I am indebted to my mother for taking care of my baby so that I could have time to work on my dissertation. I have been fortunate to have brothers Sheng Zhang, Jie Zhang and sisters, Xiaowen Zhang and Xiaofang Zhang who give me support and encouragement.

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

The demand for information systems has increased steadily as companies strive to improve performance and efficiency. Companies have invested significant amounts of money in information systems development. Unfortunately, not all information systems development efforts are successful. This research explores the problems and failures of management information systems development efforts. "The short history of computing shows that technological development does not lead inevitably to successful information systems in organizations and society" (Fitzgerald, Hirschheim, Mumford, and Wood-Harper, 1985).

Management information systems are dynamic social systems (Davis et al. 1992) with many factors involved in their development and operation. Different frameworks (Lyttinen & Hirsheim 1988; Davis et al. 1992) have been developed to explore failures of management information systems. However, since IS failure is a less appealing topic of research than IS success, the academic literature on this topic is limited. Because IS failure has the potential for significant losses for companies, and IS failure occurs in companies of all sizes, it is imperative to better understand information systems failures. Findings from this study are expected to help researchers develop guidelines to increase the likelihood of information system development success.

Definition of Terms

There is no universally accepted definition of information systems failure. Lucas (1975) argues that management information systems fail if they are not used, even if they function well. Alter and Ginzberg (1978) view failure as occurring when the potential benefits of management information systems are not realized. Bailey and Pearson (1983) indicate that failure occurs when users' attitudes are negative. Markus, like Bailey and Pearson (1987), sees failure as substantial user resistance to management information systems. According to Gladden (1982), failure occurs when a functioning information system is not delivered. Lyytinen and Hirschheim (1987) argue that the widely employed concepts of IS failure are vague, point out that even those (Morgan and Soden 1973; Lucas 1975; Sanders 1984; Markus and Robey 1988; Ginzberg and Schultz 1987) systematically exploring management information systems failure use a limiting concept of failure.

In this study, failure is defined as all planned and unplanned modifications of the program code contained within the system. The term, Management Information System refers to a discrete set of application programs. Programs are a series of instructions or statements which, when decoded by a computer or a translation program cause the computer to do work (Fox 1982). A system is a set of programs that interact with each other.

Purpose of the Research

Failure has been studied more extensively in other disciplines than in management information systems. In civil engineering, the study of bridge failure has provided important information leading to the redesign of bridges (Petrowski 1985). Lessons from car crashes due to mechanic failures are incorporated into designing better, safer cars. In education, failure has been studied to improve educational practice and produce better student performance (Anderson 1985; Ives and Olson 1984; Baenen, 1992; Brookhart 1997; Elias 1998). In electrical engineering, it is well recognized that system failures require a thorough analysis (Titus 1997). We can learn from studying failure. (Ewusi-Mensah and Przasnyski 1994; Davis et al 1992; Boddie1987). A systematic study of failures provides insights into the causes of IS failure. More importantly, it contributes in better monitoring and controlling of projects and enhancing the likelihood of the success of management information systems.

This study explores the relationship between stages of the IS development life cycle and IS failures. The focus will be on finding the relationships among the internal complexity of the software system, the change in the business environment, and the frequency of failure.

Problem Motivating This Study

> "Our achievements speak for themselves. What we have to keep track of are our failures, discouragements and doubts. We tend to forget the past difficulties, the many false starts, and the painful groping. We see our past achievements as the end result of a clean forward thrust, and our present difficulties as signs of decline and decay."
>
> -Eric Hoffer Reflections on the Human Condition, p157 (1973)

Standish group (1995) provided the following statistics:

- $250 billion is spent annually on 175,000 I.T. applications in the USA
- $140 billion is wasted due to lack of best practices and

- $110 billion is the value of the work completed -- less than half of the total investment
- Canceled Projects : USA - 31%

Management information system failures are a well-acknowledged phenomenon. Due to the fear of negative publicity and other practical reasons, research on failed management information systems is less plentiful than research on successful information systems. Companies either do not feel comfortable discussing their failures or they do not have well-documented failure project. Often, companies treat failed information systems as occurrences to be forgotten. In the United Kingdom, problems with IS failure within the public sector over the last twelve years have cost the taxpayers more than £5 billion (Collins, 1994). Ewusi-Mensah and Przasnyski (1995) reported that IS failures exist in many organizations. The impact of the failure of management information systems is not limited to financial loss; it also has an emotional impact too (Boddies 1987; Gladden 1982; Glaser 1984; McFarlan 1981).

A study done by the Standish Group showed 31% of new IS projects are cancelled before completion at an estimated combined cost of $81 billion (PC Week 1994). Furthermore, 52.7% of the projects completed are 189% over budget at an additional cost of $59 billion. These statistics reflect that the software engineering industry has experienced serious problems with information systems development.

Software development problems include major system development efforts such as:

- American Airlines' failed reservation system, CONFIRM
- The 4GL New Jersey Department of Motor Vehicles System

- The NCR inventory system
- The next-generation FAA Air Traffic Control System

It is important to analyze failed IS development projects (Abdel-Hamid and Madnick 1990; Boddies 1987). Boddies (1987) indicates that “failed projects need more than explanation or rationalization if they are to teach any lessons. Projects that fail need an organized effort to uncover what really caused the failure. What these projects need is a post-mortem.”

Ewusi-Mensah and Przasnyski (1995) wrote "Individuals as well as organizations learn by examining their past mistakes, errors or failures and by taking the necessary corrective actions to forestall future recurrence of those activities under the same or similar failed outcomes in the future.” According to the authors, “A study of how and why a particular project failed will, in essence, be the best prescription against a repetition of the same or similar problems.”

Research Question

The major contribution of this study is that it explores the failure from the systems development life cycle, understanding failures from the business requirement changes. This study builds upon this work of prior authors by asking:

(1) What is the relationship between program complexity and failure?

(2) What is the relationship between the stages of the systems development life cycle and failure?

Saul (1991) and Davis et al. (1992) propose that the technology and business environment have an effect on system failure, but empirical work to validate this framework has not yet been performed. Previous research has not connected software complexity with failure. Most software failure studies focus on project management, and quite often failure is defined as time and/or budget overruns (Wright, 1998; Gary 1998). No studies have addressed software failures by using the frequency of change of the software program during various phases of the software development life cycle to better understand IS failures.

This research uses a software maintenance log to gain insight into this research question. Maintenance is an ongoing activity occurring in any organization and it represents a significant investment. This study addresses the failure issue from the software development life cycle, combining technology and business requirement changes.

Significance of the Problem

Software project failures are frequent and lamentable. Surveys and statistical analysis results manifest the severity and the scope of software project failure. It is not uncommon to find failure examples in any industry. Industry surveys from organizations such as the Standish Group, as well as statistical data from metrics gurus such as Capers Jones, Howard Rubin, Paul Strassmann, and Larry Putnam, suggest that the average project is likely to be 6 to 12 months behind schedule and 50 to 100 percent over budget. If a project starts off with high risk factors and insufficient planning, failure is likely. Even if a project begins in a reasonably calm, rational fashion, there’s a good chance that it will deteriorate into a failed project as time goes on – either because the original schedule and the budget will turn out to have been highly unrealistic, or because additional business requirements will be added to those upon which the original schedule and estimate were based.

Organizations make a substantial investment when developing information systems to support business goals. Unfortunately, even with the advancement in technology and development methodologies, management information systems development continues to fail. Beynon-Davies (1995) writes that “management information systems failure and the study of this phenomenon is important because of its apparent frequency; it is significant because they act as an important resource for validating information systems development practice.”

Typically, software development and maintenance consumes more resources than does hardware (Mellor 1987). Software is the system driver because it defines the functionality. Stories of failure attract negative publicity. Valid and quantitative methods need to be developed to evaluate the probability of system failure. Moreover, we do not know all the possible causes of software failure. Identifying the causes and factors of software failure can provide methods, techniques, and strategies to reduce the likelihood of system failure.

It is important to study management information systems failures systematically in order to improve the understanding of failures so that preventive and corrective strategies may be developed. The findings from this study will provide answers regarding whether technology and/or business requirements have an effect on IS failures. Failures can be avoided or minimized if failure factors are controlled and monitored. Monitoring failure indicators from an organizational perspective enhances the probability of management information systems success.

Software development will likely remain a chaotic process. To improve the software development process, best practices are required. Failures help uncover what the best practices may be. Best practices can take many forms. Some of these are techniques and processes that enable one to reach a goal more efficiently, and with greater success. Studying failure helps uncover the causes of information systems development problems so that we can suggest preventative measures and procedures.

Study Scope

This study only examines program modifications to a production software system. This research concentrates on the maintenance of the software. Errors and modifications occurring before the software system was installed are beyond the scope of this study.

CHAPTER 2

LITERATURE REVIEW

Introduction

This chapter examines three streams of research to build the study's research foundations: information system failures, general systems theory and software complexity. The literature review on IS failure is organized into failure concepts, the organizational and behavioral perspectives, project management, project abandonment, failure factors, and project implementation. The literature review on information systems failure summarizes previous research findings and identifies the research construct failure for this study. The literature review on general system theory develops the idea that the cause of failure can be approached from an environment variety perspective. The review on complexity theory reveals that complexity exists in many forms and that complexity may contribute to the decay of a system.

Failure Concepts

The literature on information systems development identifies many problems in the execution of system building efforts (Markus 1983; Lyytinen and Hirscheim 1987; Robey and Rodriguez-Diaz 1989). These range from overruns in project budgets and unmet schedules to the construction of systems that fail to function as designed. Although failing systems can often be rescued by allocating additional resources to them, such solutions usually reduce the return on investment.

Information systems failure has been a subject of interest for researchers for more than three decades (Argyris 1974; Locus 1975; Ginzberg 1981; Bostrom 1989; Lyytinen

1988; Davis et al 1992; Ewusi-Mensah & Przasnyski 1994; Guinan 1998). Prior to the 1970s, attention was given to the technical problems of information systems. During the 1970s, researchers also began studying managerial issues related to system development problems (Jones and McLean 1970). In the 1980s, researchers proposed quantifying software reliability. Some viewed software as passive and its failure as a response to the way it was used. Since changes in the model of use could not be foreseen, it was meaningless to measure software reliability. Others assumed that software reliability models were "black-box" models. A software failure was an unrepeatable event, and it was not possible to quantify objectively the probability of such events.

It is difficult to define exactly what constitutes a failed information system; therefore, there is no general agreement on the definition of information system failures. However, it is clear that certain characteristics are strongly related to perceived failures (Pinto and Mantel 1990). Jones defines failure as projects that are cancelled without completion due to cost or schedule overruns or that run later than planned by more than 25 percent (1996). Lyytinen and Hirscheim (1987) classify failure into four categories: correspondence failure, process failure, interaction failure, and expectation failure. These researchers developed a new concept called expectation failure to describe a gap between stakeholders' expectations expressed in some ideal or standard and the actual performance. They used a survey to collect system analysts' perceptions regarding IS failures. Their study is the first empirical study to research IS failure from the system analysts' perspective. However, the sample size of 34 made this study's external validity somewhat questionable.

Different stakeholders may have different views of information system failures (Hebert and Benbasat, 1994; Dekleva, 1992; Kling and Gerson 1977; Necco et al., 1987; Zmud, 1979). Some approach the information systems development failure from the cognitive perspective and offer a different domain of knowledge about system failures. The development of management information systems involves management, users, and IS development personnel. Understanding people's attitudes and behaviors helps explain the human factors that contribute to the failure of the information systems. Systems analysts play an important role in the process of information systems development because they contact different stakeholders and understand the needs of these stakeholders. Systems analysts' attitudes toward system development significantly affect IS quality and the success or the failure of the information systems (Bostrom & Heinen, 1977; Jiang, 1999; Lyytinen, 1988; Zmud, 1979).

Analyst attitudes toward IS development vary, and the assignment of analysts to projects could affect project success or failure (Dos Santos & Hawk 1988). Attitudinal perspective of system failures was explored by Jiang et al. (1999), motivated by previous studies of failures of managers and users. The Jiang et al. study described analyst attitudes and examined the relationships between the analysts' attitudes and the perceived reasons for IS failures. The systems analysts were divided into three different orientations: user orientation, social-political orientation, and technical orientation or some combination of the three. A survey method was used to collect data. The findings suggest that a significant relationship exists between system analysts' orientations and perceived reasons for IS failure. However, Jiang et al. do not explicitly define information system failures; they only examines the analysts' attitudes for IS development. The views of other stakeholders' groups such as the developer's, the managers' and the users' groups were not investigated. The Jiang et al. study suggests that matching analyst orientation with project characteristics is important in achieving information systems success.

The recent study by Linberg (2000) introduced a new perspective on software project failure. He surveyed software developers on a failed project that was over budget by 419%, over schedule by 193%, (27 months vs 14 months), and over size estimates by 130% for software component and 800% for its firmware components. To management, such a project is hardly something to brag about. Linberg asked the participants for their perceptions of what had happened. To his surprise, the project participants regarded this failed project as one of the most successful ones they had ever worked on because there was no catastrophic failure, because there were no post-release software defects, and because they believe it was well managed. They believed the reasons for the delay and over budget were due to poor schedule estimates, poor understanding of the problem to be solved, and poor allocation of resources.

Organizational and Behavioral Perspective

The construct of information systems development has stimulated researchers for the past three decades and a rich body of literature has been produced. Conceptual frameworks specifying the effects of organizational, task, and individual factors upon the success of IS development or implementation effort have been proposed (Ives and Olson, 1981; Lucas and Spitler, 1999). However, reviews of this body of literature reveal that the majority of studies have emphasized the effect of individual factors (Alavi & Joachimsthaler, 1992), without examining constraints that could inhibit those effects

(Masoner & Nicolaou. 1996). Many types of constrained IS development behavior exist within organizations (Masoner & Nicolaou, 1996; ; DeLone & McLean 1992; Davis et al 1992). The optimal utilization and deployment of people and methods may have the potential of alleviating common software development problems (Palvia & Hunter, 1996).

“The failure of many information systems can be attributed to nontechnical factors rather than to technical characteristics of computer equipment” (Lin &Hsieh 1995). Accordingly, different strategies should be developed to deal with the project management and risk factors. Even though user participation can be important to the success or failure of the information system, the extent to which users should participate in the project, and the way in which users participate depend upon the risk levels of specific factors.

The commitment to an IS development project is widely believed to affect the eventual success or failure of systems. The commitment is clearly important to the success of IS development projects, but managers may sometimes become too committed to certain IS projects (Keil 1995; Neumann, 1994; Orli, 1989; Rothfeder, 1987), even when faced with indications that the project may be failing. In other cases, some IS development projects may involve many time or budget overruns, or even fail, if commitment is erratic (Reich & Benbasat, 1996). The full commitment to IS development does not always result in a successful information system.

Lucas (1975) stated that “the major reason most management information systems have failed is that we have ignored organizational behavior problems in the design and operation of computer-based information systems.” It is the organization that initiates the development and implementation of the information systems. The culture of the organization, its management and external environment have direct bearing on the success or failure of the information system. Davis et al (1992) proposed information systems failure from the sociotechnical perspective. An interpretive method is employed to diagnose IS failures. Davis et al. present a two-dimensional framework: social and technical. Each dimension has 4 characteristics, resulting in 16 areas for failure diagnosis. Studying the social system and the technical system at the same time would help in the understanding of IS failures. The four components of social systems have a chain reaction. Theories-in-use by the developers and management affect the development process, which determines the performance of the information system. The performance of the information system causes varying reactions to the information system. Failure can be traced from the theories-in-use. Technology refers to hardware and software upgrades. User interfaces refer to the human computer interface, the cognitive perspective of the systems. Information requirements are task specific. Organizational fit refers to the organizational impact of the information system. Each component of the social system dimension interacts with each of the component of the technical dimensions. The technical system is concerned with the processes, tasks, and technology needed to convert inputs into outputs whereas the social system is concerned with the attitudes, skills, and value of people, the relationship among people, and the organizational structure.

Project Management

There is a sizeable body of literature studying IS project management. Examining the last project can improve the next one. Practitioners and researchers have realized that the way a project is managed affects the outcome of the IS project. Properly applying good IS development methodologies/tools does not guarantee a successful information system because of uncertainties and risks associated with every IS project. In order for the risks to be reduced, factors that cause these uncertainties must be dealt with before a development project begins. Until recently, however, the IS community expressed little interest in the discipline of formal project management, as noted by Gopal Kapur, president of the Center for Project Management, a San Ramon, California-based education, training, and consulting firm (Goff, 1993).

Keider (1984) surveyed 100 management information systems professionals on IS failure issues and stated that “although some projects fail because of technology or design problems, the main reasons for project failures indicate a lack of understanding of project management.”

In order to find causes of project failures, some researchers have addressed information systems development problems at a specific stage in the systems development life cycle. Naumann et. al. proposed a contingency approach for determining information requirements based on four risk factors: project size, degree of structure, user task comprehension, and developer task proficiency. Observing a number of IS projects over a period of 10 years, Cash, McFarlen and McKenney (1992) discovered three serious deficiencies in practice that involve both general management and IS management: (a) the failure to assess the individual project implementation risk at the time a project is funded; (b) the lack of recognition that different projects require different managerial approaches; and (c) the failure to consider the aggregate implementation risk of the portfolio of the project. According to Meredith (1988), there are many articles on project management, but few are concerned with the identification of failing projects. The project implementation risk factors listed by Cash et al. (1990) are project size, project structure, and experience with technology. Naumann's finding listed the same risk factors as Cash et al. Assessing the project risks before the planning stage is critical to the success of an IS development project.

Martinez (1994) studied the challenges facing large-scale IS projects. Due to the size of an IS project, it often fails to meet deadline, and budget, and goes out of control. The basic functions of project management are the essentials to ensure project success. Identifying and monitoring essential project management functions increases the probability of project success and enhance earnest performance. Essential functions may be classified as executive, project, team, and analyst/doer. A comprehensive understanding of the various functions in categories will improve the project's potential for success. The steps suggested for improving the success rate of large-scale IS projects include (a) start with project scoping and planning, (b) assess culture and values, (c) clearly articulate the business vision, (d) develop a communications plan, establishing tracking systems, and (e) implement an administrative support/logistics plan.

The ideas and research concerning project failures in other fields can be used as references for conducting failure research in the IS field. Pinto and Mantel (1990) found three contingency variables associated with project failures, the precise way in which failure was defined; the type of project; and the stage of the project in its life cycle. The benchmarks used to assess the success or failure of a project are the implementation process itself, the perceived value of the project, and client satisfaction with the delivered project. According to Pinto and Mantel, failure is a multidimensional construct, encompassing both internal efficiency and external effectiveness.

Project management tools play a role in IS management as increasing numbers of companies are using these tools. One study reported the results of a nationwide survey of 1,000 randomly selected members of the Project Management Institute (Fox & Spence, 1998). The results include the identification of tools used, the level of use, the types of uses, the satisfaction with the tools employed, the level of training received, and the adequacy of the tool's use. The respondents listed not only traditional project management tools, but also several nontraditional tools. In general, project managers appeared to be satisfied with the tools available. However, sophisticated software tools do not ensure project success.

The qualitative and quantitative aspects of IS project management are brought together in a complementary manner under the umbrella of synergism. The quantitative output from group synergy is a structured and on-task team with synchronized productivity. Qualitative factors in group synergy include morale, supportiveness, participation, coordination, integration, and commitment. Qualitative and quantitative aspects of group synergy are complementary to each other, and together they can generate greater effectiveness and efficiency in IS project management. Being equipped with a broader framework of synergism, IS professionals can perform better in the field of systems development (Lai, 1997).

A formal methodology plays a role in IS development. A formal systems development methodology cannot be seen as the critical success factor for information systems (Skok & Scarre, 1992). The use of a formal methodology encourages good documentation, and quality assurance reviews. Also, it helps inexperienced staff to understand which tasks precede which, and helps project managers to control projects, even with rudimentary PC software tools. However, there are a number of drawbacks to the use of a methodology. The main difficulties are the time taken for its introduction into an IS department and the associated culture changes. There may be a substantive learning curve and high associated training costs.

In project management, integration needs to be emphasized. Systems should provide functions that seamlessly work together and leverage their strengths. Integration of point solutions is the most expensive component of systems management. End-to-end management to cover all aspects of the infrastructure such as networks, systems, DBMSs, and applications. Enterprise management is composed of the total manageability of all networks, systems, databases, and application resources that constitute the delivery of services and products.

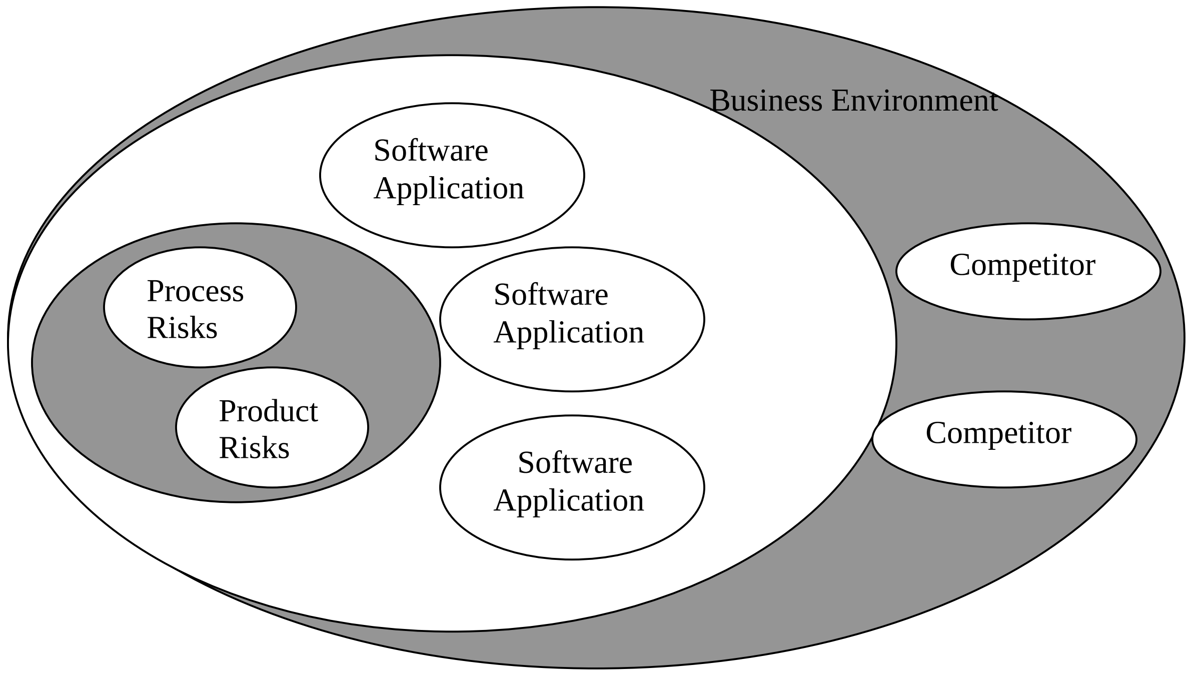

Figure 2.1 The scope of project risks

Project Abandonment

Project abandonment is one form of IS failures. Factors that may cause project abandonment, include an economic slow down, legal issues, or technology innovation. Ewusi-Mensah and Przasnyski (1991) perform an exploratory study on IS project abandonment. Two categories of IS abandonment were identified: abandonment of projects under development and abandonment of existing systems in operation. The research subjects were top executives from the Directory of Top Computer Executives. A questionnaire was used to collect data with a response rate of 8.7%. The findings suggest that many IS projects are susceptible to being abandoned--not necessarily just high-risk, complex, or unstructured ones. Project abandonment is more related to organizational, behavioral, political issues, and less related to economic and technical issues. Ewusi-

Mensah and Przasnyski’s failure factors fall into the same general categories as those in Garrison’s (1980) study.

Management and end-users are found to be the factors deciding the fate of the project (Ewuksi-Mensah & Przasnyski 1994). In Ewusi-Mensah and Przasnyski's (1995) followup study "Learning From Abandoned Information Systems Development Projects", a survey was mailed to top computer executives in Fortune 500 companies. The major concerns of their study were to address whether organizations keep records of abandoned IS projects, what they do with those records, and what they learn from the abandoned projects. Based on a 5.6% response rate, 60% of the respondents replied that more than one project was abandoned for the same reason. More than 70% responded that no records were kept for the abandoned projects.

The failure of organizations to document their project failures and use that information to avoid a repetition of similar problems, perhaps, more than any single fact attests to the continuing problem of IS development projects abandonment in organizations. Two problems are identified in this research: 1) a majority of the companies don't document the abandoned projects; for those that keep records of the abandoned projects, only IS management and developers consult the records, 2) the records weren't used as often as expected. Other groups of people such as senior management, programmers and new project leaders need to spend time on those records.

Research indicates that IS failures occur with some regularity in companies of all sizes. Therefore, it is apparent that such cases are an industry wide problem, despite the significant progress made in IS development methodologies and tools since the early days of business computing almost 4 decades ago. The cancellation of projects may be attributed to a combination of several factors, including the following: (a) project goals, (b) project team composition, (c) project management and control, (d) technical know-how, (e) technology base or infrastructure, (f) senior management involvement, and (g) escalating project cost and time of completion.

Failure Factors

The search for factors that affect project success or failure has been of great interest to both researchers and practitioners. Unfortunately, there are few extensive studies on IS failure factors. Previous failure models consider simple factors (Lyytinen & Hirschheim 1987). As early as 1973 Morgan and Soden concluded that “almost all the failures were of management and personnel, rather than technology” based on their small sample. Keider's survey results support Morgan and Soden's conclusion that, "although some projects fail because of technology or design problems, the main reasons are within the control of the project manager."

Lyytinen and Hirschheim (1988) identified 12 factors related to IS failures. These include technical and operational factors: individual, organizational, environment, method-based, decision-based, work-based, contingency, implementation, system assumption-based, analyst-based, and user-based factors. However, Lyytinen and Hirschheim also suggested that there is strong multicollinearity between these factors. A failure factor in one domain can cause failure type in other domain.

According to Block's failure classification and causes, one cause of failure can contribute to several failure types. Block's failure classification provides different aspects of failures. This kind of classification is still very useful today.

Table 2.1

Block's (1983) Classification of Failure and Cause

Failure	Cause
Resource failures	Conflicts of people, time and project scope due to insufficient personnel
Requirement failures	Poor specification of requirements
Goal failures	Inadequate statement of system goals by management
Technique failures	Failure to use effective software development approaches, such as structured analysis/design
User contact failures	Inability to communicate with the system user
Organizational failures	Poor organizational structure, lack of leadership, or excessive span of control
Technology failures	Failure of hardware/software to meet specifications; failure of the vendor to deliver on time, or unreliable products
Size failures	When projects are too large, their complexity pushes the organization's systems development capabilities beyond reasonable limits
People management failures	Lack of effort, stifled creativity, and antagonistic attitudes cause failures
Methodology failures	Failure to perform the activities needed while unnecessary activities are performed
Planning and control failures	Vague assignments, inadequate project management and tracking tools
Personality failures	People clashes

Large-scale IS projects not only are expensive, but they also have a high failure rate. Factors contributing to the failures of large-scale IS projects have been identified by Jones (1996) from the technical point of view. The project size, the nature of the industry, and the tools utilized on project development are associated with project failures. Jones proposed the factors related to IS failures, but no particular research method was employed. The conclusion was drawn from his own consulting experience. Some industries are doing better than others in managing large software systems. Six subindustries within the software community are classified as systems software, outsource vendors, commercial software, management information software, and end-user software. All six subindustries have a fairly good success rate. Tools used in IS development can cause success or failure. Successful projects all use project management tools and quality assurance tools during the entire development process to ensure the quality of the product and meet the goal of the project, tools alone do not guarantee the success of the project. Capable managers and technical personnel are also important to the fate of the project.

The heuristic of project failures was investigated by McComb and Smith (1991). Their study focused on systems projects that began well and finished less successfully. While McComb and Smith claimed that most IS managers were familiar with IS failures, they did not provide a definition of failure. Throughout the article, failure meant running over time, over budget or not meeting user expectations. Fifteen failure factors were discussed along with planning versus executing and technical versus human dimensions. Although real world examples were provided under some factors to explain the necessity of the factors, there were no hypotheses developed to test each factor. There was no empirical work done regarding the failure factors. McComb and Smith’s framework included comprehensive factors within an organization, but they failed to consider the effect of the external environment on the information systems. Using McComb and Smith’s framework as a guideline, IS managers can broaden their perspective on the source of potential problems and prevent unnecessary project failures.

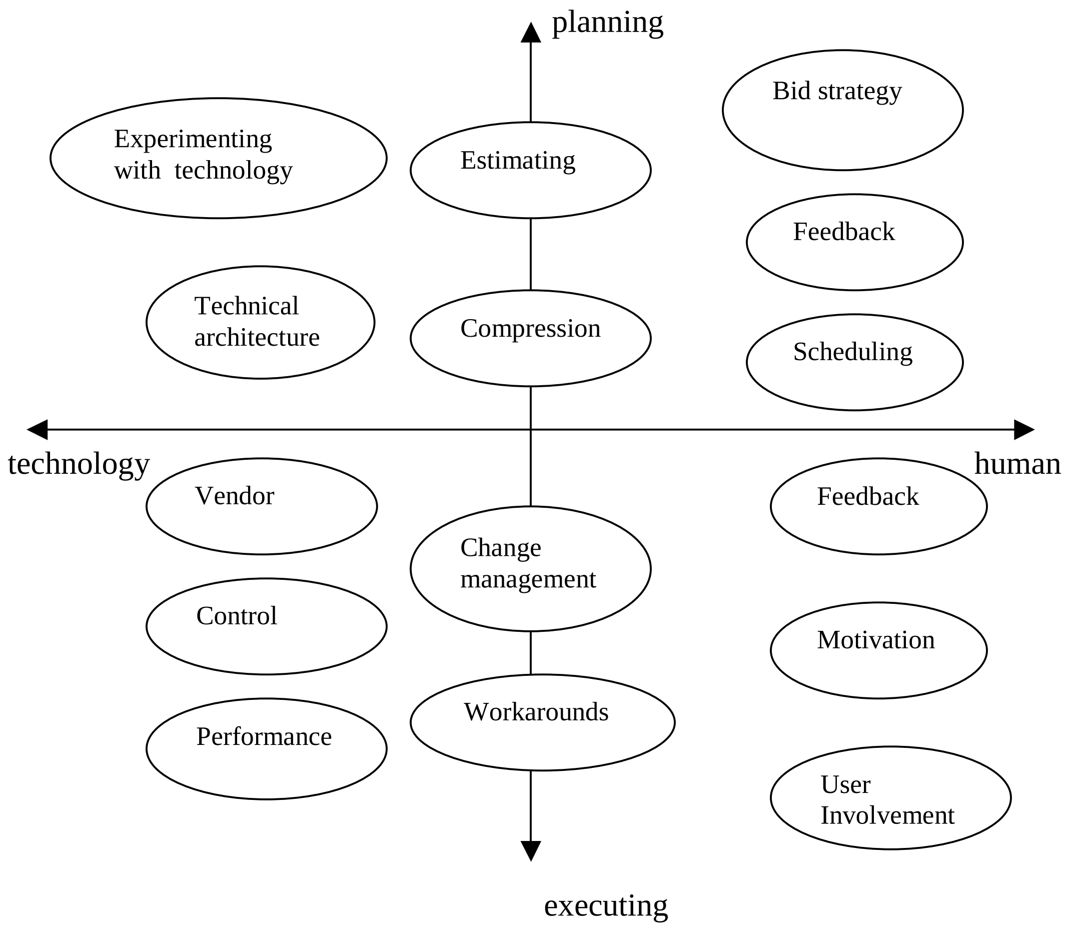

Figure 2.2 McComb & Smith framework, technical vs human, planning vs management.

Fifteen factors are located in different matrices. Failure factors related to project development are technology, leading edge system, poor consultation and change requirements, weak procurement, and project timetable slippage. Failure factors related to implementation are inadequate testing and poor training (Flowers 1997).

Much can be learned from the most catastrophic software development failures of all time. In Software Runaways, software failure expert Robert Glass (1997) shows exactly what went wrong in 16 colossal software disasters. Runaway software projects are generally caused by one of the following factors: changing specifications, bad planning, new technology, bad project management, or inexperienced staff (Glass 1997). Furthermore, he presented specific lessons learned from each failure, and showed how to "sniff out" runaway projects early enough to take action. He also considered the typical responses to potential runaways, including risk management and issue management, demonstrating their strengths and weaknesses.

Understanding the projects undertaken and thoroughly analyzing the solutions help to prevent software runaways. The better project teams understand the projects and the requirements, the better they are able to provide the right solutions, and the more they can plan for future problems. Good planning at the beginning is an important predictor of success or failure.

The reasons for death march projects are identified by Yourdon as: politics; naïve promises made by marketing , senior executives, naïve project managers; naïve optimism of youth; start-up mentality of fledging; the marine corps mentality; intense competition caused by globalization of markets; intense competition caused by the appearance of new technologies; intense pressure caused by unexpected government regulations; unexpected and unplanned crises, your hardware software vendor just went bankrupt and programmer’s passing away or quitting.

IS Implementation

Abreu and Conrath (1993) developed an integrative expectancy model based on factor studies, process studies, and expectancy studies. Their study did not empirically test the model they proposed. They suggested that expectations could be used as a predictor of IS implementation success/failure. The multiperspective approach may be used to consider stakeholders' different views of the process. They recognized a multiplicity of outcomes of the implementation process.

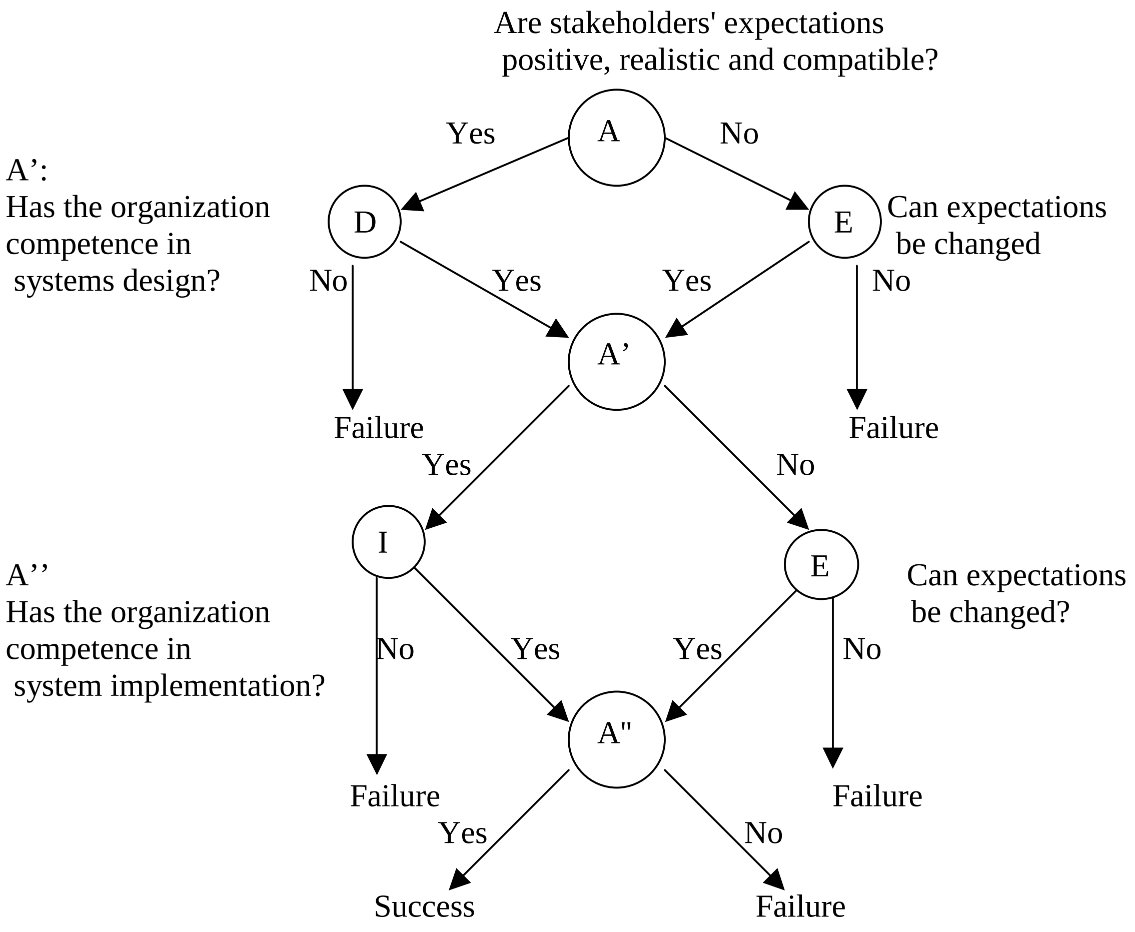
Are stakeholders' expectations positive, realistic and compatible?
A
Yes
No
A':
Has the organization competence in systems design?
D
E
Can expectations be changed
No
Yes
Yes
No
A'
Failure
Failure
Yes
No
I
E
A''
Has the organization competence in system implementation?
Can expectations be changed?
No
Yes
Yes
No
A"
Failure
Failure
Yes
No
Success
Failure

Figure 2.3 Expectancy Model. Stakeholders expectation affect the success or failure of IS project.

Information systems development has both economic and psychological determinants. For small businesses, IS procurement is often the largest, single capital expenditure (McWilliams, 1994). The identification of need and system selection and implementation also require technical knowledge and expertise that are often supplied by management, an employee, or a consultant (Geisler & Hoang, 1992). Technical knowledge requires learning and training, which in turn requires time and effort, as well as monetary expenditure. When the number of employees have created an increasing number of interactions with the information systems, the economic cost of switching between different types of systems rises. Some information systems have to be replaced if they cannot meet the business needs (Geisler & Hoang, 1992; McWilliams, 1994).

Different activities are performed in different stages of the software development life cycle. Each stage has a varying effect on the systems maintenance. Too often, fixing programs occurs late in the software development life cycle when changes are costly and less effective. It has been suggested that early risk-avoidance techniques lead to less maintenance and ongoing changes of the information systems. (Alder & Nordgren 1999).

Complexity Theory

Complexity Definition

Something is considered to be complex when it has many parts and relations. According to Brewer (1973), complexity increases as the number of "interactions . . . through the relationships" (Flood et al., 1993) increases. The ranges of complexity identified by Shannon and Weaver in 1949 are disorganized complexity, organized complexity, and organized simplicity. There is a tendency for systems to become disorganized. Complex systems are in constant transitions, self-creating small levels of order, and then adapting and changing again. Complexity theory views information systems in ways that are organic, nonlinear, and holistic. The principle of complexity theory suggests that small changes may have large effects on the systems; thus, interesting and unpredictable properties may be expected to emerge from a system.

Complexity may be understood as a combination of variety and constraint. Variety is a measure of freedom or diversity of the number of distinct possibilities or alternatives. Variety on its own leads to entropy, disorder, and chaos. Constraint limits possibilities by excluding certain alternatives. Constraint engenders order, stability, and rigidity. A system may be defined as a constraint on variety. If variety and/or constraint increase, the system's complexity increases.

In 1985, Ramamoorthy, Tsai, and Bhide pointed out that software complexity was the major reason for rapidly increasing software development costs. Some of the best-known complexity measures today are McCabe's measure of lines of code (1976), and Halstead's measures of length, volume, difficulty and effort. Other complexity measures are data flow and information flow. There are more than 100 other measures for the complexity of programs (Zuse, 1997). Ramamoorthy et al. defined software complexity as the degree or difficulty in analysis, testing, design, and implementation of software. According to Sheppard", complexity is a metaphysical property and thus not directly observable."

Software has unique characteristics. It is not a physical product, and it is difficult to measure because its components are hidden. There are many sources of software complexity. Complexity is an abstract property of software. The sources of complexity come from, control structure, size of code, information content, modularity, and data structure. While many software complexity metrics have been developed. Gonzalez (1995) argued that most of them are incomplete because they measure only one factor of program complexity, even though a program has different sources of complexity. And, the combination of two complexity metrics does not necessarily offer additional information about software (Munson & Khoshgoftar 1992).

Basili (1980) defined software complexity as "a measure of the resources expended by another system while interacting with a piece of software. If the interacting system is people, the measures are concerned with human efforts to comprehend, to maintain, to change, to test, ..., that software." Curtis et al. (1979) similarly defined this concept as psychological complexity: "psychological complexity refers to characteristics of software which make it difficult to understand and work with." Both of these authors noted that the lack of use of structured programming techniques is believed to increase the cognitive load on a software maintainer (Banker, Srikant, Chris & Zweig 1993).

There are three types of complexity: computational, psychological and representational complexity (Curtis 1979; Cant & Henderson-Sellers, 1995). Of these three complexities, Zuse (1990) suggests that psychological complexity is the most important. It is suggested that exiting design and coding complexity measures do not provide an absolute rating, rather they should be evaluated relative to the problem complexity (Cant & Henderson-Sellers, 1995). Kitchenham, Pfleeger, and Fenton (1997) observed that complexity can be viewed from a number of different perspectives including program algorithms, and structural and cognitive complexity contexts. With the existence of multiple metrics, McDonell (1991) evaluated the rigor in software complexity measurement experiment and provided suggestions for software complexity measurement.

Table 2.2 Gonzalez’s (1995) complexity definition

the nature of the problem
the number of the objects created
the interfaces and relationships
the algorithm and logic structure
methods and tools used
programming languages used
the level of human expertise
the hardware and software environment
project organization
project management

Software complexity is objectively aimed to quantify a program, based on the degree of presence or absence of certain characteristics of software. Software complexity is related to such features of software as number of errors left in the software, effort to design, test, or maintain a software product, development time, and maintenance cost. The importance of software complexity lies in the fact that knowing the complexity of a specific software product or its module enables one to: (a) predict the cost of testing, maintenance, etc., number of errors left in the software, size of the final program; (b) assess the development time, effort, cost, etc.; (c) identify critical modules or parts of the software; (d) compare programs, modules, programmers, etc. according to software complexity.

Complexity Metrics

Software metrics measure the attributes of software programs. The common software complexity measures are cyclomatic number (McCabe, 1976), software science (Halstead, 1977), lines of code (Mills 1983), number of procedures and I/O statements (McTap, 1980), and level of nesting (Harrison & Magel, 1981). There is no universally accepted measure to ensure that a software complexity metric measures complexity because there is no generally accepted definition of complexity. Furthermore, researchers have found high correlations among the software metrics (Crawford, 1985; Feurer & Fowlkes, 1979; Coupal & Robillard, 1990).

A factor analysis was performed to analyze the software metrics by Mata-Toledo and Gustafson (1992). A set of Pascal programs was analyzed for the number of decisions, the number of I/O statements, the Halstead count, the number of procedures, and the number of lines of code. Their findings suggest that a program should be normalized to avoid high correlation between program size and maintenance frequency. Dividing by lines of code appears to be the best normalization. An inverse relationship exists between the normalized number of decisions and both the normalized unique operand counts and the number of decisions.

Other researchers have approached software complexity problems from a different point of view. Khoshgoftaar and Lanning (1995) developed a neural network model to classify program modules as either high-risk or low-risk based. The inputs to the model include a selection of software complexity metrics collected from a telecommunications system. Two criteria variables are used for class determination: (a) the number of changes to enhance the program modules and (b) the number of changes required to remove faults from the modules.

With many program complexity metrics available, it is difficult to rank programs by complexity--the different metrics can give different indications. Comparison is offered through two methods of simultaneously detecting those aspects of software complexity measured by the Halstead (1977) metrics and the McCabe (1976) cyclomatic complexity number. The first method synthesizes a combined metric by weighting each Halstead metric to reflect program attributes measured by the cyclomatic complexity number. The second method uses principal components analysis. This method derives a relative complexity metric, which represents each complexity metric in proportion to the amount of unique variation that it contributes. A validation study establishes a useful statistical relationship between this relative complexity metric and faults for a military telecommunications development effort.

In object-oriented environment, Chidamber and Kemerer's (1994) metrics are the most well-known software metrics. Measuring the aspects of software complexity for object-oriented software helps to improve the quality of such systems during their development, while especially focusing on reusability and maintainability. Chidamber and Kemerer's metrics evaluate the complexity of the classes in terms of internal, inheritance, and coupling complexity.

The methodical approach in information systems development can also affect the flexibility of IS developed and the ability to cope with the dynamic world. As the complexity of IS evolves, not all information systems are able to deal with the challenge of evolutionary complexity (Lycett, Mark. & Paul, 1999).

The relationship of software complexity and maintenance has been investigated by many researchers (Schneberger, 1997; Banker et al, 1998). Component simplicity and system complexity have a direct bearing on the maintenance of the information systems. The smaller (but more numerous) the system components, the easier they are to deal with individually, but the more difficult it is to deal with the overall system.

Software maintenance is the last stage of the software development life cycle. This stage can consume a large proportion of organizational resources. It is believed that many repairs of the systems derive from inadequate work done at different stages of the software development life cycle. Poor design, analysis and coding can result in complex software that is costly to support and difficult to change. It is difficult to assess the actual maintenance performance because the causes of maintenance are realized over the software life cycle. To estimate the impact of different stages of the entire life cycle on maintenance, a two-stage model was developed using software complexity as a key intermediate variable that links design and development decisions to the effects on software maintenance. The results suggest that an important link exists between software development practices and maintenance performance (Banker, Davis & Slaughter, 1998).

In terms of the maintenance problems, some suggest that methodology has an effect on the maintenance. Dekleva's (1992) findings suggest that modern information systems development methodology does not necessarily decrease the maintenance time. It is found that the use of modern methodology can decrease the time spent on emergency error correction as well as systems failures.

General Systems Theory

The society of General Systems Theory was founded by scientists from different disciplines, biologist Bertalanffy L.V., economist Kenneth Boulding, biomathematician Anatol Rtapoport, and physiologist Ralph Gerard. The purposes of the society for General Systems Theory are to: (a) examine the isomorphy of concepts, laws, and models in different fields and transfer the concepts from one discipline to another; (b) encourage the development of adequate theoretical models in the fields that lack them; (c) minimize the duplication of theoretical efforts in different fields; and (d) form the unity of science through improving communication among specialists.

Boulding (1956) classifies systems into nine categories, based on the level of complexity: frameworks, clockworks, cybernetics, open systems, genetic-societal, animal, human, social organization, and transcendental. Ashby's (1956) law of requisite variety (1956) states that the regulator must have as much variety or more variety than the system it regulates. Control of a system depends on the variety of the regulator and the capacity of the channel between the regulator and the system. It is costly, both in time and resources, to obey this law. Ashby's (1956) law of requisite variety states that "only variety can destroy variety." An application of Ashby's law shows that system variety cannot handle environmental variety, the system will fail.

The systems approach of developing information systems originates from Bertalanffy's (1968) general system theory. The system refers to "the abstract idea of a whole having emergent properties, a layered structure, and processes of communication and control which in principle enable it to survive in a changing environment" (Bertalanffy 1968). Such a view considers the system as a whole, instead of looking at the individual function, or component of the system in isolation. According to the system approach, the IS developer has to consider the effect that the change in one part of the system would have on the rest of that system.

Checkland (1989) developed a soft systems methodology based on general systems theory as an alternative approach for coping with a major problem in systems engineering -- that of soft, or ill-defined, problem situations. It is hypothesized that the only way to develop an information system effectively is by interacting with the real problem situation, and by examining the "whats" and "hows" of the problem, before any attempt is made at a solution. This methodology is useful in improving the understanding of the problem situation during the process of requirements definition. However, Walters & Broady (1994) argue that the soft systems methodology does not provide adequate support for the actual design of a new system, or for its implementation.

In recent years, information systems development has incorporated general systems theory. The usefulness of the Viable System Model (VSM) in information systems projects was proposed by Kawalek & Wastell (1999). The VSM is a rigorous organizational model explaining information systems development from a general systems perspective and it can be used for diagnostic purpose in IS development.

Technology has been undergoing rapid development. Hardware and software have been kept upgrading. Businesses have to employ the newest technology to operate in competitive environment. Hardware and software change are common for any given systems. Today’s business environment is dynamic, so businesses have to modify their requirements to keep with the environment change.

Brooke’s et al. (1998) study took a systems approach to study information systems failures. Information system failures were examined at both the technology and the business requirement levels. They questioned the established practice in the design and implementation of information systems and focused on the systems development process and the interactions that take place between the various process participants. They argue that systems development activities are interest-based in nature to the extent that an “unholy alliance” is struck between interested parties and that ignoring the social and organizational issue makes the method to development mechanistic.

There is a growing interest in the use of soft systems methodology (SSM) in work-related to computer-based information systems. An effort has been made to identify and define, a role for SSM in information systems development that stems from its fundamental principles. Since SSM sees computer-based information systems as systems that serve purposeful human action, the notion of "information system" in SSM is one that necessarily involves two systems--a serving system and a served system of purposeful action. (Winter & Brown, 1995).

Software Reliability and Maintenance

An extensive body of research stream has focused on software reliability studies for the past two decades and many software reliability growth models have been developed for the estimation of software reliability and the number of errors embedded in software (Goel and Okumoto, 1979; Pham 1993; Ohba, 1984; Misra 1983; Hossain and Dahiya 1993, Yamada and Osake 1985; Zhang and Hoang, 1998). Exhaustive software testing can reduce the errors in software but also increase costs in software development. When to stop testing and to release software and at the same time guarantee optimal performance of the software and user satisfaction have been the greatest challenge for the software industry. Most of the reliability models focused on the software reliability before the software was released. It is well acknowledged that the cost is exponentially increasing with the amount of the time and effort spent on testing software once the software reaches a certain level of perfection. Due to various factors, it is quite common fro software to have defects after it is released. Defects may be caused by the insufficient design and analysis or maybe the coding process. The purpose of a software reliability studies are to make software more dependable and reliable and less prone to error. Most studies of this field have developed statistical models to predict the distribution of failures (Bunday and Al-Ayoubi 1990). Poisson-type distribution models are widely used in software reliability and reliability growth modeling

Software cost models address the cost/benefit issues of software development. (Kapur and Bhalla,1992; Leung, 1992; Dalal and Mallows 1988; Dalal and Mallows 1992; Ehrlich and Prasanna and Stampfel and Wu, 1993). These models are constructed to predict the set-up, warranty, risk, testing and repair costs. Software testing consumes substantial resources, such as man-power, CPU hours and test cases in order to detect and remove errors in software programs. (Yamada and Ohitera 1990).

Software reliability is one of the evaluation characteristics of ISO 9126 and it is often recognized. Many models and techniques have been proposed to quantitatively represent software reliability. Faced with the practical problems of reliability models and experiences in practice, Bazzana et al (1993) applied an empirical industrial dataset to existing software reliability models. They used graphics to aid the explanation of error-fault-failure taxonomy. They state, “an error is a human action that results in software containing a fault and fault is a manifestation of an error in software. A fault encountered may cause a failure. Failure is the inability of a system to perform required functions within specified limits."

Errors can cause failures and can happen in different phases of software development. A large industrial failure data was used to validate the software reliability models and evaluate the predictive capability of the models. Several models, Jelinski-Moranda, Goel-Okumoto, Musa-Okumoto, Duane, Littlewood, Littlewood Non-Homogeneous Poisson Process, Littlewood-Verral and Keiller-Littlewood were selected for the testing. It was concluded that no one model performed best for all the phases, system testing, and operation. Different models have different strengths in terms of operation, quality and operation. The predictive power of a particular model depends on the model selected and the quality of data analysis.

Inter-failure interval is used as a variable to test software reliability in Bazzana (1993) et al's study. Cumulative failure rates have been of interest and explored extensively in software reliability models. (Goel and Okumoto, 1979; Pham 1993; Ohba, 1984; Misra 1983; Hossain and Dahiya 1993, Yamada and Osake 1985; Zhang and Pham, 1998).

Software maintenance is the modification of a software product after delivery to correct faults, to improve performance or other attributes, and to enhance the product by adapting it to a modified environment (IEEE Standarad for Software Maintenance 1993). The high cost of software during its life cycle is largely attributable to software maintenance activities. Yau and Collofello (1985) discussed an approach to reducing maintenance costs through the utilization of metrics. Design stability measures indicate the potential ripple effect characteristics due to modifications of the program at the design level.

Much software maintenance research links soft maintenance to the cost of the system and attempts to build a model to predict the maintenance cost (Banker and Slaughter 1993; Kemeter and Slaughter 1997). Aging software may require maintenance for regulatory reasons or to make the software more interoperable with new packaged applications. Kafura and Reddy (1987) describe seven different software complexity metrics andrelate them to maintenance activities performed on a medium-size software system. They demonstrate the usefulness of software metrics in the redesign phase by revealing a poorly structured component of the system.

Yau et al (1988) presents an integrated life-cycle model for use in a software maintenance environment. The proposed model, which represents information about the development and maintenance of software systems, emphasizes relationships between different phases of the software life cycle and provides the basis for automated tools to assist maintenance personnel in making changes to existing software systems. The model is independent of particular specification, design, and programming languages because it represents only certain fundamental semantic properties of software systems, such as control flow, data flow, and data structure. The software development processes, by which one phase of the software life cycle is derived from another, are represented by graph rewriting rules that indicate how various software system components have been implemented. The approach permits analysis of the fundamental properties of a software system throughout the software life cycle.

Software Volatility

Software are social systems which evolve constantly (Davis et al 1992). Literature has addressed the changeable nature of software for three decades in the name of maintenance, debugging, and improving reliability. However, software volatility has recently received attention by a few researchers (Barry, 2000; Butcher, 1997; Heales, 2000’ Banker, 2000). This theory holds that software is changeable. Software development phases involve changeable factors, such as the requirements change, the turnover of the development team, hardware and software change or maybe the management change. The evolution of software is doomed by its environment and its purpose to serve the business functions and goals.

Different definitions of software volatility have been defined. Barry and Slaughter examine and develop software volatility measurements based on three dimensions: amplitude, periodicity and deviation. “Amplitude measures the size of software modifications made to a system” (Barry and Slaughter 2000). This has been operationalized as the total size of software modifications divided by the total system size. Periodicity measures time between software modifications. In Barry and Slaughter’s study, periodicity is normalized by dividing mean time of between system modifications by system age. Deviation is the variance of the time between system modifications. Periodicity and deviation are derived from the same variable, time between system modifications. Software volatility measures are aggregate measures of longitudinal behavior. By combining the measures Barry and Slaughter develop a predictive model of software complexity.

Factors affecting information system volatility have been explored by Heales (2000). Volatility in this study means the propensity of an information system to change its state from evolution to revolution. Four factors are identified to influence the volatility of the software system: semantic relativism, size, age, and period. Heales's study combines a model developed in 1998 and Wand and Weber's model to examine information systems changes. Heales's study suggests that more effort put on deep structure change would result in a reduction in the volatility index. The high level language is associated with low volatility index which correspondingly lead to low cost in fixing the program. Heales further provides implications from his study along management, staff perspective, and selection of tools and techniques.

It is well recognized that the cost of enhancing software applications to adapt to user requirements is growing and significant (Kemerer and Slaughter 1997; Banker and Slaughter 1993, 2000). Banker and Slaughter developed a software enhancement model to illustrate the relationship among software volatility, total data complexity and enhancement costs and errors. Structure is introduced as a moderator factor. Empirical data from two field studies are used to test the model. In Banker and Slaughter's study, volatility is defined as the frequency of enhancement per unit of functionality. They find the optimal level of structure increases with volatility and complexity.

Researchers in this area have employed the common methodology to study software volatility. Predicting volatility of software appears to be the common interest. Different variables are identified and selected to predict software volatility. All three existing studies use multiple regression models to perform data analysis.

Software Development Life Cycle

Software development goes through analysis, design, development and maintenance stages. Any errors that occur in any of these phases will affect the performance of the systems and incur costs in keeping the system up and running and meeting user's requirements. The eventual performance of the system is closely related to each phase of software development life cycle.

Research has identified design faults contribute to software failure (Mellor 1987, Zhang and Pham 1998). Numerous real life catastrophes also provide evidence that design error cause not only financial loss but also the loss of human life. One such disaster occurred on June 4, 1996, when an explosion ended the maiden flight of the European Space Agency's Ariane 5 rocket less than 40 seconds after lift off, due to the Ariane 5's flight control system. This disaster was caused by a few lines of Ada code containing three unprotected variables. Zhang and Pham (1998 and 2000) state design error and insufficient software testing are possible causes of disastrous failures.

The principle of total quality management in the Information Systems (IS) environment focuses on zero defects in a software development to satisfy customers and meet business objectives. Huq (2000) emphasizes that total quality management in the software development life cycle should be enforced in each phase of software development. Huq's study suggests that near zero defects can be achieved through testing after each phase (concurrently), of the development life cycle, as opposed to testing only after the entire system is complete. The ability to detect errors using measures in the early phases of the software development life-cycle leads to better management and quality of the systems in the later phases. Effective quality assessment in each phase of the software development life cycle is preventive (Briand, Morasca and Basili 1999). Software inspection and testing conducted together with previews and postmortems within each phase of the software development life cycle are widely recognized as necessities for building a solid foundation for defect prevention and quality enhancement.

Hong and Xie (1999) propose a statistical method to control the software defects detection process and to provide defect prevention analysis. Adler, Leonard and Nordgren (1999) investigate software reliability using a risk management framework which emphasizes preventive risk management in early phase of the software development. In the later phase of the software development life cycle, changes are costly and less effective.

Summary

Both trade journals and academic journals have discussed the factors that are attributable to software failures. Software failures have been examined from the following perspectives: the system development life cycle, tools used, management, stakeholders, users, project management strategies and development methodologies. The following chapter develops a research framework using the constructs identified from the literature.

CHAPTER 3

RESEARCH FRAMEWORK

Introduction

This chapter presents the conceptual model and the research framework. The conceptual model describes the breadth of the research and the research framework narrows the conceptual framework to specific research constructs. Construct definitions and operationalization of the constructs are discussed next.

Theoretical Framework

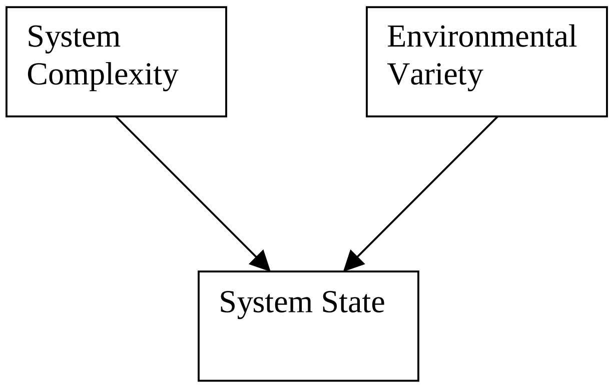

Figure 3.1 Theoretical Framework.

The research framework borrows ideas from complexity theory; general systems theory and information systems change theory. It is constructed by combining complexity theory's notion of software *complexity*, general systems theory's definition of *environmental variety*, and information systems change theory's concept of *system state*. The change to business requirements is used to operationalize system state. Several researchers claim that information system change requests are the accepted method for initiating and managing information system changes (Arthur 1988; Martin and McClure 1983; Wand and Weber 1995; Heales 2000). This research is focused on how environmental variety and the software's internal complexity affect system state. The frameworks of Davis et al.(1992) and Sauer (1991) address the social and technological impact on systems failure. The framework used in this study also considers the impact of technology and business requirements on the stability of information systems.

Research Framework

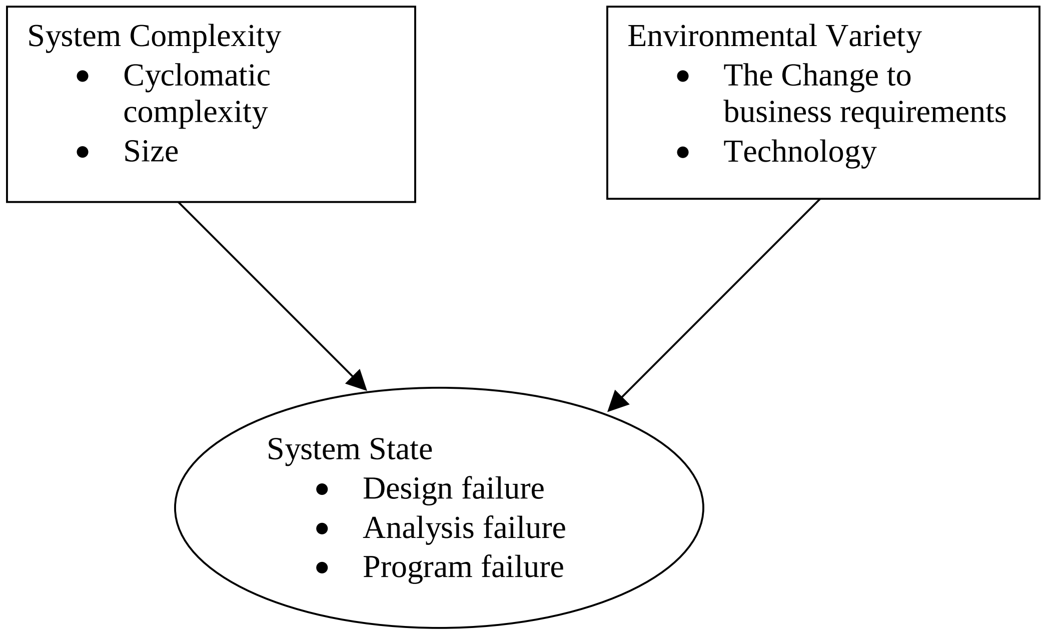

Figure 3.2 Research Framework.

The conceptual framework has three components: organizational complexity, system variety and system state. System complexity and system variety impact the system states. Environmental variety can be measured by business requirements changes and technology impact. System complexity can implicate software size and software complexity in research framework. System state is measured by the state of failure which is further classified into analysis, design and programming failure.

As suggested in the literature, program size affects the complexity of a system (Kimberly & Evanisko, 1981; Stanwick, 1998). System complexity causes organizational complexity (Damanpour, 1996; Keidel et al 1994). In the present study, system complexity and program size will be used to measure the conceptual model construct - system complexity.

There is ample evidence in the literature suggesting that environmental variety plays a role in the system performance (Lucas et al., 1999; Marsden et al. 1999; Meyer & Goes, 1988). Business requirements are directly linked to information systems development and maintenance (McLean et al. 1993; Hay and Munoz, 1997). Business requirements specify the functions of the desired the system (Gilbert, 1983). According to Boehm (1979), a precise specification greatly aids effective management control of a project. The frameworks of Block (1983) and Davis et al. (1992) identified technology as an important component affecting the overall performance of the system. In this study technology and business requirements are employed to measure the conceptual model construct environmental variety.

Failure is defined as the state when the change of code is needed to meet the system requirements. Three factors cause the program to change: (a) repair change, which happens more often after the program is newly developed when certain tasks can not be completed; (b) business requirements; (c) technology. The change of code may be traced to a stage in the IS development life cycle, such as analysis, design, and development.

Systems or programs are a series of instructions or statements that when decoded by a computer or by a computer and a translation program cause the computer to do work (Fox, 1982). Software is a set of programs that interact with each other.A system consists of many components. A computer-based system consists of hardware, software, people, procedure, database, and telecommunications (McLean et al. 1992). Requirements definition is a system level problem that will affect the software level. Any changes in the component or the subsystem affect the entire system.

Independent Variables

Four variables are employed in this study: program size, technology, cyclomatic complexity, and the changes in business requirements. Two variables, program size and cyclomatic complexity are used to measure system complexity. Literature has a number of ways to measure these three variables. In this study, system complexity is measured by using Cyclomatic complexity metrics (McCabe, 1976). Program size is measured by the total number of executable lines. The change to business requirements is a continuous variable. It is measured by summing those requests that are related to changes in business requirements in a program. Technology is classified into two groups: hardware and software. Hardware is operationalized by adding the modifications that are related to hardware changes. Software is measured by totaling the modifications that are related to software changes.

Complexity is an abstract property of software, Fenton (1997) says that complexity can be viewed from a number of different perspectives including program algorithms, structural and cognitive complexity contexts. It cannot be directly measured. Complexity can produce entropy for a program. Complex programs are hard to modify and maintain. In this study we will explore how software complexity correlates to failures.

The structured programming primitives used in the calculation of cyclomatic complexity are based on the control flow graph rather than the source code text. This allows cyclomatic complexity metric to measure structural quality, independently of the syntax of any particular programming language. The process of calculating cyclomatic complexity is similar to the calculation of module design complexity. When calculating cyclomatic complexity, primitive constructs can be removed whether or not they involve module calls.

Dependent Variables

In this study, the frequency of change of program is a dependent variable. Four variables, design failure, analysis failure, program failure and total failure are employed as a measure of system state. The total failure is the summation of design, analysis and program failure. The maintenance log was documented by the professional EDP auditor who has advanced knowledge on programming and maintenance. The maintenance log contains 1702 modifications on 100 COBOL programs over the period 1978-1992. The log records the program installation date, the date when the request for modification was proposed, the date the programmer was assigned to modify the program, the date when the modification was completed and the reasons that caused the modifications. The reasons for modifications are grouped into three categories: analysis failure, design failure and programming failure. Each of these categories is defined by the set of modifications which arise from each respective step of the software development life cycle. Analysis failure is measured by the frequency of modifications that are due to the analysis stage of the software development life cycle. Design failure is measured by the frequency of modifications that are due to the design stage of the software development life cycle. Programming failure is measured by the frequency of modifications that are due to the programming stage of the software development life cycle.

Table 1. Constructs and Variables Table

Dependent Construct	Previous Research	Measurement
Total failure		The total frequencies of changes to the program due to any reason
Design failure		The frequency of the changes to the program due to the design error.
Analysis failure		The frequency of the changes to the program due to the analysis error.
Program failure		The frequency of the changes to the program due to the programming error.
Independent Construct		
Cyclomatic complexity	McCabe1976	Cyclomatic metrics.
Total Procedure lines	Banker& Slaughtere (2000)	The total number of executable lines of code
Software size	Jones (1983); Sumner (1999); Mills (1983) Dijkstra (1972)	Total number of executable lines.
Business requirements	Block (1983); Davis et al (1992); Sumner (1999)	The frequency of the change to the program due to changes in business requirements change.

Research questions:

Each stage of traditional software development life cycle requires different activities and has a varying role in the quality of the final system. In this study, the failures are traced to design, analysis and development. The frequency of design error analysis error and development error are used to measure the dependent variable IS failure. This method of analysis allows the identification of which stage of software development life cycle is most prone to errors. By identifying this, practical suggestions on software development can be made. Certain stages of software life cycle require more efforts than others.

This study will examine the following research questions:

1. Is there a relationship between changes to business requirement and software failure?
2. Is there a relationship between software complexity and software failure?
3. Is there a relationship between program size and software failure?
4. Is there a relationship between the age of software and software attributes?

Chapter 4

Research Methodology

This chapter describes the research design, data collection, sample population, hypotheses tested, and data analysis methods used to validate the conclusions of this study. Logistic regression analysis was conducted to determine the significance of the variables: program size, business requirements, and cyclomatic complexity in predicting analysis, design, and programming failures. In addition, the relationship between the age (life of the software) and the following variables was investigated: design failure, analysis failure, program failure, software failure, hardware failure, and cumulative failure. Regression analysis was used with a log transformation of these variables.

Research Design

Archival data analysis was performed in this study by analyzing data collected by professional programmers and auditors in a large insurance company between 1978-1992. Both qualitative and quantitative data were documented for the 100 COBOL programs. These programs include commercial billing, statistical reports, programs to print requested information, programs to give summary reports, commercial bill cancellation programs, commercial bill zero balance programs, deposit premium programs, programs capable of listing commercial policies, backup statistical system transaction programs, and programs for the annual warehouse purge of cancelled/expired policies. The smallest program had 101 lines of codes and the largest one had 18488 lines of codes.

Characteristics of the 100 COBOL programs

Archival corporate data provided the following information related to the 100 COBOL programs: program size, changes due to new business requirements, cyclomatic complexity, and total procedure lines. The number of lines of codes was used as a measure of the size of the program. The literature suggests that the longer the program, the more effort needed to maintain the program. Also, the literature suggests that complexity measures are positively related to both the cost of developing a program and the time spent on developing a program. As theses program characteristics are focused on the design and internal complexity of the programs, it is not surprising that these variables are moderately correlated.

Programs Sampled

In this study, 100 COBOL programs are used to test the proposed hypotheses. These programs were from different departments of a single company. From program logs, dates and reasons for changes in the programs were obtained. Software complexity measures were obtained from programs designed to calculate complexity metrics. Cyclomatic complexity, procedure codes and the number of lines of code were reported from programs designed to calculate these metrics. Information related to technology and business requirement changes were obtained from the documentation in the program log. The program log also reported the date when a program was released, the date when a program was assigned to a programmer to modify the program, the date when errors were corrected, and the time allotted to make changes in a program. The number of programs with analysis failures, design failures, and failures due to a change in business requirements were 29, 55, and 58, respectively.

Unit of Analysis

Following the principles of data collection established by Eisenhardt (1989) and Yin (1989), data were obtained from the program logs of a large corporation. The unit of analysis for this study is a COBOL program used in the corporation. Each of the 100 programs performed a different function. Maintaining commercial billing systems and ordering systems to process invoices are examples of the functions performed by these programs. The date of each software failure (modification) and its cause were accurately recorded from the company's program logs.

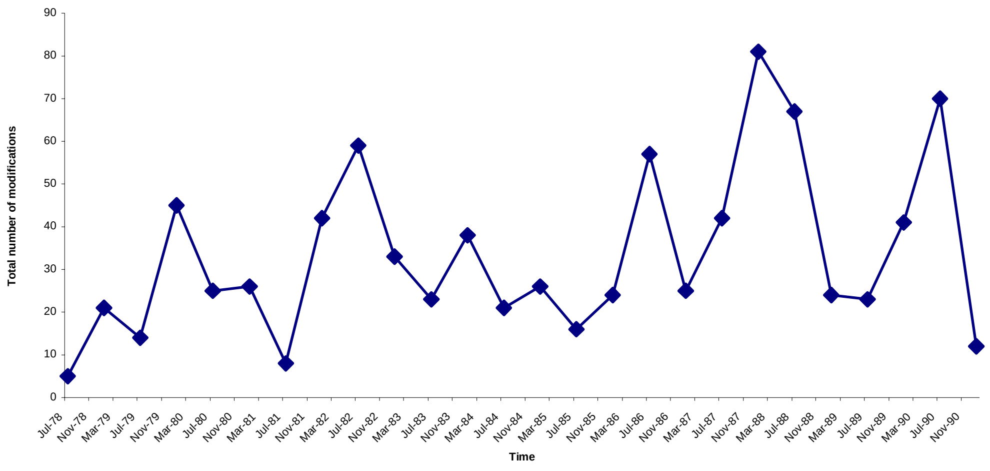

Figure 4.1. Total number of modifications due to analysis error from 1978 through 1992.

Figure 4.1 shows that the number of software modifications fluctuated over the 10- year period used in this study. The following periods experienced an increase in the numbers of modifications made to the software in use by the company: July 1982 – December 1982, and July 1986 – December 1986, January 88- June 1988, July 1988 – December 1988, July 1990 – December 1990. The number of modifications ranges from 5 to 81 over this ten-year period. Further investigation of this software may reveal cyclical causes for the increased modifications during these periods.

Analysis Failure

Analysis failure is defined as a necessary modification to a program code to correct a problem in the software that is related to the analysis stage of software development. The 100 COBOL programs were dichotomized into two groups: those with modifications due to analysis and those having no modifications related to the analysis stage of software development. There were 75 programs in the non-analysis failure group and 25 programs in the analysis failure group. The mean number of business requirement changes in the analysis failure group is 6.92 whereas the mean number of business requirement changes in the non-analysis failure group is 3.11.

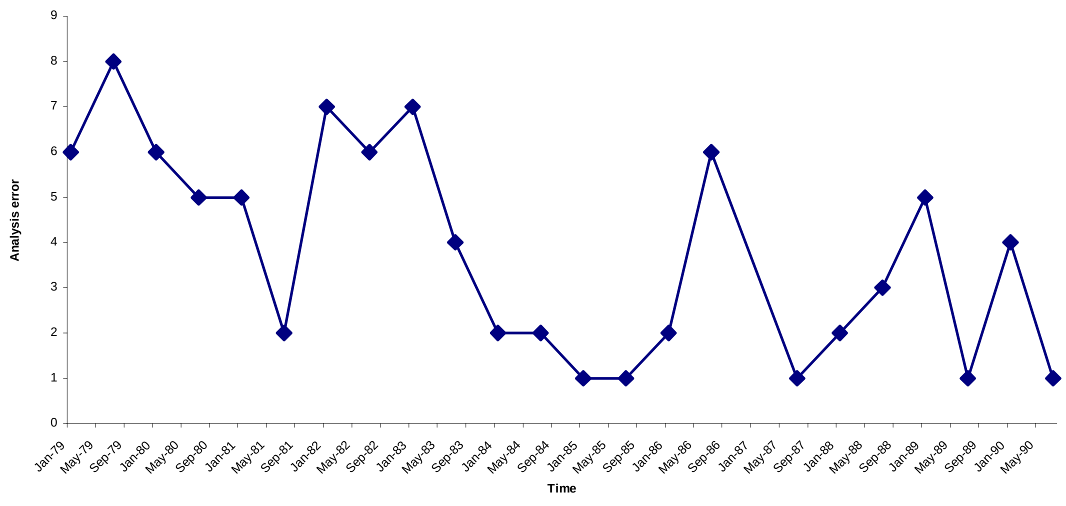

Figure 4.2. Number of modifications due to analysis error from 1978 through 1992

The accumulated number of modifications due to analysis error is 87 for the period 1978 through 1992 for the 100 COBOL programs. The number of errors due to the analysis stage appears to be slightly larger between the years 1979 to 1983 than for the years 1984 to 1990. Over time, perhaps the analysts improved their ability to analyze business situations and problems. Figure 4.2 also shows an improvement over time with respect to software modifications due to analysis error. During the period July 1979 through December 1979, the number of modifications due to analysis error reached a peak at eight. The least value for the number of modifications is one which occurred during certain periods of 1985, 1987, 1989, and 1990. The range of modifications is from one to eight.

Design failure

The software development life cycle consists of analysis, design, development and maintenance. Failures can occur at different stages of the software development life cycle. The design phase of software development is important because the cost of future maintenance can be easily impacted by poor design of software. Design failure in software is defined as code needing modification because of errors related to the design phase of the software development life cycle. The 100 programs were classified into two categories, one is with design failures and the other is without design failures.

In this study, 47 programs fall into the category of non-design failure and the other 53 programs belong to the group of design failure. Thus, 53% of the programs have design failure problems. The frequency of design error in the group with design failure ranged from 1 to 33. The mean for business requirements change is 0.38 for non-design failure programs whereas the mean for business requirements change is 7.32 for programs with design failure. This suggests that changes in business requirements may have an effect on design failure.

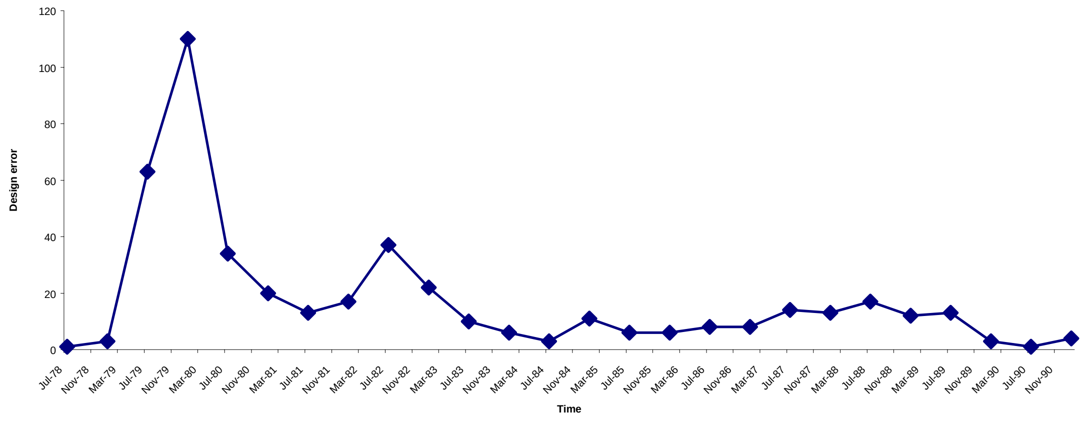

Figure 4.3. Number of modifications due to design error from 1978 through 1992

The total number of modifications due to design error is 449 for 100 COBOL programs from 1978 through 1992. There were two time periods in which unusually large number of modifications were made due to design error. One occurred in the first six months of 1980 during which 110 modifications were made, the other occurred during the period of the last six months of 1982 during which 37 modifications were made. During other time intervals, the number of modifications due to design error was less than 20 without any sharp spikes. The range of modifications is from one to 110.

Programming Failure

The development stage of software development involves the coding activities of the programmers. Logical errors incurred in this stage will cause the system to malfunction. Extra attention should be given to the logical flow of the program to ensure that the final system will have minimal failures traced back to this stage. Software programs having failures due to programming are classified into two groups. One group contains programs with programming failures and the other group includes only programs without programming failures.

In this study, 51 programs were in the non-programming failure group and 49 programs were in the programming failure group. Thus, we can say 49% of the programs have programming failures. The mean number of changes due to the number of changes in business requirements is 0.60 for the non-programming failure group whereas the mean for the number of changes in business requirements is 7.65 for the programming failure group. This implies that business requirements change may have an important effect on programming failure.

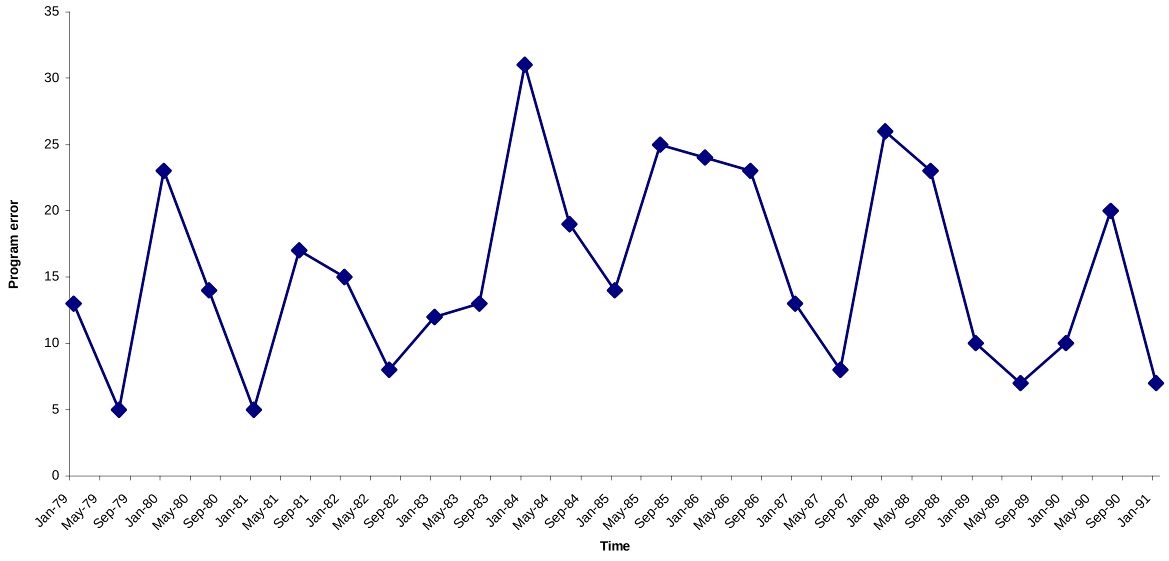

Figure 4.4. Number of modifications due to programming error from 1978 through 1992

The total number of modifications due to programming errors is 385 from 1978 through 1992 for the 100 COBOL programs. The range of modification is from five to 31. The lowest number of modifications occurred from July 1979 through December 1979 and January 1981 through December 1981. During January 1984 though July 1984, the heaviest modifications due to programming error occurred. Figure 4.4 illustrates that the number of modifications due to programming errors is erratic.

Software Complexity

Metrics that measure software complexity provide an indication of the ease in understanding and modifying a program. Many different kinds of software complexity metrics exist. Big companies can develop their own complexity metrics. The metrics known as essential complexity and cyclomatic complexity were developed by Halstead (1976) to measure the difficulty in maintaining programs. Because these two metrics are highly correlated, only cyclomatic complexity is used in this study. Special programs designed to calculate complexity were used to obtain this metric.

Cyclomatic Complexity	10-299	300-599	600-1382
The number of programs	88	75	5
Mean	102	423	989
Standard deviation	74	80	336

Most of the programs have cyclomatic complexity values less than 600. Only 5% of the programs have a cyclomatic complexity value of 600 or more. The standard deviation of those programs with a complexity score of 600 or more is approximately four times that for programs with a complexity score less than 300 or between 300 and 599. This variable will be used as a predictor of program failure.

Changes to Business Requirements

An important aspect of system development is converting user "needs" into clear, concise, and verifiable system requirements. Businesses function in a dynamic environment. New products and services are coming into the market every day. It is quite common that old products and services are exiting the market due to competition and changing customer taste. Nothing can last forever. Change is one phenomenon that is perpetual. Many reasons can cause a change in business requirements such as the change of organizational structure, the change of management, downsizing, merging, the turnover of the employees, expanding business activities and the pressure from the competitors. The number of changes in business requirements is believed to be significantly correlated with the number of modifications in software programs. Programs that are part of a company's information system are usually integrated into the organization itself and developed to improve the performance of a company. Every information system is based on clearly defined business requirements. When business requirements change, the current information systems usually cannot meet the needs of business activities and cannot fully support business operations. Information systems must change to cope with the changes in business requirements to better support organizational activities and operations.

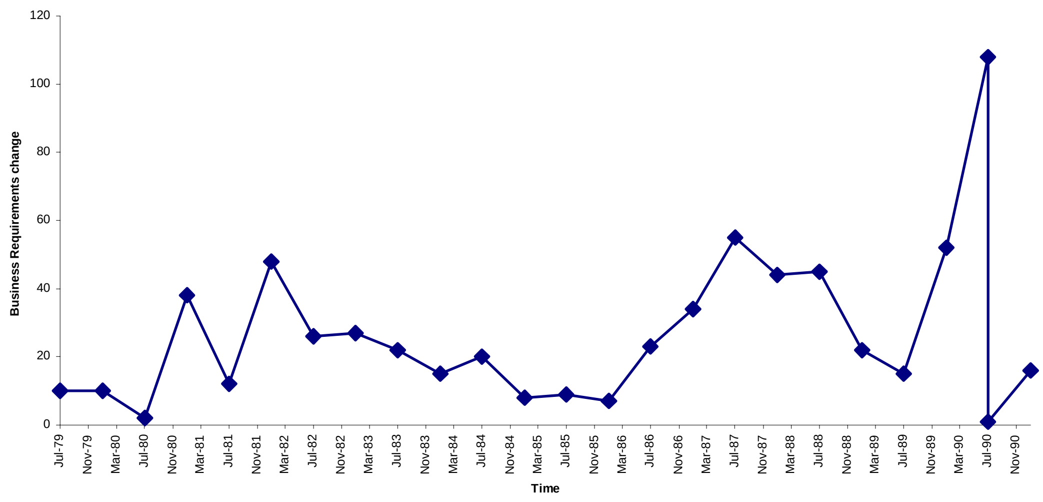

Figure 4.5. Changes in business requirements from 1978 through 1992

The accumulated modifications due to business requirements over the period 1978-1992 for the 100 COBOL programs occurred 643 times. Among the 100 COBOL programs, 58 programs had modifications due to business requirements change. As figure 4.5 illustrates, the maximum occurrence is 58 and lowest occurrence is 1. The frequency of changes in business requirements is displayed below. Most programs had 10 or fewer modifications due to changes in business requirements. Note that 42% of the programs had no business requirements change while 5% of the programs have 20-58 modifications due to business requirements change. When business requirements change, there is a strong likelihood that programs will fail. This relationship will be investigated in the hypotheses tested in this study.

Frequency of business requirements change	0	1-10	11-20	21-58
Number of programs	42	44	9	5

Technology

Program logs are used to document the dates for modifications in the software due to changes in technology. Changes in technology include hardware replacement, program database changes, interface changes, DBMS errors, and revisions of COBOL2. The number of software modifications due to technology changes may be related to the age or life of software programs. This relationship will be investigated in this study. The total occurrences of modifications in software programs due to changes in technology over a 10-year period on the 100 COBOL programs used in this study are 63. Only 41 programs experienced technology changes while the other 59 programs did not. The maximum number of technology changes for any single program is 3. Three programs had a technology change frequency of 3. Overall, 16 programs had a technology change frequency of 2 and 21 programs had a technology change frequency of 1.

In this study, changes due to technology are typically due to changes in hardware, software and using the newest, innovative devices or procedures to complete business tasks. Businesses implement new technologies to improve efficiency and gain competitive advantages in the market. Companies have realized that in the information age changes in technology are an inevitable part of business practices and operations. There are always costs involved in acquiring new technologies, such as the cost to buy the technologies, the operation and training cost for using the new technology.

Frequency of technology change	0	1	2	3
The number of programs	59	22	16	3

Overall, 59% of the programs have no modifications due to changes in technology. Only one modification due to changes in technology was experienced by 22% of the programs. Two and three modifications due to changes in technology were experienced by 16% and 3% of the programs, respectively.

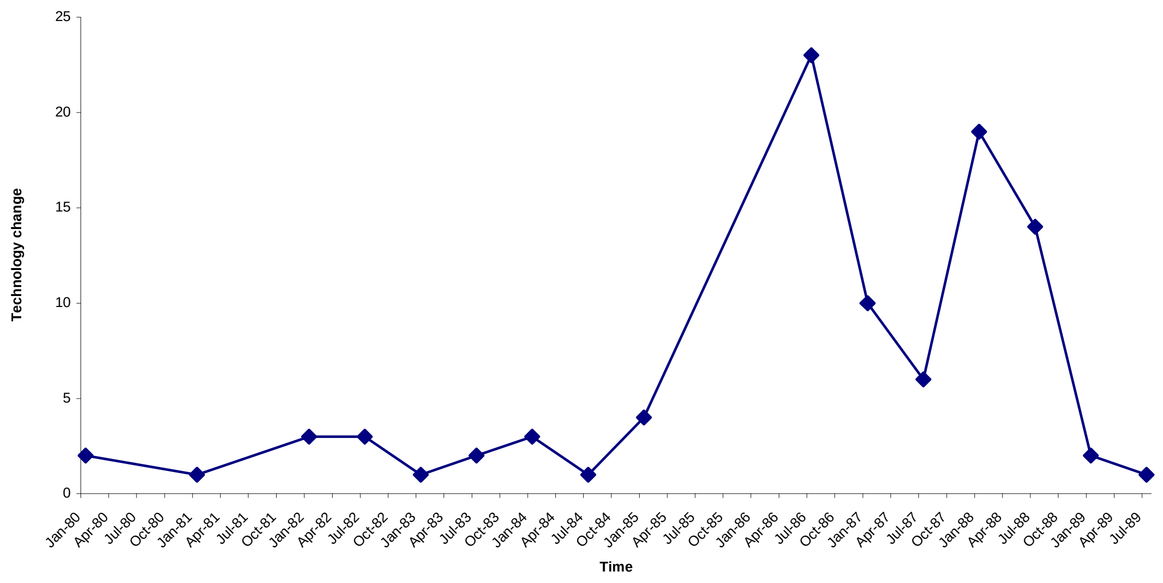

Figure 4.6. Number of modifications due to changes in technology from 1978 through 1992

The total number of modifications due to changes in technology was 95 from 1978 through 1992. The range of modifications due to changes in technology is from one to 23. Figure 4.6 illustrates that the period from January 1985 through January 1989 had an unusually large number of modifications from changes in technology. Before January 1985, few modifications were made because of changes in technology.

Design failure hypotheses

In this study the statistical analysis to test the relationship between modifications due to design failure and the variables program size, business requirements, cyclomatic complexity, and total procedure lines are presented in tables 4.1 through 4.11. Theses hypotheses are as follows:

H_{1a}: There is a relationship between design failure and business requirements controlling for cyclomatic complexity and program size.

H_{2a}: There is a relationship between design failure and program size controlling for business requirements and cyclomatic complexity.

H_{3a}: There is a relationship between design failure and cyclomatic complexity controlling for business requirements and program size.

In analyzing the logistic model used to test the hypotheses related to each of causes of failure in software, an overall test is first presented to show the significance of the model itself. For the logistic model used to investigate the relationship between modifications due to design failure and the predictor variables, the Wald's statistic is used. However, alternative global tests are presented as well. As shown in table 4.1, Wald's test has a p-value of less than 0.0003. The overall model with the intercept and 4 independent variables is significant.

Table 4.1. Overall model for modifications due to design failure

Criterion	Intercept Only	Intercept and Covariates
AIC	140.269	83.126
SC	142.874	93.546
-2 Log L	138.269	75.126

Testing Global Null Hypothesis: BETA=0

Test	Chi-Square	DF	Pr > ChiSq
Likelihood Ratio	63.1939	3	<.0001
Score	26.3625	3	<.0001
Wald	20.9351	3	<.0003

Table 4.1 also displays the AIC, SC, and –2LogL statistics to make comparisons between the full model (intercepts and covariates) and the null model (only has an intercept term). From these values, the overall adequacy of the model can be assessed further. The –2LogL statistic is commonly used in understanding the contribution by the

predictor variables. The value –2Log L for the model with only the intercept has a value of 138.269 09 whereas the model with intercept and covariates has a value of 75.075. By including the covariates, the independent variable in the model, there is a large decrease in –2Log L. This suggests that the model with intercept and independent variables is a good fit to the data.

Table 4.2. Parameters for logistic model for modifications due to design failure

Parameter	DF	Estimate	Standard Error	Chi-Square	Pr > ChiSq
Intercept	1	1.1620	0.4200	7.6545	0.0057
Business requirements	1	-1.0053	0.2208	20.7239	<.0001
Cyclomatic complexity	1	-0.0145	0.00914	2.5189	0.1125
Program size	1	0.00328	0.00194	2.8690	0.0903

Table 4.2 presents the significant levels of the three independents variables, business requirements, cyclomatic complexity and program size. At the 1% significance level, the variable business requirements is the only variable significant when controlling for the other variables in the model. However, the classification rate was 82%. Thus the model appears to have good predictive validity. Table 4.2 shows support for hypotheses H_{1a} and H_{2a} at the 1% and 10% significance levels, respectively.

Table 4.3. Testing model with modifications due to design failure when influential data are removed

Criterion	Intercept Only	Intercept and Covariates
AIC	135.634	74.363
SC	138.209	87.237
-2 Log L	133.634	64.363

Testing Global Null Hypothesis: BETA=0

Test	Chi-Square	DF	Pr > ChiSq
Likelihood Ratio	69.2710	3	< .0001
Score	26.0507	3	< .0001
Wald	20.5110	3	0.0004

During the analysis of the logistic model, it was found that certain observations were influential. So further analysis was performed to assess the effect of the influential observations. Without three influential observations, -2 Log L is smaller than it was with all the observations. The model with intercept only has a much larger value of –2 Log L by including the three covariates in the model. This implies that the model is a good fit to the data. Note that Wald's test is significant at the 1% significance level (p=0.0004). The results of Wald's test including the influential observations agree with the results of Wald's test deleting the influential observations. Wald's test confirms the overall fit of the model.

Table 4.4. Logistic regression parameters with influential data removed

Parameter	DF	Standard Estimate	Standard Error	Chi-Square	Pr > ChiSq
Intercept	1	1.4915	0.4460	11.1833	0.0008
Business requirements	1	-1.2733	0.2810	20.5362	< .0001
Cyclomatic complexity	1	-0.0186	0.0112	2.7827	0.0953
Program size	1	0.00363	0.00213	2.8995	0.0886

After removing three influential observations, the logistical regression analysis was redone. Table 4.3 and table 4.4 display the logistical regression results. After removing three influential observations, variables business requirements, cyclomatic complexity and program size are significant at the 10% significance level in predicting the dependent variable design failure. Therefore, with the removal of influential observation, the hypotheses H_{1a} and H_{2a} and H_{3a} are now supported.

Analysis Failure Hypotheses

The following hypotheses are presented to test the relationship between analysis failure and each of the independent variables.

H_{4a}: There is a relationship between analysis failure and business requirements controlling for program size and cyclomatic complexity.

H_{5a}: There is a relationship between analysis failure and program size controlling for business requirements and cyclomatic complexity.

H_{6a}: There is a relationship between analysis failure and cyclomatic complexity controlling for business requirements and program size.

Table 4.5. Testing overall model with dependent variable analysis failure

Criterion	Intercept Only	Intercept and Covariates
AIC	114.467	102.800
SC	117.072	113.221
-2 Log L	112.467	94.800

Testing Global Null Hypothesis: BETA=0

Test	Chi-Square	DF	Pr > ChiSq
Likelihood Ratio	17.6668	3	0.0005
Score	15.3092	3	0.0016
Wald	10.1649	3	0.0172

Table 4.5 presents the results for testing the overall model. The value -2 Log L is much smaller for the model with the intercept and covariates than for the model with only the intercept and therefore indicates that the logistic model with the independent variables

is an improvement over the null model. Wald's test is significant with the probability of 0.0001. This test supports that the independent variables contribute significantly to predicating the likelihood of modifications due to analysis failure.

Table 4.6. Parameter estimate for model with dependent variable analysis failure

Parameter	DF	Estimate	Standard Error	Chi-Square	Pr > ChiSq
Intercept	1	1.1314	0.3676	.4743	0.0021
Business requirements	1	-0.1793	0.0611	8.6156	0.0033
Cyclomatic complexity	1	-0.0106	0.00544	3.7817	0.0518
Program size	1	0.00298	0.00115	6.6933	0.0097

Table 4.6 shows that the chi-square tests on the individual independent variables are all significant. The three independent variables business requirements, cyclomatic complexity, and program size are good predicators for the dependent variable analysis failure. The logistical regression model also has a reasonable classification rate. Eighty percent of the observations are correctly classified. The analysis supports H_{4a}, H_{5a}, and H_{6a}. We can conclude analysis failure is related to business requirements change controlling for the effect of other variables in the model. At the 1% significance level, there is a relationship between analysis failure and program size controlling for the effect of other variables in the model. At the 10% significance level, there is a relationship between analysis failure and cyclomatic complexity controlling the effect of other variables in the model.

Programming failure hypotheses

In order to test the relationship between programming failures and the independent variables, four hypotheses are stated as follows:

H_{7a}: There is a relationship between programming failures and business requirements controlling for program size and cyclomatic complexity.

H_{8a}: There is a relationship between programming failures and program size controlling for business requirements and cyclomatic complexity.

H_{9a}: There is a relationship between programming failures and cyclomatic complexity controlling for business requirements and program size.

Table 4.7. Testing overall model with dependent variable programming failure

Criterion	Intercept Only	Intercept and Covariates
AIC	140.589	97.849
SC	143.195	106.281
-2 Log L	138.589	87.849

Testing Global Null Hypothesis: BETA=0

Test	Chi-Square	DF	Pr > ChiSq
Likelihood Ratio	50.7400	3	< .0001
Score	24.8897	3	< .0001
Wald	19.7596	3	0.0002

Table 4.7 presents the statistics for testing the overall model. Wald's statistics provides evidence that the overall fit of the model with intercept and the three independent variables is adequate. The data only support H_{7a} and we conclude that there is relationship between programming failure and business requirements controlling for program size and cyclomatic complexity being in the model.

Table 4.8 Parameter estimates for model with dependent variable programming failure

Parameter	DF	Estimate	Standard Error	Chi-Square	Pr > ChiSq
Intercept	1	1.1854	0.3998	8.7899	0.0030
Business requirements	1	-0.6869	0.1551	19.6141	<.0001
Cyclomatic complexity	1	-0.00256	0.00660	0.1503	0.6983
Program size	1	0.000825	0.00134	0.3801	0.5376

Chi-square tests on the individual independent variables show that only the variable business requirements is significant. The other independent variables cyclomatic complexity and program size are not significant predicators in predicting the dependent variable programming failure. At the 1% significance level, we can state that there is a relationship between business requirement change and programming failure controlling for the effects of program size, and cyclomatic complexity. The classification rate is 75% which is not very high in correctly classifying all the observations.

Table 4.9 Testing overall model with dependent variable programming failure after removing influential observations

Criterion	Intercept Only	Intercept and Covariates
AIC	136.460	92.147
SC	139.035	105.021
-2 Log L	134.460	82.147

Testing Global Null Hypothesis: BETA=0

Test	Chi-Square	DF	Pr > ChiSq
Likelihood Ratio	52.3127	3	< .0001
Score	24.1259	3	< .0001
Wald	19.0887	3	0.0008

The value -2 Log L after deleting three influential observation is smaller when compared to –2 Log L including all observations. There is also a big reduction in –2 Log L in the model with intercept and covariates. Wald's statistic is 0.0008 after deleting three influential observations. Therefore, the overall model fits the data.

Table 4.10 Parameter estimates for model with dependent variable programming failure after removing influential observations

Parameter	DF	Estimate	Standard Error	Chi-Square	Pr > ChiSq
Intercept	1	1.2793	0.4175	9.3875	0.0022
Business requirements	1	-0.7372	0.1732	18.1184	<.0001
Cyclomatic complexity	1	0.00161	0.00778	0.0427	0.8362
Program size	1	0.000484	0.00187	0.0674	0.7951

Chi-square tests on individual independent variables show that only the variable business requirements is a significant predicator in predicting dependent variable programming failure. After deleting three influential observations, the same conclusion as before. We can only state that business requirement change is related to programming failure when controlling for the effects of other three independent variables.

Hypotheses Testing Total Failure

A new variable, total failure is formed to test another three hypotheses. Total failure is the summation of design failure, analysis failure and programming failure.

H_{10a}: There is a relationship between total failure and business requirements controlling for program size and cyclomatic complexity.

H_{11a}: There is a relationship between total failure and program size controlling for business requirements and cyclomatic complexity.

H_{12a}: There is a relationship between total failure and cyclomatic complexity controlling for business requirements and program size.

Using dependent variable total failure to model the relationship, the following results are produced:

Table 4.11 Analysis using total number of failures

Criterion	Intercept Only	Intercept and Covariates
AIC	136.602	65.768
SC	139.208	76.189
-2 Log L	134.602	57.768

Testing Global Null Hypothesis: BETA=0

Test	Chi-Square	DF	Pr > ChiSq
Likelihood Ratio	76.8343	3	<.0001
Score	24.8660	3	<.0001
Wald	15.4348	3	0.0015

Analysis of Maximum Likelihood Estimates

Parameter	DF	Estimate	Standard Error	Chi-Square	Pr > ChiSq
Intercept	1	0.8253	0.4412	3.4993	0.0614
Business requirements	1	-1.8993	0.4921	14.8980	0.0001
Cyclomatic complexity	1	-0.0166	0.0120	1.8967	0.1685
Total lines in procedure	1	0.00412	0.00259	2.5228	0.1122

-2Log L is significantly reduced with the independent variables in the model. Wald’s statistics show the significance of the overall model fit. Coefficients analysis suggests that only the variable business requirements is the good predictor whereas with respect to the other two independent variables, program size and cyclomatic complexity are not significant predicators in predicting the total number of modifications.

Hypotheses Testing Using Independent Variable Age to Predict Failure

The following tests are performed to analyze the relationship between age and several dependent variables. These dependent variables include design failure, analysis failure, program failure, software failure, hardware failure, total failure, and cumulative failure. A graph of the dependent variables with age indicated some curvature in the model. After experimenting with several transformations, the log transformation was used, as it appeared to be a good fit for several of the models. Software that had been in use for 8 to 18 years was included in this portion of the study. The total number of software programs was 35 with this restriction. In this section, hypotheses are listed to test the relationship between the variable program age and each of 7 variables using regression analysis with a log-transformed dependent variable.

H_{1b}: There is a relationship between program age and design failure.

Dependent Variable: log of number of modifications due to design error

Table 4.12. Design errors occurring over time

Variable	DF	Parameter Estimate	Standard Error	t Value	Pr > |t|
Intercept	1	0.34673	0.82337	0.42	0.6764
age	1	0.00012777	0.00014932	0.86	0.3983

Since the p-value is 0.3983, the conclusion is that there is no relationship between the log of design error and the age of the system. Thus the first hypothesis in this section is not supported.

H_{2b}: There is a relationship between program age and analysis failure.

Dependent Variable: log of number of modifications due to analysis error

Table 4.13. Analysis errors occurring over time

Variable	DF	Parameter Estimate	Standard Error	t Value	Pr > |t|
Intercept	1	-0.75156	0.56988	-1.32	0.1963
age	1	0.00023330	0.00010335	2.26	0.0307

The p-value from the regression analysis is 0.0307. Thus, at the 5% significance level, we conclude there is a relationship between the log of analysis error and the age of the system.

H_{3b}: There is a relationship between program age and programming failure.

Dependent Variable: log of number of modifications due to programming error

Table 4.14. Programming errors occurring over time

Variable	DF	Parameter Estimate	Standard Error	t Value	Pr > |t|
Intercept	1	-0.52523	0.89100	-0.59	0.5595
age	1	0.00028072	0.00016158	1.74	0.0917

At the 10% significance level, we conclude that there is a relationship between the log of the program errors and the age of the systems.

H_{4b}: There is a relationship between program age and software failure.

Dependent Variable: log of number of modifications due to software failure

Table 4.15. Software failures occurring time.

Variable	DF	Parameter Estimate	Standard Error	t Value	Pr > |t|
Intercept	1	0.11106	0.47776	0.23	0.8176
age	1	0.00007616	0.00008664	0.88	0.3857

Since the p-value is equal to .3857, there is no relationship between the log of number of modifications due to software failure and the age of the systems.

H_{5b}: There is a relationship between program age and hardware failure.

Dependent Variable: log of number of modifications due hardware failure

Table 16. Hardware failures occurring over time.

Variable	DF	Parameter Estimate	Standard Error	t Value	Pr > |t|
Intercept	1	0.45087	0.18783	2.40	0.0222
age	1	-0.00007202	0.00003406	-2.11	0.0421

Since the p-value less than .05, we conclude that there is a relationship between the log of modifications due to hardware failure change and the age of the systems at the 5% significance level.

H_{6b}: There is a relationship between program age and changes in business requirements.

Dependent Variable: log of number of modifications due to changes in business requirements

Table 4.17. Changes to business requirements occurring over time.

Variable	DF	Parameter Estimate	Standard Error	t Value	Pr > |t|
Intercept	1	-0.48545	0.80761	-0.60	0.5519
age	1	0.00030778	0.00014646	2.10	0.0433

At the 5% significance level, the hypothesis that there is a relationship between the log of changes in business requirements and age of software is supported. To gain further insight into the number of software errors or modifications over time, the total number of errors was analyzed. This new variable was the sum of each of errors produced by the causes previously analyzed individually.

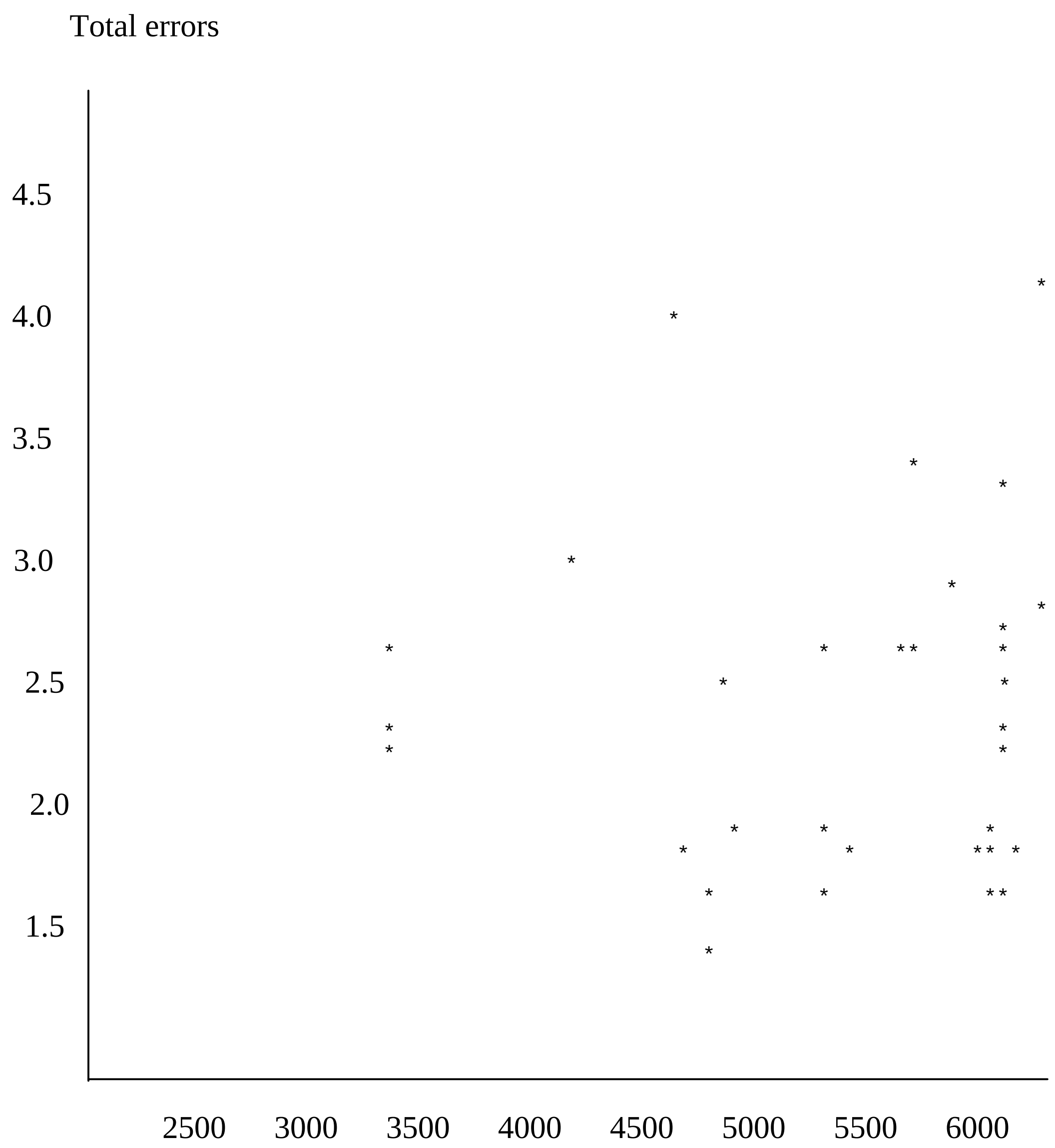

Figure 4.7. The number of modifications over years

To understand if the relationship between the log of the total number of errors and the age of software programs is significant, the following regression analysis was performed. Note that this analysis produced a significant relationship at the 1% significance level. However, the R-square was still small with a value of 20.79%.

H_{7b}: There is a relationship between program age and total error.

Table 4.18. Total failures occurring over time.

Parameter Variable	DF	Estimate	Standard Error	t Value	Pr > |t|
Intercept	1	0.33313	0.67150	0.50	0.6231
age	1	0.00035843	0.00012178	2.94	0.0059

To gain a more realistic view of the number of errors produced over time and the age of software, a new variable was used which cumulatively summed the errors. A graph of this variable and the age of the software is presented below. The graph of this new variable with age has a very strong relationship with the square of the independent variable age. However, the log transformation of this new variable, which is the cumulative sum of the errors, also has a strong linear relationship with age. With this log transformation, the R-square value was a high 94.08%. The regression analysis reveals that age is a significant predictor at the 1% significance level.

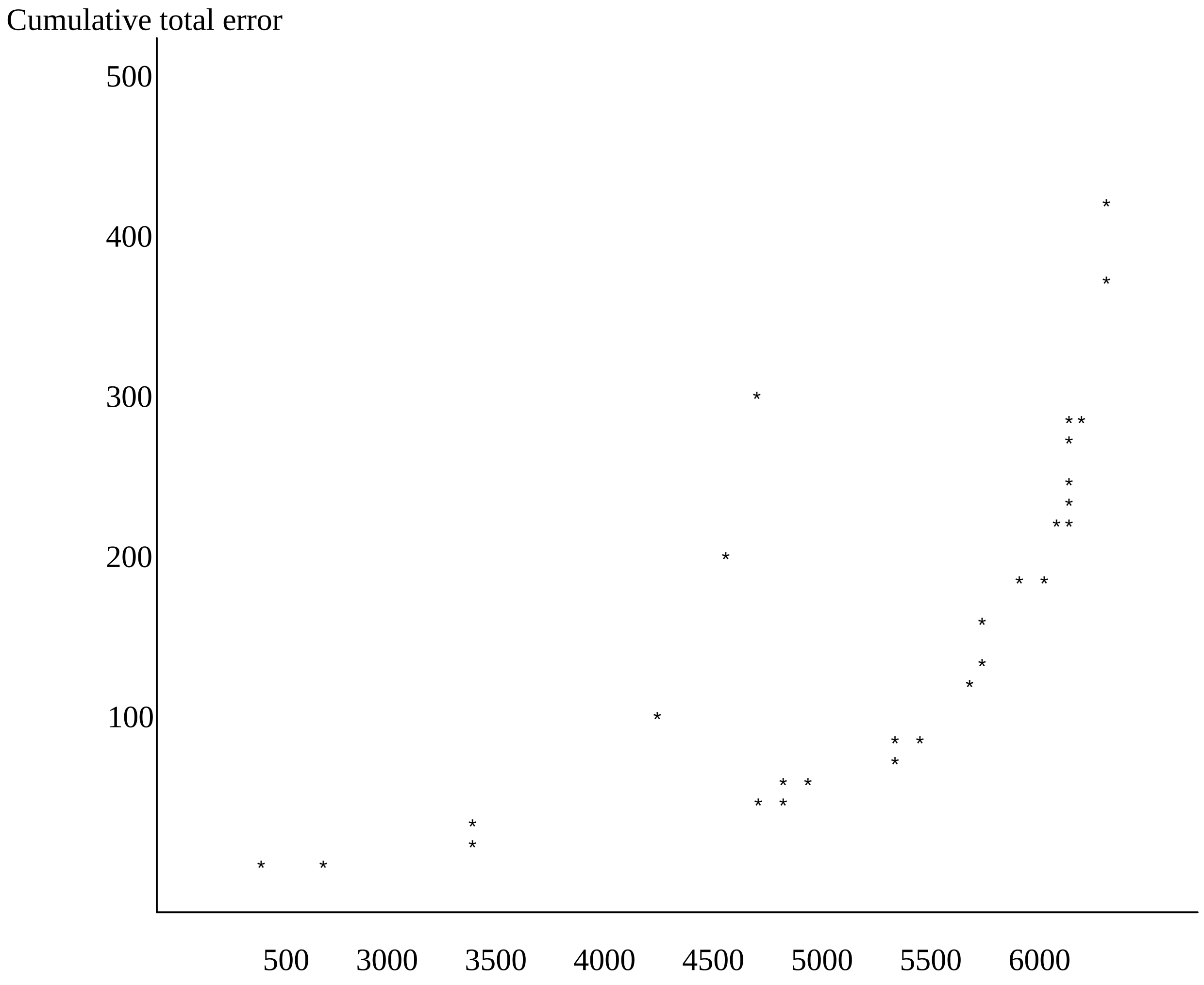

Figure 4.8. The cumulative number of modifications over years.

Dependent Variable: log of the cumulative total number of modifications

Table 4.19. The log transformation on total failures.

Variable	DF	Parameter Estimate	Standard Error	t Value	Pr > |t|
Intercept	1	-2.49208	0.31477	-7.92	<.0001
age	1	0.00131	0.00005708	22.90	<.0001

Table 4.20. Summary of hypothesis tests

Hypotheses	Result
H_{1a}: There is a relationship between design failure and business requirements controlling for cyclomatic complexity and program size.	Supported at the 1% significance level
H_{2a} : There is a relationship between design failure and program size controlling for business requirements and cyclomatic complexity.	Supported at the 10% significance level
H_{3a}: There is a relationship between design failure and cyclomatic complexity controlling for business requirements and program size.	Supported at the 10% significance level
H_{4a}: There is a relationship between analysis failure and business requirements controlling for program size and cyclomatic complexity.	Supported at the 1% significance level
H_{5a}: There is a relationship between analysis failure and program size controlling for business requirements and cyclomatic complexity.	Supported at the 1% significance level
H_{6a}: There is a relationship between analysis failure and cyclomatic complexity controlling for business requirements and program size.	Supported at the 5% significance level
H_{7a}: There is a relationship between programming failure and business requirements controlling for program size and cyclomatic complexity.	Supported at the 1% significance level
H_{8a}: There is a relationship between programming failure and program size controlling for business requirements and cyclomatic complexity.	Not supported
H_{9a}: There is a relationship between programming failure and cyclomatic complexity controlling for business requirements and program size.	Not supported
H_{10a}: There is a relationship between total failure and business requirements controlling for program size and cyclomatic complexity.	Supported at the 1% significance level
H_{11a}: There is a relationship between total failure and program size controlling for business requirements and cyclomatic complexity.	Not supported
H_{12a}: There is a relationship between total failure and cyclomatic complexity controlling for business requirements and program size.	Not supported

Hypotheses	result
H_{1b}: There is a relationship between program age and design failure.	Not supported
H_{2b}: There is a relationship between program age and analysis failure.	Supported at 5% significant level
H_{3b}: There is a relationship between program age and program failure.	Supported at 10% significant level
H_{4b}: There is a relationship between program age and software failure.	Not supported
H_{5b}: There is a relationship between program age and hardware failure.	Supported at 5% significant level
H_{6b}: There is a relationship between program age and business requirements change.	Supported at 5% significant level
H_{7b}: There is a relationship between program age and total error .	Supported at 1% significant level

SUMMARY

Table 4.20 summarizes the results of this chapter. Logistic regression and multiple regression analysis were used to test the proposed hypotheses. Each of these hypotheses is based on a literature review of software development. These findings are particularly insightful as they are from data in the field. Much effort was involved in obtaining the necessary statistics from the program logs of the 100 COBOL programs used in this study. Implications from the hypotheses testing will be presented in chapter 5 along with limitations and future directions of the research.

Practices in different phases of software development life cycle impact the quality, performance and stability of the information systems. Software programs experience changes due to different reasons. Software attributes such as software complexity and program size determines the ease of change to the program. These behaviors also affect software volatility. In this study we examine the effect of program size, software complexity and business requirements on design failure, analysis failure and program failure.

This study shows that the variable business requirements has a significant relationship with design failure, analysis failure and programming failure. Companies face challenges, regulations, and competitions constantly, which contribute to changes in business requirements over time and correspondingly causes modifications of the information systems. Opportunities also lead to changes in business requirements. As information systems are developed to serve the needs of organizations, changes in business requirements will inevitably incur the changes in information systems. Information systems need to be designed as adaptive and flexible to cope with the changes in business requirements.

Software complexity and program size are related to modifications caused by the analysis and design phase of the software development. The more complex a system is, the more likely it will have failures due to either analysis or design. The larger the program is, the more modifications will incur because of inadequate analysis and design. For large and complex programs, resources should be allotted to ensure producing quality analysis and design of software development in order to reduce the later likelihood of fixing the programs.

No significant relationship was found between the software attributes, software complexity, and program size with programming failure. Programming practices have improved greatly over the years. As a result, programmers are well trained so that they can produce code that is not error-prone. Thus, it is reasonable to expect no relationship between the software attributes used in this study and programming error.

The findings from this study suggest that developing an adaptive system should be set as a goal to live with the changes in business environment. It is hard to foresee what will happen tomorrow. Change is the only constant. Design and analysis phases are very critical. Improved practices in these two phases will reduce the workload in the maintenance stage.

CHAPTER 5

CONCLUSIONS

Introduction

This chapter describes the contributions, limitations, future directions and the implications of this study, organized with respect to management, technology, business requirements and software development life cycle. Conclusions follow at the end.

The purpose of this research was to identify from the longitudinal data, what factors impact changes and modifications of the systems. The study proposed the relationships between software complexity, program size and business requirement changes within the analysis, design and programming stages of software development. Therefore, 12 hypotheses were tested against these relationships. Five of the 12 hypotheses are supported at 1% significance level; one is supported at 5% significance level; two are supported at 10% significance level; and another four are not supported based on the longitudinal data.

In addition, a different set of hypotheses was constructed to test the relationship between the independent variable of program age and several dependent variables, such as design failure, analysis failure, programming failure, software upgrades, hardware upgrades, changes to business requirement. The independent variable program age was also tested against the total failures, which is the summation of analysis, design, programming, software, hardware and changes to business requirement. One hypothesis is supported at 1% significance level; three hypotheses are supported at 5% significance level; one is supported at 10% significance level and two hypotheses are not supported.

Contributions

This research has contributed to the knowledge in the following domains. First, it has employed theories from the general system domain and the complexity domain, and it has extended the model of information system change developed by Heales (1998), which was grounded in Wand and Weber's (1995) model of information systems and the models of software volatility produced by Banker and Slaughter (2000) and Heales (2000). Second, this study has examined system modifications throughout the life cycle. Based on the findings, this research has confirmed that different types of change requests exist. Furthermore, activities and practices in the life cycle of software development have an impact on the system evolution. Third, it has presented the idea that system attributes affect the modifications to an information system. Fourth, this research has addressed the importance of software design phase, which affects the flexibility and adaptability of an information system. Finally, this study has analyzed software volatility using the longitudinal data.

Limitations

This study used the longitudinal data from a single company to perform analysis. Therefore, it is subject to the limitations of the single case study method. Findings from this study are not easily extrapolated to other companies and different situations. The findings, however, are based on a company that used COBOL language, so they can be generalized for COBOL environment.

The reliability of the findings can be examined by replications which can minimize the biases and errors in the study and validate the findings from this study. Procedures documented in this study can facilitate the future replications of the study. Future replications should follow the exact procedures in the previous study in order to have an equal base for comparison. Data collecting is the major stage that impacts for the results of replication.

Data collection techniques affect the internal validity of the study (Yin, 1994). The longitudinal data collected by professional EDP auditors and programmers strengthens the internal validity of the study. The methods to improve internal validity of the study are pattern-matching or time series analysis as suggested by Yin (1994). Data analysis process plays a important role in enforcing internal validity. This study used the maintenance log over a 10-year period. Accordingly, as the time series data was used in the analysis, possible pattern matches could be achieved for different periods of time. Future research intends to collect data from another company with a 10-year period to address the pattern matching issue.

In terms of construct validity in this study, establishing content validity can help improving construct validity. Content validity is enforced by the identification and classification of the change of the programs professional EDP auditors who have good understanding of the programs and business. Well-established objective measurements were employed to measure the construct system complexity, such as cyclomatic complexity used to measure complexity, the number of lines used to measure the size of the program. Even though we do not have established and validated measurement for business requirements change, the measurement we employed is considered very objective.

Future Directions

By conducting research in a particular business unit, we can build basic theories on certain phenomena. This theory can be used to explain the practices in business, and in the mean time provide effective feedback to the business community. The future research should focus on 1) validating the findings from this study; 2) examining other factors affecting the modifications of the system, 3) estimating modification costs, 4) evaluating software life-cycle cost benefits in object-oriented environments, 5) investigating the correlations of error occurrence in analysis, design and program phase of software development life cycle, 6) measuring and predicting software volatility throughout life cycle, 7) expanding the failure studies into e-commerce area, 8) furthering the understanding of how the CASE tools affect software maintenance, 9) exploring factors affecting software volatility, 10) focusing on the relationship between a complexity measure and maintenance effort.

Cumulative research on similar research questions provides validity and credibility of the findings. Generalization can be constructed based on replications of the study. Numerous articles mention the importance of each phase of the software development life cycle; however, no empirical study has focused on quantifying the effect of each of these phases so far. This study contributes to the knowledge in this stream of research. To advance our understanding in this area, linking costs to each phase of the software development life cycle with modifications in maintenance phase is another area of interest for future study.

Finding the reasons and factors of program modifications will provide effective suggestions on software development process. The interesting factors identified for future research are the mean time between modifications, the use of language and system age. System age and the mean time between modifications were factors used in Heales' recent ICIS 2000 proceedings which studies the system modifications from a single company, South Africa Mining Company. Heales's study found the significant relationships among the factors previously mentioned. Generalization is the limitation of his study. Our future study will explore the effects of these variables and add industry as a moderate variable. It is possible that certain industries have better programs and less modifications than other industries. Exploring the industry characteristics and program development attributes can give better explanations of the difference between industries and different companies.

Maintenance costs of the software programs claim a large resource of an organization. Identifying costs incurred in each phase of the software development life cycle, quantifying maintenance costs and building relationships between the costs in each phase of the software development life cycle will provide insight into project management and development. If one phase is more expensive than another phase, is it true that the corresponding errors are found more in one phase than the other? Or is more effort needed to spend in one phase of the software development?

Implications

Software development goes through analysis, design, development and maintenance. Activities carried out and requirements set in each phase determine the final functions of the systems. Any errors occurring in any of these phases will affect the performance of the systems and incur costs in keeping the systems up and running and meeting user's requirements. The eventual performance of the systems is closely related to each phase of the software development life cycle. Various reasons can require fixing and updating the systems after the systems are implemented. Both the literature and the findings from this study suggest software practices in the software development life cycle, software complexity, and business requirements are related to failures. In an effort to maximize efficiency and effectiveness of information systems, management should motivate system analysts and IS development personnel to devote more effort in design and analysis phase of software development.

Management

Even though this research did not directly study the effect of management on software project failures, the variable, the variable, changes to business requirements is closely related to management, since management determines the short term and long term business objectives, which in turn decide the parameters of information systems. Therefore, management should envision what position the company will be in the industry, and what products and services will be offered in the long run. This foresight will help the IS development team to design and develop an adaptive system to evolve with the scope and growth of an organization. Information systems serve the goals and objectives of an organization. Strategic competitiveness and return on investment can be realized by clearly defining business objectives, which, in turn, improves information system's longevity and reduces its maintenance costs.

Management is responsible for the allocation of resources. Any error or mishandling can produce adverse effect. Yourdon (1999) stated the existence of software project failures is usually the result of a conscious management decision. Software project failure is the norm. The solution will involve issues of peopleware, processes and methodologies, as well as tools and technologies. IS management addresses issues like adequate programming languages, workstations and CASE tools or may sometimes involve fighting political battles over the issues of people and processes.

Management should be centered around people. Productivity studies in the past 30 years has identified people as the greatest source of productivity improvements - far greater than the productivity gains typically achieved with development tools or new software development methodologies. Talented people, corporative and fully committed team members, decent working conditions are necessary conditions for success. Having all above factors is not sufficient to create success, but the absence of one of these elements, however, is good enough to create failure.

Management has different drivers. Bureaucracy and corporate insanity can cause failures to happen again and again. The bureaucratic procedures can be time consuming and waste time. The management should find ways to keep the team motivated. The MIS directors should be involved in the decision-making process so that they know the objectives and goals of an organization and are able to better define the systems which allow an organization to adapt and to succeed. To survive in today's turbulent e-business world, software project teams must exhibit adaptability, speed, and collaboration.

Many practitioners suggest a set of key "best practices" to ensure success. Jones (1996) has identified are twelve attributes that are indeed "necessary and sufficient" for software project success. Management needs to think about how many of them are being carried out effectively in the organization and how to know whether they were successful. What kind of metrics does the organization have, and how do the metrics compare with the metrics documented by Jones?

Management should strictly enforce documentation during project development, documenting whatever practices and processes have worked in the project, for the benefit of the next project. It has been found that interviews for failed projects rarely occur because the project team is so frustrated, exhausted and frazzled by the end of the project. A series of mini-audits throughout the project is suggested because these audits are helpful in identifying and tracing the possible causes of failures.

At last, formal risk management needs to be used to cover the following areas: agreement on interfaces, hardware interfaces, software interfaces, and interfaces between the system and other external systems; peer reviews, inspections, walkthroughs, reviews; metric-based scheduling and management; binary quality gates at the inch-pebble level, project-wide visibility of project plan and progress vs plan; defect tracking against quality targets, configuration management and people-aware management accountability.

Technology

In this study, the effect of technology on the longevity of programs is examined. The study found there is no relationship between software upgrades and program age, which means software programs can be upgraded no matter how long the program has been used. A negative relationship was found between hardware failure and program age. Over the years, hardware failures have been decreasing due to improved engineering techniques and processes.

Technology itself can not guarantee the success of the project. The absence of appropriate methodologies can transform a project into a failure. CASE tools help the management and the team with the configuration management. The functions of the system determines the selection of CASE tools to be used. Using OO and the web as building blocks for a telephone company’s billing system might sound trendy, but if the nature of the job is batch processing, it is better off to choose COBOL. It is believed that picking the right CASE tool would support high level design and communication. Any CASE tool that requires a new mode of effort or a more sophisticated way to thinking can create a problem and result in the waste of time and money.

Introducing new technology can also bring troubles into the software development. Practitioners provide suggests on what tools *must* have, what *should* have, and others *could* have. It is essential for team members to agree on common tools within the project. The project manager should have the discretion to decide what tools are critical and must have. A configuration management tool helps the development team to track the versions of the system. Thus, a configuration management tool is strongly recommended, but it needs to be well-integrated with other primary development tools in order to yield benefits.

Management plays a role in promoting efficient design. Controlling the complexity at the design phase is an effective way to improve maintainability of software. Management needs to create incentives or establish a reward policy for good designs. Alternatively, holding training sessions on the design and complexity control, and educating the developers, analysts and programmers on the cost of maintenance may result in better performance, and allow programmers and developers perform in the best interest of the employer.

Business Requirements

Evidence was found indicating that the variable, changes to business requirements, is a significant variable affecting analysis, design and programming stage the software development at 1% significance level. Although literature mentioned that business requirements is an important factor affecting the success or failure of the system, no empirical work has been conducted to operationalize the variable, changes to business requirements.

In 1999, Yourdon summarized business requirements as follows:

> In the 1970s, requirements analysis was usually considered primarily a question of *eliciting* requirements from users through interviews, brainstorming, and JAD sessions. In the 1980s, it was seen as a *documentation* problem, with great emphasis on visual modeling techniques such as structured analysis and object-oriented analysis. And in the 1990s, the emphasis shifted to *managing* requirements, with automated tools, and concepts like "triage." In the first decade of the new millennium, requirements analysis is now seen as a carefully-balanced *combination* of elicitation, documentation, and management. Focusing on one or two of these items while short-changing the other(s) is a recipe for failure.

Businesses compete in a dynamic environment. Information systems are part of the organization itself and it plays a critical role in business performance. Change is inevitable. Adhikari (1996) stated that software development should put more effort in defining business requirements. Interviews with software production mangers in Sabre and MCI all support the notion that business requirements change frequently and the initial definition of business requirements has the direct bearing on later modifications. Information systems have to be adaptive in order to meet the needs of the organization. Whenever there is a change in business requirements, it will inevitably cause changes in the coding of the systems. Business requirements determine the design and development of the systems. As the purpose of developing the systems is to meet the business requirements, capturing the right business requirements will reduce the likelihood of later modifications.

The modification of software products is considered a software enhancement process, which involves adding, changing and deleting software functionality to adapt to new and evolving business requirements (Kemerer and Slaughter 1997). For today's hectic pace of "Internet time" projects, requirements analysis is still crucial and is often more difficult than ever before. Because users are even less sure than ever before of their "true" requirements, and they often lack the patience to discuss them calmly before demanding to see some evidence of implementation. Keeping in touch with the users and identifying requirements are critical in better defining business requirements.

Incorrectly defined requirements can cause software failure. Misunderstanding user requirements is a major cause of project failure. Communication with the users promotes the better business requirement definitions. Requirement errors are usually far more expensive than other types of errors (e.g., coding bugs), because adding new requirements involves more effort and sometime may initiate a new lifecycle to begin. Well-articulated requirements provide the basis to start off the project even though there is inadequate time or resources to implement the entire system.

There are many requirement capture techniques available, such as interviewing, brainstorming, storyboarding, Role Playing, Prototyping, JAD sessions, and Modeling. The combination of these techniques can produce well-defined requirements. Requirements need to cover functional and non-functional requirements. To be specific, requirements need to address performance requirements, user interface requirements, security requirements, hardware requirements, environmental requirements, acceptance requirements, and user-friendly requirements. It is imperative to prioritize requirements if resource constraints are imposed. Prioritizing requirements can be served as a guideline in building the system and give the project team the order of importance of each requirement.

After requirements are defined and prioritized, managing requirements is the next important task to perform. CASE tools are effective for automating the modeling process. Other tools need to be used to manage a "database" of requirements. Requirement changes need to be recorded. They consume resources in an organization and may incur unpleasant consequences. The use of modern software engineering methods allows developers to prove that the system meets the requirements defined by users.

Systems Complexity

This study finds that software complexity is related to analysis and design phases of software development. The research corroborates the claims of Kemerer (1995), Banker (1993) and Banker and Slaugher (2000). This study also supports a finding of Linberg (2000). The software field needs, as computer scientists and software engineer academics so frequently advocate, better ways of approaching and building software. More accurate and honest estimates and more understanding of the complexity of scope is needed. It is necessary to develop a new theory to address software project failure, to emphasize the realistic expectations, to place the importance on a quality product, organizational congruency and the interactions between the software project and its own environment.

Large systems are more complex than small systems in terms of requirements for performance, reliability, testability, maintainability and ability for integration with existing systems. Jones (1996) suggests that software project of huge size is in deep trouble from the start, no matter which computer science techniques are or are not used. Software project complexity increases with size. Business functions and activities usually determine the scope of a software project. Management of complexity of a project is not an easy task because many factors contribute to the complexity of the software project. since maintenance activities can claim a large portion of an organization’s resources, it is necessary to control the design stage, and to motivate the designers to produce an efficient and effective design which will reduce the maintenance workload.

Large and complex programs have high risks than smaller, and less complex programs. Risk can be assessed by evaluating the complexity of the system or products being developed, as well as evaluating the client and project team environments. Product complexity can be assessed in terms of size (number of function points), performance constraints, and technical complexity. Risk assessment focuses on the complexity of the system, so as to alert the organization to potential problems.

Research confirms prior findings that complexity is associated with the volatility of the system (Barry and Slaughter 1999, Banker and Slaughter 2000; Heales 1999). The study also confirmed the prior findings. The complex programs have experienced more modifications over the years. The more complex a system is, the more volatile the system is. The increased frequency of modifications of software products translates into increased maintenance hours, which in turn, increases maintenance cost.

Software development life cycle

Each stage of software development can cause ripple or major effects on the maintenance stage. Therefore, performance measures should develop in each stage of the software development life cycle. It must be possible to measure the effects of design approaches on maintenance, and to measure the effects of maintenance approaches on future maintenance (Schneidewind 1987).

The analysis shows that design failure occurred more often than analysis failure and programming failure. In order to improve quality and reliability of software continuously throughout the software development life cycle, it is imperative to develop measurement criteria along the life cycle, especially in the early stage (Tian, 1999). The quality and reliability of software in early phases determine the faults, failures and program modifications after the implementation. The activities like inspections and evaluations in early phase of software development, can ensure software reliability.

Conclusions

This study provides insight into the factors that are related to software failures. Practices in different phases of the software development life cycle impact the quality, performance and stability of information systems. Software programs experience changes as a result of changes in business requirements and technology upgrades. Software attributes such as software complexity and program size impact the ease of making changes in the program.

The findings from this study suggest design phase and analysis phase affect the modification workload. Therefore, it is important to develop an adaptive system to copy with the changes in business environment. Improved practices in these two phases will reduce the modifications workload, hence reducing the maintenance cost on software programs. IS managers should pay special attention to the design and analysis phases where maintenance problems are likely to begin. The selection of qualified analysts and developers may contribute to sound analysis and design.

Although this study focused on a COBOL environment, we believe that the procedures used to conduct this study can be generalized to other settings. The finding from this study can be used to guide the development practices in a COBOL environment. The longitudinal data in the field study over the 10-year period highlights the strength of variables identified in this study. The findings reveal the importance of design stage and the effect of business requirement changes.

REFERENCES

Abdel-Hamid, T., & S. M. (1990). The elusive silver lining: how we fail to learn from software development failures. Sloan Management Review.

Abreu, A. F., & Conrath, D. W. (1993). The role of stakeholders expectations in predicting information systems implementation outcomes. ACM Transactions on Software Engineering and Methodology: 408-415.

Ackoff, R.L. (1960). Unsuccessful case studies and why. Operations Research, 8: 259-263.

Ackoff, R. L. (1971). Towards a system of systems concepts. Management Science, 17(11), 661-671.

Adhikari, Richard Developers benefit from a process blueprint. Software Magazine. 16(3): 59-69. 1996

Alavi**,** M. Joachimsthaler, E. A (1992). Revisiting DSS Implementation Research: A Meta-Analysis of the Literature and Suggestions for Researchers. MIS Quarterly. 16(1): 95-116.

Alder, R. L., G. & Nordgren, K. (1999). Improving risk management: Moving from risk elimination to risk avoidance. Information & Software Technology. 41(1): 29-34.

Albrecht, A. J. & Gaffney, J. E. (1983). Software function, source lines of code, and development effort prediction: a software science validation, 9(6): 639-647. IEEE Transaction on Software Engineering

Alter, S. & Ginzberg, M. (1978). Managing Uncertainty in MIS Implementation. Sloan Management Review, 20(1), 23-31.

Argyris, C. & Schon, D. A. (1974). Theory in Practice: Increasing Professional Effectiveness. Josser-Bass, San Francisco, CA, 1974.

Arthur, L. J. Software Evolution: The Software Maintenance Challenge, New York; John Wiley and Sons, Inc., 1988.

Ashby, R. (1956). Requisite variety and its implications for the control of complex Systems. Cybernatica, London.

Bailey, J. E. & Pearson, D. J (1983). Development of a Tool for Measuring and Analyzing Computer User Satisfaction. Management Science, 29(5), 530-545.

Banker, R. D., Srikant M. D., Chris, F., & Zweig, D. (1993). Software complexity and maintenance cost. Communications of ACM, 36(11): 81-94.

Banker, D. Datar, M. & Zweig, D. (1994). Software complexity and maintenance costs. Communications of the ACM. 36(11): 81-94.

Banker, R. D. Slaughter, S. A. (1997). A field study of scale economies in software maintenance. Management Science. 43(12): 1709-1725.

Banker, R. D. Davis, G. B. Slaughter, S. (1998). A Software Development Practices, Software Complexity, and Software Maintenance Performance: A Field Study. Management Science. 44(4): 433-450.

Banker, R. D., and Slaughter, S. A. "The Moderating Effects of Structure on Volatility an Complexity in Software Enhancement," Information Systems Research, September 2000.

Basili, V.R. Quantitative software complexity models: A panel summary. In V.R. Basili, Ed., Tutorial on Models and Methods for Software Management and Engineering, IEEE Computer Society Press, Los Alamitos, Calif, 1980.

Banker, R. D. Davis, G. B. & Slaughter, S. A (1998). Software Development Practices, Software Complexity, and Software Maintenance Performance: A Field Study. Management Science. 44(4): 433-450.

Barry, Evelyn and Slaughter, S.A. "measuring software volatility: a multi-dimensional approach," the 21th International Conference on Information Systems, Dec 10-13, 2000, Brisbane, Australia

Bazzana, G., Zontini, G., Damele, G. Maiocchi, M. and Giunchi, M. (1995) Applying software reliability models to a large industrial dataset. Journal of Information and Technology, 35,11/12, 669-677.

Beer, S. (1959). Cybernetics and Management. London: English Universities Press.

Beer, S. (1966). Decision and Control. New York: John Wiley & Sons.

Behrens, C. A. (1983). Measuring the productivity of computer systems development activities with function points. IEEE Transactions on Software Engineering, 9(6):

648-652.

Belady, L. A. & Leham, M. M. (1976). A model of large program development. IBM Systems Journal, 3: 225-243.

Bertalanffy, L. M. (1968). General systems theory. Braziller, New York.

Beynon-Davies, P. (1995). Information systems 'failure': the case of the London ambulance service's computer aided dispatch project. European Journal of Information Systems, 4:171-184.

Boddie, J. (1987). The Project Postmortem. *Computerworld*, 7, 77-82.

Boehm, B. M. (1979). Software engineering: R & D trends and defense needs. Research Directions in Software Technology, Cambridge, MA: MIT Press.

Bostrom, R. P., & Heinen, J. S. (1977). MIS Problems and Failures: Social-Technical Perspective, Part I: the Causes. MIS Quarterly, 11(1): 17-32.

Bostrom, R. P. & Heinen, J.S. (1977). MIS Problems and Failures: a social-technical perspective, Part I: the Causes, MIS Quarterly, 1(3), 17-32.

Bostrom, R. P., & Zmud, R. W. (1987). Information technology planning in the 1990's: directions for practice and research. MIS Quarterly, 11(1): 59-71.

Bostrom, R. (1989). P Successful Application of Communication Techniques to Improve the Systems Development Process. *Information Management. 16(5): 279-295.*

Boujarwah, A S. Saleh, K. (1997). Compiler test case generation methods: A survey and assessment. Information & Software Technology. 39(9): 617-625.

Boulding, K. (1956). General Systems Theory – the Skeleton of Science. General Systems I: 11-17.

Boynton, A.C., and Zmud, R.W. Information technology planning in the 1990's: directions for practice and research. MIS Quarterly, (1987), 11(1): 59-71.

Briand, L. C., Morasca, S., Basili, V. R .Defining and validating measures for object-based high-level design. IEEE Transactions on Software Engineering. 25(5): 722-743. 1999.

Brown, Courtney. (1995). Chaos and Catastrophe Theories. Sage Publications.

Buckley, W. (1967). Sociology and modern systems theory. Pretice-Hall, Englewood Cliffs, N.J.

Bunday, B. D., and Al-Ayoubi, I. D. (1990). Likelihood and Bayesian Estimation Methods for Poisson Process Models in Software Reliability. International Journal of Quality and Reliability Management, 7:5.

Butcher, G. *Addressing Software Volatility in the System Life Cycle*, Unpublished Doctoral Dissertation, Colorado Technical University, 1997.

Cale, E.G., Curley, K. F. (1987). Measuring implementation outcome: beyond success And failure. Information and Management, 13: 245-253

Cannon, J. A. (1994). Why it applications succeed or fail: the interaction of technical and organizational factors. Industrial and Commercial Training, 26(1): 10-15.

Cant, S. N., Jeffery, E. R. & Henderson-sellers B. (1995). A conceptual model of cognitive complexity of elements of the programming process. Information & Software Technology, 37(7): 351-362.

Cash, McFarlan, McKenney, and Vitale. (1992). Corporate Information Systems Management: Text and Cases, 3rd ed., Irwin, Homewood, Illinois.

Charette, R. N. Application Strategies for Risk Analysis, (New York: McGraw-Hill, 1990).

Checkland, P. B. (1989). Soft Systems Methodology. Human Systems Management. 8(4): 273-289.

Chidamber, S. R. Kemerer, C. F. (1994). A metrics suite for object oriented design. IEEE Transactions on Software Engineering. 20(6): 476-493.

Cox, B. (1990). The threats and opportunities for systems management. A Presentation By George Cox to Australian Foundation Members.

Coupal, D. Robillard, P. (1990). N Factor Analysis of Source Code Metrics. Journal of Systems & Software. 12(3): 263-269.

Curis, B. Sheppard, S., Milliman, P., Borst, M., & Love, T. (1979). Measuring Psychological Complexity of Software Maintenance Tasks with the Halstead and

McCabe Metrics. IEEE Transactions Software Engineering, 5(2):96-104.

Dekleva, Sasa M. (1992). The Influence of the Information Systems Development Approach on Maintenance. MIS Quarterly. 16(3): 355-372.

Dalal, S.R. and Mallows, C.L. (1988) When should one stop testing software. IEEE

Journalon Selected Areas in Communication, 83, 872-879-175.

Dalal, S.R. and Mallows, C.L. (1992) Some graphical aids for deciding when to stop testing software. IEEE Journal on Selected Areas in Communication, 8, 169-175.

DeLone, W.H. and McLean, E.R. (1992). Information Systems Success: the Quest for The Dependent Variable. Information Systems Research, 3 (1): 205-226.

Damanpour, F. Organizational complexity and innovation: Developing and testing multiple contingency models. Management Science. 42(5): 693-716. 1996

Davis, M. A. Software Requirements: Objects, Functions, and States. (Englewood Cliffs, NJ: Prentice Hall, 1993).

Davis, G. B. *Management information systems: conceptual structure, and development.* McGraw-Hill, New York, 1974.

Davis, G. B., Lee, A. S., Nickles, K. R., & Chatterjee, S. (1992). Diagnosis of an Information System Failure, Information & Management 23: 293-318.

Davis, J. S., Richard, J. L. (1988). A study of the applicability of complexity measures. IEEE Transactions on Software Engineering, 14(9):1366-1371.

Dekleva, S. M. (1992). The influence of the information systems development approach on maintenance. MIS Quarterly, 16(3): 355-372.

DeVore, P. W. (1980). Technology an introduction. Davis Publications, Inc.

Dishaw, M.T., Strong, D.M., (1998). Supporting software maintenance with software engineering tools: a computed task-technology fit analysis. 44:107-120.

Earl, M. J. (1993). Experiences in Strategic Information Systems Planning. MIS Quarterly, 17(1): 1-24.

Elias, C. (1998). Timely reminders. Incentive. 172(1): 55.

Ehrlich, W., Prasanna, B., Stampfel, J. and Wu, J. 91993) Determining the cost of a stop-test decision. IEEE Software, March, 33-42.

Ernest, R. C. Baenen**,** L. B. (1985). Analysis of Attitude Survey Results: Getting the Most from the Data. Personnel Administrator. 30(5): 71-80.

Ebert, Christof. Morschel, (1997). Ivan Metrics for quality analysis and improvement of object- oriented software. Information & Software Technology. 39(7): 497-509.

Elgin, D. S. & Bushnell R. A. (1977). The limits of complexity: are bureauracies becoming unmanageable? The Futurist, December 1977, p338.

Evanco, W. M. Poisson analyses of defects for small software components. Journal of Systems & Software. 38(1): 27-35. 1997

Ewusi-Mensah, K., & Przasnyski, Z. (1991). On Information Systems Project Abandonment: an exploratory study of organizational practices. MIS Quarterly: 67-86.

Ewusi-Mensah, K., & Przasnyski, Z. (1994). Factors contributing to the abandoned information systems development projects. Journal of Information Technology, 9: 185- 201

Ewuisi-Mensah, & Przasnyski, Z. (1995). Learning from Abandoned Information Systems Development Projects. Journal of Information Technology, 10: 3-14.

Fagan, M. E. (1999). Design and code inspections to reduce errors in program development. IBM Systems Journal, 38(2&3): 258-287).

Fenton, N. (1994). Software Measurement. (1994). Software Measurement: A Necessary Scientific Basis. IEEE Transactions on Software Engineering, 20(3): 199-206.

Flowers, Stephen. (1997). Information systems failure: identifying the critical failure factors. Failure & Lesson Learned in Information Technology Management, 1: 19-29.

Forbes, R. J. (1958). *Man and maker: a history of technology and engineering*. London: Abelard-Schuan, Ltd.

Fox, T. L. Spence, J**.** W. (1998). Tools of the trade: A survey of project management tools. Project Management Journal. *29(3): 20-27.*

Franz, C. R., & Rovey, D. (1984). An investigation of user-led system design: rational And political perspective. Communications of ACM, 27(12): 1202-1209.

Frost, P. J. (1987). Power, Politics and Influence. Handbook of Organizational Communication: an Interdisciplinary Perspective: 503-548.

Garrison, E. Wynn. (1980). Factors in Management Information System Failures. Master Thesis. University of Texas at Austin.

Gary**,** D. (1999). Customers fading away? Wireless Review. 16(21): 22-26.

Geisler**,** E. Hoang, W. (1992). Purchasing Information Technologies: Behavior Patterns in

Service Companies. International Journal of Purchasing & Materials Management. 28(3): 38-42.

Gilbert, P. (1983). Software design and development. The SRA Computer Science Series.

Gill, G. K. Kemerer, C. F. (1991). Cyclomatic Complexity Density and Software Maintenance Productivity. IEEE Transactions on Software Engineering. 17(12): 1284-1288.

Ginzberg, M. J. (1981). Early diagnosis of MIS implementation failure promising results and unanswered questions. Management Science 28(7): 459-478.

Ginzberg, M. & Schultz, R. (1987). The Practical Side of Implementation. Interfaces, 17(3), 1-5.

Gladden, G. R. (1982). Stop the Life Cycle, I Want to Get off. Software Engineering Notes., 7(2), 35-39.

Glaser, G. (1984). Managing Projects in Computer Industry. Computer, 17, 45-53.

Glass, R. L. (1998). Software Runaways. Prentice Hall.

Glass, R. L (2000). Evolving a New Theory of Project Success, Communications of the ACM, 42(11), 17-24.

Goel, A. L. and Okumoto, K. (1979) Time-dependent error-detection rate model for software and other performance measures. IEEE Transaction on Reliability, 28, 206-211.

Goff, L (1993). Leslie Mastering project management. Computerworld. 27(51): 76-78.

Gonzalez, R. R. (1995). A unified metric of software complexity: measuring productivity, quality and value. Journal of Systems Software, 29:17-37.

Gorla, N. (1991). Techniques for Application Software Maintenance. Information & Software Technology. 33(1): 65-73.

Gray, P. The state of software development. Information Systems Management. 15(1): 88-92. 1998.

Green, M. and MacColl, G. (1987). Mission Impossible.

Guinan, P. J. Cooprider, J. G. Faraj, S. (1998). Enabling software development team performance during requirements definition: A behavioral versus technical approach. Information Systems Research. 9(2): 101-125.

Haffer, J. & George, J. & Valacich, J. (1996). Modern system analysis and design. The Benjamin/Cummings Publishing Company, Inc.

Halstead, M H. (1977). Elements of software science. Eksevier/North-Holland.

Harrison, W., & Cook, C. (1977). A micro/macro measure of software complexity. Journal of Systems and Software, 7: 213-219.

Harrison, W. & Magel, K. (1981). A Complexity Measure Based on Nesting Level. ACM SIGplan Notices, 16(3):63-74.

Hart, K. E Introducing Stress and Stress Management to Managers. Journal of Managerial Psychology. 5(2): 9-16. 1990.

Harvey, A. (1970). Factors Making for Implementation Success and Failure. Management Science, 16 (6): 312-321.

Hass, J. Lagarias, J. C. and Pippenger, N. (1999). The computational complexity of knot and link problems. Journal of the Association for Computing Machinery. 46(2): 185-211.

Hay, Gordon. Munoz, Rick Establishing an IT architecture strategy. Information Systems Management. 14(3): 67-69. 1997

Hebert, M**.** Benbasat**,** I. (1994). Adopting information technology in hospitals: The relationship between attitudes/expectations and behavior. Hospital & Health Services Administration. *39(3): 369-383.*

Heals, J. “Factors affecting information system volatility,” in *Proceedings of Twentieth International Conference on Information Systems*, Brisbane, Australia, Dec 10-13, 2000.

Hetzler, S. A. (1973). Applied measures of promoting technological growth. Boston: Toutledge and Kegan Paul.

Highsmith, J. A. “Adaptive Software Development : A Collaborative Approach to Managing Complex Systems.” 1999. Dorset House.

Hoffer, E. (1973). Relfections on Human Conditions.

Hossain, S.A. and Dahiya, R.C. (1993) Estimating the parameters of a non-homogeneous Poisson process model for software reliability. IEEE Transactions on reliability, 42, 604-612.

Hong, G Y. Xie, M. Shanmugan, P A statistical method for controlling software defect detection process. Computers & Industrial Engineering. 37(1,2): 137-140. 1999

Huq, F. Testing in the software development life-cycle: Now or later. International Journal of Project Management. 18(4): 243-250. 2000

Hurwitz, J. S. (1998). Scale back software complexity. Informationweek. (678): 198.

Irwin, J. G. Hoffman, J. Geiger, S. W. The effect of technological adoption on organizational performance: Organizational size and environmental munificence as moderators. The International Journal of Organizational Analysis. 6(1): 50-64. 1998

Ives, B. and Olson, M. (1981). User Involvement in Information Systems: A Critical Review of the Empirical Literature, CRIS, 6BA 81-07, New York University, New York.

Ives, B., & Olsen, M. (1984). User Involvement and MIS Success: A Review of Research. Management Science, (1984), 30: 586-603.

Ives, S. (1994). High performance customer management. Logistics Information Management. 7(2): 45-51.

Jiang, J. J., Klein, G., Balloun, J. L., & Crampton, S. M. (1999). System Analysts Orientations and Perceptions of System Failure. Information and Software Technology, 41: 101-106.

Jiang, J., Margulis, S., Pick, R., & Motvani, J. (1997). System Developers' Judgements of System Development Use and Failure. International Journal of Computer Applications in Technology, 10 (5/6): 300-307.

Jones, L. H., and Kydd, C. (1988). Information Processing Framework for Understanding Success and Failure of MIS Development Methodologies. Information & Management: 263-271.

Jones, T. C. (1978). Measuring Program Quality and Productivity. IBM System Journal, 17(1):39-63.

Jones, C. (1996). Large software system failures and successes. American Programmer: 3-9.

Kagurazaka, S. (1996). Some discussions on Systems Failure, IEEE International Conference on Systems, Man, and Cybernetics, 4: 3130-3131.

Kafura, D. & Reddy, G. R. (1987). The use of software complexity metrics in software maintenance. IEEE on Software Engineering, 13(3): 335-343.

Kapur, P.K. and Bhalla, V.K. (1992) optimal release policies for a flexible software reliability growth model. Reliability Engineering and Systems Safety, 35, 45-54.

Kawalek, P & Wastell, D. G. (1999). A case study evaluation of the use of the viable system model in information systems development. Journal of Database Management. 10(4): 24-32.

Keider, S. P. (1984). Why system development projects fail. Journal of Information Systems Management, 1: 33-38.

Keil, M. (1995).Escalation of commitment in information systems development: A comparison of three theories. Academy of Management Journal. (Best Papers Proceedings. (1995): 348-352.

Keil**,** M. (1995). Pulling the plug: Software project management and the problem of project escalation. MIS Quarterly. 19(4): 421-447.

Kemerer, C. F. (1995). Software complexity and software maintenance: a survey of empirical research. Annals of Software Engineering. 1: 1-22.

Kemerer, C. F. and Slaughter, S. (1997). Determinants of software maintenance profiles: An Empirical investigation. Software Maintenance: Research Practice 9:235-251.

Khoshgoftaar, M., Munson, C. & Lanning, L (1994). Alternative approaches for the use of metrics to order programs by complexity. Journal of Systems & Software. 24(3): 211-221.

Khoshgoftaar, M. & Lanning, L (1995). A neural network approach for early detection of program modules having high risk in the maintenance phase. Journal of Systems & Software. 29(1): 85-91. 1995

Kitchenham, B. Pfleeger, S. Fenton, N. (1997). Reply to: Comments on "Towards a Framework for Software Measurements Validation". IEEE Transactions on Software Engineering. 23(3): 189.

Klein, H. K. Hirschheim**,** R. (1985). A Fundamental Issues of Decision Support Systems: A Consequentialist Perspective. Decision Support Systems. 1(1): 5-23.

Kling, R. (1974). Computer and Social-Power. Computers and Society, 5(3).

Kling, R. and Gerson, E. M. (1977). The Social Dynamics of Technical Innovation in the Computer World. Symbolic Interaction, 1, 132-146.

Klir, G. (1985). Complexity: some general observations. System Research, 2(2): 131-140.

Klir, G. (1985). Architecture of systems problem solving. A division of Plenum Publishing Corporation, New York.

Lai, L. (1997). A synergistic approach to project management in information systems development. International Journal of Project Management. 15(3): 173-179.

Leung, Y.W. (1992) Optimal software release time with a given cost budget. Journal of Systems and Software, 17, 233-242.

Lew K. L., Dillon, T. S., & Forward, K. E. (1988). Software complexity and its impact on software reliability. IEEE, 14(11):1645-1655.

Lin, E. Hsieh, C. (1995). The seven deadly risk factors. Journal of Systems Management. 46(6): 48-52.

Low G. C. & Jeffery D. R. (1990). Function points in the estimation and evaluation of the software process. IEEE Transactions on Software Engineering, 16(1): 64-71.

Lucas, H.C. Jr. (1975). Why Information Systems Fail. Columbia University Press, New York.

Lucas, H.C. Jr. (1975). Implementation – The Key to Successful Information Systems, Columbia.

Lucas, H. C Jr. Spitler, V K. (1999). Technology use and performance: A field study of broker workstations. Decision Sciences. 30(2): 291-311.

Lycett, M. & Paul, R. J. (1999). A perspective on the challenge of evolutionary complexity. European Journal of Information Systems. 8(2): 127-135.

Lyytinen, K., & Hirschheim, R. (1987). Information Systems Failures – A Survey and Classification of the Empirical Literature. Oxford Surveys in Information Technology, 4: 257-309.

Lyytinen, K. (1988) Expectation Failure Concept Systems Analysts View of Information System Failures: Results of an exploratory Study. Information and Management, 14(1): 45-56.

Mata-Toledo, R. A. Gustafson, D. A. (1992). A Factor Analysis of Software Complexity Measures. Journal of Systems & Software. 17(3): 267-273.

McDonell, S. G. (1991). Rigor in software complexity measurement experimentation. Journal of Systems Software, 16:141-149.

Markus, M. L. (1983). Power, Politics and MIS Implementation. Communications of the ACM, 26(6), 430-444.

Markus, M. L., & Bjorn-Andersen, N. (1987). Power over Users: Its Exercise by System Professionals," Communications of the ACM, 30(6): 498-504.

Markus, M. L. & Robey, D. (1988). Information Technology and Organizational Change: Causal Structure in Theory and Research, Management Science, 34(5): 583-598.

Marsden, James R. Tung, (1999). Y Alex The use of information system technology to develop tests on insider trading and asymmetric information. Management Science. 45(8): 1025-1040.

Martin, J., and McClure, C. Software Maintenance: the Problem and its Solutions, Englewood Cliffs, NJ: prentice Hall, 1983.

Martinez**,** E. V (1994). Avoiding large-scale information systems project failure: The importance of fundamentals. Project Management Journal. *25(2): 17-25.*

Masoner, M. M. & Nicolaou, A. I. (1996). An empirical examination of information systems development strategies in organizational contexts. Mid-Atlantic Journal of Business. 32(3): 205-219

Mate-Toledo, R. A., & Gustafson, D. A. A factor analysis of software complexity measures. Journal of Systems Software. 17:267-273.

May, R. M. (1972). Will large complex system be stable? 238: 413-414.

McCabe T J. (1976). A complexity measure. IEEE Transaction Software Engineering 11(2), 308-320.

McColl, R. B., & McKim, J.C. Evaluating and extending Npath as a software complexity measure. Journal of Systems Software. 17:275-279.

McComb, D. & Smith J. Y. (1991). System Project Failure: The Heuristics of Risk. The Journal of Information System Management, 8(1): 25-34.

McFarlan, F. W. (1981). Portfolio Approach to Information Systems. Harvard Business Review, 65, 68-74.

McWilliams**,** G. (1994). Mom and Pop go high tech. Business Week. *(3400): 82-90.*

Mellor, P. (1987). Software Reliability Modeling: the State of the Art. Information and Software Technology. 29(2): 81-98.

Meredith, J. (1988). Project monitoring for early termination. Project Management Journal, 19(5): 31-38.

Mills, H. D. (1975). Software Engineering, Science, No 4283:1149-1205.

Mills, H. D. (1976). Software Development, IEEE Transactions on Software Engineering, SE-2, 4:265-273.

Misra, P. N. (1983) Software reliability analysis. IBM Systems Journal, 22, 262-270.

Morgan, H. & Robey, D. (1983). Understanding MIS failures. Data Base, 5(1).

Morgan, H. & Soden, J. (1973). Understanding MIS Failures. Data Base, 5, 157-171.

Munson, J. C. & Khoshgoftar. T. M. Applications of a relative complexity metric for software project management. *Journal of System and* Software, 12, 283-291.

Munson, J. C Software faults, software failures and software reliability modeling. Information & Software Technology. 38(11): 687-699. 1996

Necco**,** Charles R. Gordon, Carl L. Tsai, Nancy W. (1987). Systems Analysis and Design: Current Practices. MIS Quarterly. *11(4): 461-476.*

Nejmeh, B. A. (1998). NPATH: a measure of execution path complexity and its applications, Communications of ACM, 31(2), 188-200.

Newman, M. & Sabherwal,R. (1996). Determinants of commitment to information systems development: A longitudinal investigation. MIS Quarterly. 20(1): 23-54.

Oakley, K. (1949). *Man the tool maker*. London.

Ohba, M. (1984). Software reliability analysis model. IBM Journal of Research Development, 28, 428-443.

Orli**,** R. J (1989). Battling Project Escalation. Computerworld. *23(14): 64,66.*

Osborne, W. M. (1988). All About Software Maintenance: 50 Questions and Answers. Journal of Information Systems Management. 5(3): 36-43.

Palvia, P., Patula, A., & Nosek, J. (1995). Problems and issues in application Software Maintenance Management. Journal of Information Technology Management, 6(3): 17-28.

Palvia & Hunter (1996). Information systems development: A conceptual model and a comparison of methods used in Singapore, USA and Europe. Journal of Global Information Management. 4(3): 5-16

Paulk, M. C. and Weber, B. C., Chrissis, M. B., et al. The Capability Maturity Model: Guidelines for Improving the Software Process. (Reading,MA: Addison-Wesley, 1995)

Pfeffer, J. (1992). Managing with Power: Politics and Influence in Organizations. Harvard Business School Press.

Pham, H. (1993) Software Reliability assessment: imperfect debugging and multiple failure types in software development. Technical report, EG&G-RAAM-10737, Idaho National Engineering Laboratory.

Pinto, J. K. & Mantel, S. J. (1990). The Cause of Project Failure. IEEE.37(4), 269-275.

Pinto , J. K. & Slevin, D. P. (1987). Critical Factors in Successful Project Implementation. IEEE Transaction Engineering Management. 32, 22-27.

Poo, C-C D. Layzell, P J. (1990). Enhancing Software Maintenance Through Explicit System Representation. Information & Software Technology. 32(3): 176-186.

Poo, C-C D. Layzell, P J. (1992). An evolutionary structural model for software maintenance. Information & Software Technology. 18: 113-123.

Poulymenakou A., & Holmes A. (1996). A contingency framework for the investigation of information systems failure. European Journal of Information Systems 5: 34-46

Pursell, W. Technology and western civilization.

Reich**,** B. H.**.** Benbasat**,** I. (1996). Measuring the linkage between business and information technology objectives. MIS Quarterly. *20(1): 55-81.*

Robey, D. (1987). Implementation and the organizational impacts of information systems. Interfaces, 17(3), 72-84.

Robey**,** D. Rodriguez-Diaz, A. (1989). The Organizational and Cultural Context of Systems Implementation: Case Experience from Latin America. Information Management. 17(4): 229-239.

Roman, D. (1983). A Proposed Project Termination Audit Model. IEEE Engineering Management, 30(3):123-127.

Rothfeder, J. (1987). Communications Management: A Dangerous Constriction. Network World. *4(4): 29.*

Rouse, R. (1980). Human Detection and Diagnosis of System Failures.

Royce, W. W. Managing the Development of Large Software Systems: Concepts and Techniques. *Proceeding* IEEE Wescon: 1978, 1-9.

Sanders, G. L. (1984). MIS/DSS Success Measure. Systems, Objectives, Solutions. 4, 29-34.

Santos, D. & Hawk S. R. (1988). Differences in analyst's attitudes towards information systems development: evidence and implementations. Information & Management, 24(1): 31-41.

Schmitt, J., & Kozar, K. (1978). Management's Role in Information System Development Failures: A Case Study. MIS Quarterly: 7-16.

Schneidewind, N. F. The State of Software Maintenance. IEEE Transactions on Software Engineering. SE-13(3): 303-310. 1987

Schneberger, S. L (1997). Distributed computing environments: Effects on software maintenance difficulty. Journal of Systems & Software. 37(2): 101-116.

Schoderbek, schoderbek, & Kefalas. Management Systems, Forth Edition (1990).BPI IRWIN, Homewood, IL 60430.

Senn, J. (1978). A management view of system analysts: failures and shortcomings. MIS Quarterly, 2(3): 25-32.

Shannon, C. E. & Weaver, W. (1949). *The Mathematical Theory of Communication,* University of Illinois Press, Urbana.

Smith, Peter. (1998). *Explaining Chaos*. Cambridge University Press.

Simpson, R. L. (1992). Learning from Software Development Failures. Nursing Management 23(8): 30-31.

Simon, H. A. (1969). *The Sciences of the Artificial*. MIT Press, Cambridge, Mass.

Skok & Scarre, (1992). Gina Computer Project Management: An Information Systems Development Methodology as the Critical Success Factor - Myth or Universal Truth? Management Research News: Mrn. 15(2): 6-13.

Stanwick, P. A. Stanwick, S. D. The relationship between corporate social performance and organizational size, financial performance, and environmental performance: An empirical examination. Journal of Business Ethics. 17(2): 195-204. 1998.

Sumner, M. (1999). Critical Success Factors in Enterprise Wide Information. ACM Transactions on Software Engineering and Methodology: 297-303.

Symons, C. R. (1988). Function point analysis: difficulties and improvements. IEEE Transaction on Software Engineering, 14(1):2-11.

Tian, J. (1999). Measurement and continuous improvement of software reliability throughout software life-cycle. Journal of System and Software, 47, 189-195.

Titus, J. (1997). System failures require thorough analysis. Test & Measurement World. 17(10): 41-44.

Tsaur, W. & Horng, S. (1998). A new generalized software complexity metric for distributed programs. Information & Software Technology. 40(5,6): 259-269.

Turner, J. A. (1982). Observations on the use of behavioral models in information systems research and practice. Information & Management, 15(3):207-213.

Vessey, I. & Weber, R. (1983). Some factors affecting program repair maintenance: an empirical study. Communications of the ACM, 26(2): 128-134.

Valach, Klir. (1967). *Cybernetic Modeling*. D. Van Nostrand Company. New Jersey.

Waelchli, E. (1989). The VSM and Ashby's law as illuminants of historical management thought. *The Viable System Model*. Chilchester, England: John Wiley.

Walters, S A. Broady, J E. Hartley, R J. (1994). A review of information systems development methodologies. Library Management. 15(6): 5-19.

Wand, Y., and Weber, R. 1995. On the deep structure of information systems, Information Systems Journal, 203-223.

Weill, P. & Olson, M. H. (1989). Managing Investment in IT: Mini CASE Examples and implications. *Management Information Systems Qua*rterly, 13(1).

Weyuker, E. J. (1988). Evaluating software complexity measures. IEEE, 14(9):1357-1365.

Weyuker, E. J. (1996). Using failure cost information for testing and reliability Assessment. ACM Transactions on Software Engineering and Methodology, 5(2) : 87-98.

Wicken, J. S. (1954). Entropy and information: suggestions for common language. Philosophy of Science, 54, 176-193.

Wilson, Francis A. Wilson, John N The role of computer systems in organizational decision making. Information Society. 10(3): 173-180. 1994

Winter, C., Brown, H. & Checkland, P. B. (1995). A role for soft systems methodology in information systems development. European Journal of Information Systems. 4(3): 130-142.

Wright, J. (1998). Software Failure: Management Failure: Amazing Stories and Cautionary Tales. International Review of Law Computers & Technology. 12(1): 188-191.

Xie, M. Hong, G Y A study of the sensitivity of software release time. Journal of Systems & Software. 44(2): 163-168. 1998 Dec.

Yamada, S. and Osaki, S. (1985). Software reliability growth modeling: models and applications. IEEE Transactions on Software Engineering, 11, 1431-1437.

Yamada, S. and Ohtera, H. (1990) Software reliability growth models for testing-effort control. European Journal of Operational Research, 46:343-349.

Yau, S. S. Collofello, J. S. (1985). Design Stability Measures for Software Maintenance. IEEE Transactions on Software Engineering. SE-11(9): 849-856.

Yau, S. S., Nicholl, R. A. Tsai, J. J-P. Liu, S. An Integrated Life-Cycle Model for Software Maintenance. IEEE Transactions on Software Engineering. 14(8): 1128-1144. 1988.

Yin, R. (1994). Case study research: design and methods. Sage Publications, 2nd Edition.

Yourdon, E. (1999). Death March. Prentice Hall.

Zhang, X. Pham, H. A software. cost model with warranty cost, error removal times and risk costs. IIE Transactions. 30(12): 1135-1142. 1998 Dec.

Zhang, X. and Pham, H. An analysis of factors affecting software reliability. Journal of Systems & Software. 50(1): 43-56. 2000 Jan 15.

Zmud, R. (1979). Individual differences and MIS success: a review of the empirical literature. Management Science, 25(10): 966-979.

Zmud, R.W. (1982). System Implementation Success – Behavioral/Organizational Influence and Strategies for Effective Change. Teaching Informatics Courses, Jackson, North Holland: 125-142.

Zuse, H. (1997). Property-based Software Engineering Measurement. <u>IEEE Transactions on Software Engineering</u>. 23(8): 533.