Software Engineering and Research Issues in Accounting Information Systems

Daniel E. O'Leary

ABSTRACT

This paper asserts that certain research topics and methodologies from software engineering can benefit from the specificity of context provided in accounting information systems. In addition, the paper asserts that many software engineering research questions, research issues, and research databases can be used to analyze important accounting information systems questions. Further, the paper asserts that accounting information systems researchers need to be aware of issues apparently being decided by software engineers that can directly impact accounting information systems. Accordingly, this paper reviews topics from software engineering in the areas of designing software, developing software, securing software and testing software. It also analyzes the relationship between complexity and reliability for accounting information systems. For each of these topics, research issues in software engineering are elicited. Corresponding research issues in accounting information systems are elicited. Finally, the paper investigates the use of prototyping and the development of research databases in accounting information systems.

1. INTRODUCTION

A number of issues from various branches of computer science have found their way into accounting information systems (AIS) research. For example, research on database systems has been used to delineate research efforts in accounting information systems (e.g., McCarthy [1982]). However, one branch of computer science, software engineering, apparently has contributed few research issues to accounting information systems.

The purpose of this paper is to attempt to remedy this difficulty. Accordingly, the focus of this paper is two-fold. First, it is concerned with briefly reviewing selected topics in software engineering and selected research issues within those software engineering topics. Second, it is concerned with applying those software engineering issues to accounting information systems research problems.

Daniel E. O'Leary is Assistant Professor of Accounting, School of Accounting, University of Southern California, Los Angeles, California 90089-1421.

The author would like to acknowledge the extensive comments by Bill McCarthy on earlier versions of this paper. Of course any limitations remain due to the author.

In this paper, a broad view of accounting information systems is used. Reference to accounting information systems will be assumed to include traditional accounting information systems and more recent systems such as accounting-based decision support systems and expert systems.

1.1 Software Engineering

In one of the first conferences on software engineering, Bauer (as quoted in Pressman [1987, p. 19]) defined software engineering as "the establishment and use of sound engineering principles in order to obtain economically, software that is reliable and works efficiently on real machines." More recently, Card et al. [1987, p. 845] noted that, "the basic goal of software engineering is to produce the best possible software at the lowest possible price." In a survey of software engineering, Zelkowitz [1978] called software engineering multidisciplinary. As Zelkowitz [1978, p. 198] notes, software engineering "...uses mathematics to analyze and certify algorithms, engineering to estimate costs and define trade-offs, and management science to define requirements, assess risks, oversee personnel, and monitor progress."

Production of software requires a number of interlocking steps, typically referred to as the "classic" life cycle of software (Pressman [1987]). These steps include, analysis of the system needs, design of a system to meet those needs, programming of the system to meet those needs, securing the system from unwanted tampering, testing the system to ensure it does what it is supposed to do in a correct manner and then maintaining the system. The elements of the life cycle form the focus of much of this paper.

Two of the primary variables that impact the ability of a software engineer to develop "the best possible software" are the complexity of the problem being analyzed and the required reliability necessary for the particular application. These variables affect the entire software development process.

As with other engineering disciplines, prototyping of a system frequently is used as a means of demonstrating "proof of concept" and thus is an important research tool. However, prototyping also provides an alternative to the classic system life cycle. In the prototyping life cycle, the design and development of a system is less sequential and there is more interaction between developing a model of the system and analysis of the system needs.

1.2 Scope of the Paper

The basic nature of software engineering suggests that we focus on research issues dealing with the complexity and reliability of software environments in the design, development, securing and testing of specific software systems. Since artificial intelligence increasingly is becoming an important part of software engineering (Simon [1986] and Goldberg [1986]), both traditional software engineering and newer artificial intelligence approaches to the solution of these issues will be examined. As a result, the paper will include some discussion of some selected recent contributions in software engineering and AIS using artificial intelligence and expert systems. For a more detailed discussion of artificial intelligence and expert systems in AIS and auditing, the reader is referred to Amer et al. [1987] or Gal and Steinbart [1987].

Because software directly interfaces with the system user, user interface issues are critical to software engineers (Draper and Norman [1985], Laughery and Laughery [1985]). However, because of the recent discussion by Reneau and Grabski [1987], these issues will not be discussed in this paper.

In addition, this paper will not be concerned directly with database systems research, although software engineering journals frequently include papers relating to database issues. Further, the paper will not concern itself with communications issues although, clearly, matters such as local area networks (Tripathi et al. [1987]), are of concern to software engineers.

The areas of software engineering that have been investigated in this paper were chosen because of the likelihood that they could be of interest or use to accounting

information systems researchers. As a result, certain topics such as software maintenance (Scheidewind [1987]) have not been included in this discussion.

Finally, there is a vast software engineering literature. As a result, this paper can only examine a few selected references. Unfortunately, this means that many appropriate papers will have been neglected.

1.3 Contentions of this Paper

This paper has at least three contentions. First, certain research topics, methodologies and database approaches used in software engineering research can benefit from the specificity of context offered by accounting information systems. Accounting applications have unique characteristics and needs, which can drive the design, development, securing and testing of those systems. Accounting information systems often are the only computerized applications of "for profit" businesses. Accounting systems must meet the needs of both internal and external users and they must include appropriate controls. Accounting information systems provide information for financial reporting purposes and for decision making purposes. Further, accounting systems are not limited only to the software, but are embedded in an organizational context of compensating controls.

In addition, accounting decisions are different than many other decision making domains (O'Leary [1987]). In contrast to some scientific decisions that have a unique solution, many accounting decisions have multiple solutions. Further, a good decision does not necessarily result in good consequences. Decisions are based on the information available at the time they are made; this does not guarantee desirable consequences at a later time. In addition, many accounting decisions can have measurable, costly consequences.

The paper's second contention is that software engineering research questions, research databases and research methods can be used in the development and analysis of research questions and databases of interest to accounting information systems researchers. For example, although virtually all the management consulting departments of major accounting firms implement accounting systems, there is little in the literature about the factors that impact cost and time of implementation. However, software engineers have analyzed the cost and time required in order to implement other types of software. Thus, the approaches of software engineers may aid accounting information system researchers in assessing these issues.

The paper's third contention is that many issues being discussed and decided by software engineers affect accounting information systems. For example, software engineers are concerned with the development of standards for mainframe applications software (e.g., Software International Corporation [1986]). Such standards could have a major impact on accounting information systems. Further, there is some work being done on the automatic deletion of obsolete information (Haberman [1985]). Automatic deletion of information may be inappropriate for accounting data because of unique accounting and auditing needs.

1.4 Outline of this Paper

This paper proceeds as follows. Section 1 provides an introduction by defining software engineering and eliciting the contentions of the paper. Section 2 analyzes reliability and complexity. A framework is devised to explore the relationship between these two important aspects of software. Section 3 investigates program design and development issues. Section 4 analyzes the security concerns in software engineering. Section 5 reviews software testing. Section 6 studies one of the unique development methodologies used in software engineering, prototyping. Section 7 analyzes the use and implications of software engineering databases. Section 8 reviews some of the resources available to the reader for further investigation in software engineering. Section 9 provides a conclusion to the paper.

2. COMPLEXITY AND RELIABILITY

Two of the primary factors that affect

the development of computer software are complexity and reliability. Each of these factors impacts each step of the software life cycle. For example, the amount of testing required of a system is a function of the complexity of the system and the necessary reliability.

2.1 Characteristics of Complexity

Complexity is defined by Conte et al. [1986, p. 17] as "...a characteristic of the software interface which influences the resources another system will expend or commit while interacting with the software." This definition is output oriented and does not reflect the factors that relate to complexity.

However, most of the measures of complexity are *size* related (Conte et al. [1986, p. 187]). Generally, the larger the program, the more complex it is. In addition, the act of partitioning a program into individual *modules* can reduce the complexity (Myers [1978]). Thus, the number of modules is related to the complexity of the software. In addition, the *coupling* (or interaction) between the modules impacts complexity (Conte et al. [1986]): the more coupling the more complex the software.

Because of the different dimensions of complexity, software engineering researchers have developed a number of metrics to measure complexity (see Conte et al. [1986] for a summary of such measures). These measures are then used in studies that relate complexity to various factors. There are a number of examples of such studies. Adrion et al. [1982] summarize much of the literature on the impact of complexity on program testing. Carver and Simmons [1985] discuss impact of programming methodology on the program complexity.

2.2 Reliability

Myers [1976, p. 6] defines software reliability as "...the probability that the software will execute for a particular period of time without a failure, weighted by the cost to the user of each failure encountered." While, Zelkowitz [1978, p. 201] notes that, "...a reliable program need not be correct, but gives acceptable answers even if the data or environment do not meet the assumptions made about them." Thus, software reliability is a function of the extent of the impact that the errors have on the user of the software, not of the size of the error. Small errors can have a large impact causing a substantial loss in reliability. In an example given in Myers [1976], the United States' first space mission to Venus failed because of a missing comma in a FORTRAN DO statement.

Substantial research efforts have gone into methods designed to ensure that reliable software can be built, to estimate the reliability and then to test it. Wasserman et al. [1986] discuss a methodology for systems specification, design and implementation that focuses on the critical success factor of reliability. Ross [1985] develops a model for statistical estimation of reliability. Dunn and Ullman [1982] summarize the testing of the reliability of software.

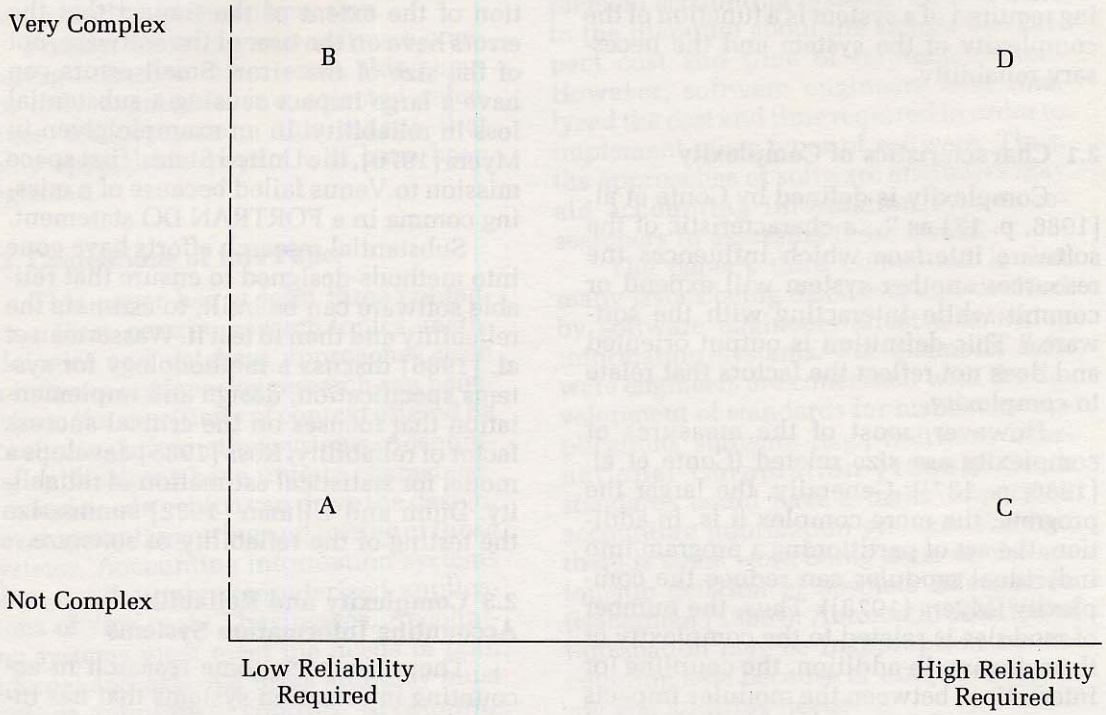
2.3 Complexity and Reliability in Accounting Information Systems

There has been some research in accounting information systems that has integrated, to a certain extent, complexity and reliability. Cushing [1974] developed a model of an accounting information system using a mathematical model of reliability theory. Later Bodnar [1975] and others extended Cushing's original work to more complex systems. Thus, there appears to be an important relationship between these two variables in AIS.

Gorry and Scott Morton [1971] developed a two dimensional framework to analyze management information systems. The X-axis on their framework measured managerial activity (operational control, management control and strategic planning) and the Y-axis measured the structural nature of the problem (structured, semi-structured and unstructured). In the same sense, complexity and reliability can be used to develop a framework to characterize software. The axis on the framework would include complexity and reliability (see Figure 1). As above, complexity would refer to the size of the application, the extent of modularity (e.g., accounts receivable, accounts payable, inventory, etc.),

FIGURE 1

COMPLEXITY AND RELIABILITY



and the extent of the interaction between modules (e.g., if the accounting system is not integrated then the software would have the interactions between the modules). Reliability would refer to the need for the system to provide "acceptable" answers to the user, even if the answers are not exactly correct.

In this schema, accounting programs that produce accounting data may be unreliable without causing substantial problems. In this case, if the system produced data that was incorrect then the problems clearly would be attributable to system failure. This would ensure that incorrect data did not receive any further processing and thus would meet the auditing needs of accounting data. Alternatively, in accounting information systems software, used for decision making, reliability of systems would become important. Thus, even if the system had an error, then the user would still be able to make acceptable decisions in a timely manner.

In terms of complexity, accounting general ledger packages that are *not* integrated would probably be regarded as "not complex." An integrated package would be more complex. However, no accounting software is likely to be as complex as something like the Strategic Defense Initiative ("Star Wars") system.

Thus, a general ledger for a personal computer environment might be around point A, while point D might represent programs like the well-publicized "Star Wars" effort. Point B might be a taxation adviser, while C might be an automated teller machine.

3. PROGRAM SPECIFICATION, DESIGN AND DEVELOPMENT

Typically, one of the first tasks in program development is development of software specifications. Fickas et al. [1987] investigate the process of how a problem description can be acquired from a client

and turned into formal software specification. They explore the use of verbal protocols of system analysts to obtain an understanding of the design process. In so doing, they are building and testing a system designed to automate the analysis phase of software development. The system goes under the assumption that the customer may "not always be right," and that the expert systems analyst can help guide the customer to a better solution than if the customer simply provided specifications.

Given the program specifications, software design is the process which translates those specifications into a detailed design representation of the software. In order to facilitate the development of a detailed design, various tools and techniques have been developed. A number of those techniques (e.g., modular programming, HIPO and other techniques) are discussed in Yau and Tsai [1986].

Once the detail design has been completed, the system can be programmed. There has been a substantial amount of research on the development of different types of programming methodologies, the impact of different computer languages and the management of programmers. In addition, recent studies have begun to examine the nature and impact of computer programming expertise.

Structured programming (Dijkstra [1976] is a popular programming methodology. However, as noted by Myers [1978], there is no concise and universally agreed upon definition. As Myers [1978, p. 144] notes, "At one extreme, it is used as a generic description of anything good under the sun in the field of software development, and at the opposite extreme, structured programming is simply coding without the use of the GO TO statement." However, some researchers appear to use structured programming and a top-down modular design in conjunction with structured programming (e.g., Vessey and Weber [1984]). At any rate, in spite of its broad acceptance, relatively few empirical studies have obtained evidence that either supports or refutes the use of structured programming. Vessy and Weber [1984] survey the literature on structured programming and provide a framework for future work in the area.

Software engineering researchers also have studied the impact of using high level programming languages on programmer productivity (Jones [1986]). Jones [1978] found that productivity, as measured by lines of code produced, decreases with the use of higher order languages. However, Gaffney [1986] suggests that rather than *human-produced* code, the measure should be lines of *machine* code.

Researchers also have analyzed differences among programmers and the impact of management techniques on programmers. Curtis [1981] investigated the notion and impact of programmer variability. Jeffery and Lawerence [1985] found that attitude toward the supervisor was significantly correlated with productivity.

Software engineering researchers have analyzed more than just the use of various techniques. Soloway and Ehrlich [1984] investigated the claim that advanced computer programmers use and have (1) programming plans (program fragments that would be frequently used) and (2) rules of computer programming discourse (rules that specify the conventions of programming). They found that advanced computer programmers have "strong" expectations about what computer programs should look like. When those expectations are violated, even in minor ways, the advanced programmer's performance drops dramatically, so much so that it reduces to that near the novice programmers.

Software engineering researchers also have investigated an issue of great importance to software developers: the ability to accurately estimate costs of designing and developing software. There has been substantial model development and testing on a wide range of databases. Recent research in this area includes Kitchenham and Taylor [1985], Conte et al. [1986] and Kemerer [1987].

3.1 Specification, Design and Development in Accounting Information Systems

There are a number of potential research issues for accounting information

systems researchers that can be gleaned from the above discussion. Software engineering researchers have examined and are examining the manner in which "experts" analyze program specifications and develop computer programs. This suggests that AIS researchers examine the existence of expertise in accounting information systems design and implementation. Based on the experience with programmer expertise it is likely that user requests for certain types of nonstandard systems may have a substantial impact on the development time and cost of accounting systems.

The existence of expertise by accounting information systems designers and developers has received little research attention. O'Leary and Munakata [1988] developed a prototype expert system that incorporated some accounting information systems design heuristics. Their system performs a small portion of the design process for accounting information systems.

Accounting information systems researchers also could be concerned with the impact of the structured methodologies used to design and develop accounting information systems. First, they may be concerned with customizing those methods to meet the needs of the accounting environment. Are there unique design and development methodologies (e.g., Mathur [1987]) for accounting information systems? Second, does the use of these methodologies make a measurable difference in the design or development of accounting systems?

Accounting information systems practitioners likely are very interested in forecasting the cost of the development of such accounting systems. Generally, management consulting departments of accounting firms estimate the hours and the rates of the personnel doing the work. Then they multiply the one figure times the other to get an estimate of the cost of the system. There is little in the literature going beyond this basic model indicating how to estimate such costs or implementation hours, or what factors may impact the cost or implementation hours. Yet, those same firms probably have substantial data that would facilitate such a study.

4. COMPUTER SECURITY AND PRIVACY

Software engineers are necessarily concerned with the security and privacy of software because without security the software can be easily altered or the software could be used without permission. A special issue of *IEEE Transactions on Software Engineering* was devoted to this specific set of issues in February 1987.

Some of the research by software engineers on security derives from the need to develop tools to meet the specific security needs of users. One of the primary sources of such constraints has been issued by the United States Department of Defense, as summarized in their *Trusted Computer Systems Evaluation Criteria*. Denning et al. [1987] provide one such investigation.

Other research in security involves the integration of expert systems technology into security and privacy. Denning [1987] describes a model for an expert system that is designed to detect a wide range of security violations, ranging from outsiders attempting to penetrate the system to insiders abusing the system. The model is based on the notion that exploitation of the system involves abnormal use of the system. Thus, the model includes knowledge about the expected use of the system. The model is not designed to eliminate more traditional security methods, but merely provides an extra layer of protection that can be used to enhance the security of the system. Similarly, Millen et al. [1987] developed a Prolog program designed to search for security vulnerabilities.

Recent technological developments have lead to an increase in the extent of use of distributed and other types of advanced systems, thus prompting research on security and privacy in those environments. Nesset [1987] examines and extends recent work on distributed system security requirements. Jain [1987] provides a general model of distributed systems.

Interorganization computer networks support person-to-person communication over media such as electronic mail. The security of interorganization networks is

easily ignored, yet the damage can be high. Estrin [1987] develops a conceptual model for implementing usage controls in such interorganization computer networks.

4.1 Implications for Accounting Information Systems

There are a number of implications for research in AIS of the software engineering work on security and privacy. First, expert systems may be used to aid in the security of accounting information system databases and software. However, little has been done using accounting expert systems for such security problems as those investigated by Denning [1987].

Second, in order for accounting information systems to exploit distributed processing, accounting information systems researchers need to address the security and control implications of technological developments such as distributed systems.

Third, to the extent that information about accounting transactions is communicated over computer networks, accounting information systems researchers need to be concerned about privacy and security in interorganizational computer networks.

5. SOFTWARE TESTING

A major portion of the total effort required in developing commercial software is incurred in software testing. Software testing is the process of examining software to determine the existence of errors and then finding those errors. Software testing has been discussed extensively by a number of researchers including Myers [1979], Adrion et al. [1982], Shooman [1983], and Beizer [1984].

The extent of the testing is, in part, a function of the necessary reliability. Generally, the higher the required level of reliability, the more extensive the testing process. In addition, testing is guided partially by the same concerns faced in the analysis of complexity. For example, program testing employs "module testing" (Zelkowitz [1978]). Further, the amount of testing also is a function of the amount of complexity in the application. Highly complex applications will require more testing than less complex applications.

Zelkowitz [1978] divides testing into module testing (individual testing of each module), integration testing (testing of groups of modules) and systems testing (testing of the completed system). Initially, software testing is aimed at identifying the existence of individual errors in software and of types of error-prone software. A number of ways have been developed for identifying errors (e.g., Beizer [1984, Chapter 3]), including path testing, transaction flow testing, input validation, and other approaches. A path through a program is any executable sequence of instructions through that routine. The purpose of path testing is to test enough of the different paths to demonstrate that the routine does what it is supposed to do. A transaction is a unit of work that consists of a sequence of steps or tasks, for example, accept input, validate input, search file, and record transaction in log. Transaction testing is testing every link, every decision outcome and every process. Input validation includes the use of test data to test various aspects of the system. Velasco [1987] provides a recent analysis of test data selection.

An alternative approach is to try to identify characteristics of that software likely to have errors. Shen et al. [1985] found that the amount of data and the structural complexity of the software were likely to be the factors most useful in identifying error-prone software.

Equally important in program testing is ascertaining the cause and source of programming errors. Discussion with a Vice President of system development indicated that most errors in software developed by that firm likely were committed by programmers who had been with the firm for less than a year and who were in tightly time-constrained situations. Such an analysis of causation of programming errors can be used to guide the use of testing resources.

Once a software error has been discovered, then the programmer must find the location of the error or bug. Accordingly, debugging computer programs is a critical issue to software engineer studies. Researchers, (Johnson and Soloway [1985]), have investigated the existence

and the content of debugging expertise. Those researchers then embedded that program expertise into a system designed to debug other programs. In addition, specific domains can make substantial differences in the analysis and testing of software. The need for domain specific theories of program testing is illustrated by Howden's [1982] discussion of validating scientific programs.

5.1 Software Testing for Accounting Information Systems

Accounting information systems designers and implementers face a problem similar to the one faced by software engineers. When an accounting system is being implemented there is generally a test of individual components and then a system-level test period, where the existing system and the new system are run in parallel or at least partially in parallel. The purpose of running in parallel is, in part, to determine the existence of errors. It is in these testing situations that the software engineering processes discussed above may prove useful.

However, there has been little work by accounting researchers to investigate the problem of testing an accounting information system. For example, there are few ways, other than running in parallel, that have been suggested to provide the tester with insight into the existence of errors. Software engineering research suggests that accounting researchers examine the characteristics that lead to error-prone accounting information systems and the causes of errors in accounting information systems. For example, does the complexity of the system contribute to the possibility of errors and are accounting information systems errors like computer programming errors in that they are often caused by people new to the job? In any event, if we assume that there will be errors, to what extent can heuristics be used to discover and investigate these errors? Additionally, there is always the question of how much to test (how many tests are necessary to validate a system?). This is all part of the fundamental problem that there is no underlying framework or theory for testing accounting information systems (Weber [1987]).

There has been, however, progress in development of a software engineering-based approach to testing accounting and auditing expert systems. O'Leary [1986] developed a framework for validating expert systems based in part on the software engineering framework developed by Shooman [1983] and others. That software engineering framework was extended by taking into account the specificity of the accounting and auditing domain and the symbolic nature of knowledge processed by expert systems.

Finally, accounting information systems researchers may wish to examine the substantial literature on program testing for potential use in auditing and testing accounting information systems.

6. PROTOTYPING

In software engineering, prototyping is a process of creating a model of the software that is to be developed. Information gained in that model development is then fed back for future model development. As noted by Brooks [1975], prototyping is a necessary part of the software engineering process.

> In most projects, the first system built is barely usable. It may be too slow, too big, too awkward in use, or all three. There is no alternative but to start again, smarting, but smarter, and build a redesigned version in which these problems are solved....When a new system concept or technology is used, one has to build a system to throw away, for even the best planning is not so omniscient as to get it right the first time.

In practice, prototyping is an iterative process that consists of gathering requirements, designing quickly, building the prototype, evaluating and refining the prototype and engineering the product (Pressman [1987]). Since prototyping is offered as an alternative approach to more structured processes such as requirements specification, software engineers have investigated the differences, if any, between the two approaches to developing software.

In one experiment of seven software teams, Boehm et al. [1984] found that there were a number of differences in the resulting software products. Prototyping yielded products that were approximately the same in performance, with about 40 percent less code and 45 percent less effort. The prototype products rated somewhat lower on functionality and robustness, but higher on ease of use and ease of learning. Software built using requirements specification produced more coherent designs and was easier to integrate. In addition, Alavi [1984] found that prototyping facilitates communication between users and designers. However, she found that prototyping made managing and controlling the design process more difficult.

However, prototyping is not just used as a development methodology. In software engineering, prototyping often is used to demonstrate a "proof of concept." Researcher claims for various problem solving approaches must be substantiated by developing a prototype of the system. For example, the GANDALF project (Notkin [1985]) involved the development of a number of prototypes in the analysis of the automated generation of software development environments. Wasserman et al. [1986] developed a prototype to test USE (*U*ser *S*oftware *E*ngineering), an interactive system that provides conversational access to data.

In addition, there are other uses of prototypes. O'Leary [1988] has investigated the use of prototyping as a research tool. He argues that prototypes can be used to both test theories and generate theories. The inability of prototypes to explain anomalies can lead to the generation of new theories. O'Leary [1988] discusses a theory of prototyping based on traditional research methods and some of the experimental controls that can be integrated into the research prototyping process. An extensive list of references on prototyping is given in Jenkins and Fellers [1986].

6.1 Prototyping in Accounting Information Systems

Prototyping in accounting information systems has been used as a research tool focusing on understanding particular decision processes using expert systems (e.g., Steinbart [1986] and others). In this vein, accounting information systems researchers may wish to test theories of AIS design and development using a prototype.

Prototyping also can be used to test whether new techniques can be used or even developed. For example, it can prove useful to prototype various types of new computer system controls to see if they work as anticipated.

From a systems design, development and implementation perspective, it is not clear if prototyping accounting information systems is a better methodology than more structured approaches, in contrast to previous software engineering studies. Many accounting information system needs are well-structured and there are definite audit and security requirements associated with accounting data that may limit the effectiveness of prototyping. Alternatively, there are other less structured accounting systems problems, for example, cash management decision making, for which prototyping may be an effective systems design and development methodology. However, it is not clear in what environments or at what level of complexity or reliability of the software we would expect prototyping to offer a beneficial alternative to more structured requirements analysis. Accounting information systems researchers have the opportunity to study these issues using a series of models ranging from personal computer general ledger packages to complex expert systems.

7. RESEARCH DATABASES

It can be argued that many of the research questions and the development of specified research tools used in a given discipline derive from the available databases used in those disciplines. If there are central databases available to researchers then well-defined means of studying those databases can be developed. Thus, a consistent set of methodologies can be established. For example, market-based research in finance and financial accounting

uses an established set of tools to investigate the large databases of stock price behavior.

Researchers in software engineering have accumulated a number of different databases. Conte et al. [1986, pp. 177-180] are part of a group that collects and maintains what they refer to as the Software Metrics Data Collection. That database includes data describing commercial software developed in a number of industrial environments and using controlled experiments for the analysis of software metrics. It includes data from Boehm [1981], which is a large database of statistics based on commercially developed products, in which the identity of the product is not given. The database also includes data on software products developed at the NASA Goddard Space Center.

Other types of databases have been accumulated. Jeffery and Lawrence [1985] studied a database of survey data. They used a previously established survey database from 1976 and another database from 1980 in a longitudinal study of programmer productivity. Additionally, Gurbaxani and Mendelson [1987] studied published data supplied by International Data Corporation on data processing budgets.

7.1 Databases in Accounting Information Systems Research

Unfortunately, there are few publicly available databases for research in accounting information systems. As a result, accounting information systems researchers have employed commercially available systems or previously published systems on only a limited basis. For example, Weber [1986] examined a number of commercially available software packages (order entry modules) in a study of McCarthy's [1982] database model. O'Leary [1987] investigated a validation framework for accounting and auditing expert systems using a number of previously published accounting expert system prototypes.

Potentially, accounting information systems researchers have available large databases, with rich descriptions of AIS design and development efforts. The management consulting departments of public accounting firms have developed and implemented vast numbers of accounting information systems. Typically, well documented work papers support such systems. Since work papers summarize information in a structured manner, they could be used to develop a database that could be used to study a broad base of accounting information systems issues.

In addition, corporations implementing accounting information systems and/or accounting firms involved in the ongoing development of accounting information systems offer the opportunity for researchers to study a number of issues beyond the scope of the work papers. Accounting researchers can function as participant observers or just observers in the development of accounting information systems.

Further, the accounting information systems journals also offer the opportunity to develop databases on accounting information systems. The reporting and publishing of accounting information systems designs and developments provides a potential database for researchers studying the behavior of such systems. Recent articles discussing the design and implementation of particular systems in *The Journal of Information Systems* are consistent with this database need. However, in order to ensure a quality database, it may be helpful to delineate a set of requirements related to particular research issues that each research paper describing a system would discuss.

8. SOFTWARE ENGINEERING RESOURCES

There are a number of good introductions to software engineering, including Pressman [1987], Shooman [1983] and Zelkowitz [1978]. There are a number of journals that publish software engineering research, including *IEEE Transactions on Software Engineering, IEEE Transactions on Reliability, IEEE Software, Computer, Communications of the ACM, ACM Software Notes, IBM Systems Journal, Journal of Systems and Software,* and the *AT&T Technical Journal.* In addition, the new journals, *Information Systems Research*

and *OR/CS* (Operations Research and Computer Science) may include software engineering issues of concern to the management and decision sciences.

There are also technical meetings in software engineering. IEEE and ACM publish the papers given at one of these meetings in *Proceedings of the International Conference on Software Engineering.*

Software Engineering Standards contains standards for software requirements specification, software test documentation, software configuration management and software quality assurance.

At Carnegie Mellon University, the United States Government has established a Software Engineering Institute. This organization has been charged with the responsibility of developing a software engineering monograph series.

Finally, Glass [1986] is a paper that discusses some of the limitations of software engineering and artificial intelligence with a great deal of insight.

9. CONCLUSION

If accounting information systems researchers are to differentiate themselves from management information systems researchers, they need to exploit and explore the specificity of the accounting information systems context. Software engineering can provide some guidelines as to methods, databases and issues. Further, accounting information systems researchers can provide input into software engineering by researching the impact of the specificity of a single domain on software requirements, design and development.

Accounting information systems researchers have the opportunity to examine a number of issues including the

- impact of system complexity and reliability on, e.g., cost and development time of the accounting information system,
- judgment and expertise used in the design, development and implementation of accounting expert systems,
- factors that impact the cost of the design and development of accounting information systems,
- impact of the use of "structured" development and design methodologies on accounting information system design,
- development of new means of securing accounting information systems,
- means of testing accounting information systems and a theory of testing that takes into account the specificity of the accounting context and
- use of prototyping in the analysis, design and development of accounting information systems.

Research in software engineering can provide a basis of those investigations. However, in order to pursue these research issues to the extent allowed by software engineering-based approaches, databases on accounting information systems need to be developed and prototyping as a research methodology needs to be accepted.

REFERENCES

Adrion, W., M. Branstad, and J. Cherniavsky, "Validation, Verification, and Testing of Computer Software," *Computing Surveys* (June 1982), pp. 159-192.

Alavi, M., "An Assessment of the Prototyping Approach to Information Systems Development," *Communications of the ACM* (June 1984), Volume 27, Number 6, pp. 556-563.

Amer, T., A. Bailey, and P. De, "A Review of the Computer Information Systems Research Related to Accounting and Auditing," *The Journal of Information Systems*, Vol. 2, No. 1 (Fall 1987), pp. 3-28.

Beizer, B., *Software System Testing and Quality Assurance* (New York: Nostrand Reinhold Company, 1984).

Boehm, B., *Software Engineering Economics* (Englewood Cliffs, New Jersey: 1981).

———, B., T. Gray, and T. Seewaldt, "Prototyping Versus Specifying: A Multiproject Experiment," *IEEE Transactions on Software Engineering*, Vol. SE-10, No. 3 (May 1984), pp. 290-303.

Bodnar, G., "Reliability Modeling of Internal Control Systems," *The Accounting Review* (October 1975), pp. 747-757.

Brooks, F., *The Mythical Man-Month* (Reading, Massachusetts: Addison-Wesley, 1975).

Card, D., F. McGarry, and G. Page, "Evaluating Software Engineering Technologies," *IEEE Transactions on Software Engineering*, Vol. SE-13, No. 7 (July 1987), pp. 845-856.

Carver, D. and D. Simmons, "The Impact of Programming Methodology on Program Complexity," *The Journal of Systems and Software*, Vol. 5 (1985), pp. 279-289.

Conte, S., H. Dunsmore, and V. Shen, *Software Engineering Metrics and Models* (Menlo Park: Benjamin/Cummings Publishing, 1986).

Curtis, B., "Substantiating Programmer Variability," *Proceedings of the IEEE Conference*, Vol. 69, No. 7 (July 1981), p. 846-850.

Cushing, B., "A Mathematical Approach to the Analysis and Design of Internal Control Systems," *Accounting Review* (January 1974), pp. 24-41.

Denning, D., "An Intrusion-Detection Model," *IEEE Transactions on Software Engineering*, Vol. SE-13, No. 2 (February 1987), pp. 222-232.

———, S. Akl, M. Heckman, T. Lunt, M. Morgenstern, P. Neuman, and R. Schell, "Views for Multilevel Database Security," *IEEE Transactions on Software Engineering*, Vol. SE-13, No. 2 (February 1987), pp. 129-139.

Dijkstra, E., *A Discipline of Programming* (Englewood Cliffs, New Jersey: Prentice-Hall, 1976).

Draper, S. and D. Norman, "Software Engineering for User Interfaces," *IEEE Transactions on Software Engineering*, Vol. SE-14, No. 3 (March 1987), pp. 252-261.

Dunn, R. and R. Ullman, *Quality Assurance for Computer Software* (New York: McGraw-Hill, 1982).

Estrin, D., "Controls for Interorganization Networks," *IEEE Transactions on Software Engineering*, Vol. SE-13, No. 2 (February 1987), pp. 249-261.

Fickas, S., S. Collins, and S. Oliver, "Problem Acquisition in Software Analysis: A Preliminary Study," Unpublished Working Paper, Department of Computer and Information Science, University of Oregon (August 31, 1987).

Gaffney, J., "The Impact on Software Development Costs of Using HOL's, *IEEE Transactions on Software Engineering*, Vol. SE-12, No. 3 (March 1986), pp. 496-500.

Gal, G. and P. Steinbart, "Artificial Intelligence and Research in Accounting Information Systems: Opportunities and Issues," *The Journal of Information Systems*, Volume Two, Number 1 (Fall 1987), pp. 54-62.

Glass, R., "Editor's Corner—'Dangerous and Misleading': A Look at Software Research via the Parnas Papers," *The Journal of Systems and Software*, Vol. 3 (1986), pp. 217-218.

Goldberg, A., "Knowledge-Based Programming: A Survey of Program Design and Construction Techniques," *IEEE Transactions on Software Engineering*, Vol. SE-12, No. 7 (July 1986), pp. 752-768.

Gorry, G. and M. Scott Morton, "A Framework for Management Information Systems," *Sloan Management Review*, Vol. 13, No. 1 (Fall 1971), pp. 55-70.

Gurbaxani, V., and H. Mendelson, "Software and Hardware in Data Processing Budgets," *IEEE Transactions on Software Engineering*, Vol. SE-13, No. 9 (September 1987), pp. 1010-1017.

Habermann, A., "Automatic Deletion of Obsolete Information," *The Journal of Systems and Software*, Vol. 5 (1985), pp. 145-154.

Howden, W., "Validation of Scientific Programs," *Computing Surveys*, Vol. 14, No. 2 (June 1982), pp. 193-212.

Jain, H., "A Comprehensive Model for the Design of Distributed Computer Systems," *IEEE Transactions on Software Engineering*, Vol. SE-13, No. 10 (October 1987).

Jeffery, D. and M. Lawerence, "Managing Programming Productivity," *Journal of Systems and Software*, Vol. 15 (1985), pp. 49-58.

Jenkins, A. and J. Fellers, "An Annotated Bibliography on Prototyping," Unpublished Working Paper, Graduate School of Business, Indiana University (June 1986).

Johnson, W., and E. Soloway, "Proust: Knowledge-Based Program Understanding," *IEEE Transactions on Software Engineering*, Vol. SE-11, No. 3 (March 1985), pp. 267-275.

Jones, T., "Measuring Programming Quality and Productivity," *IBM Systems Journal*, Vol. 17, No. 1 (1978).

———, *Programming Productivity* (New York: McGraw-Hill, 1986).

Kemerer, C., "An Empirical Validation of Software Cost Estimation Models," *Communications of the ACM*, Vol. 30, No. 5 (May 1987), pp. 416-429.

Kitchenham, B., and N. Taylor, "Software Project Development Cost Estimation," *The Journal of Systems and Software*, Vol. 5 (1985), pp. 267-278.

Laughery, K., and K. Laughery, "Human Factors in Software Engineering: A Review of the Literature," *The Journal of Systems and Software*, Vol. 5 (1985), pp. 3-14.

Mathur, R., "Methodology for Business System Development," *IEEE Transactions on Software Engineering*, Vol. SE-11, No. 5 (May 1987), pp. 593-601.

McCarthy, W., "The REA Accounting Model: A Generalized Framework for Accounting Systems in a Shared Data Environment," *The Accounting Review* (July 1982), pp. 554-578.

Millen, J., S. Clark, and S. Freedman, "The Interrogator: Protocol Security Analysis," *IEEE Transactions on Software Engineering*, Vol. SE-13, No. 2 (February 1987), pp. 274-288.

Myers, G., *Software Reliability* (New York: John Wiley & Sons: 1976).

———, *Composite/Structured Design* (New York: Van Nostrand Reinhold, 1978).

———, *The Art of Software Testing* (New York: John Wiley & Sons: 1979).

Nessett, D., "Factors Affecting Distributed System Security," *IEEE Transactions on Software Engineering*, Vol. SE-13, No. 2 (February 1987), pp. 233-248.

Notkin, D., "The GANDALF Project," *The Journal of Systems and Software*, Vol. 5 (1985), pp. 91-105.

O'Leary, D., "Validation of Business Expert Systems: A Software Engineering Approach," Unpublished Paper Presented at the *USC Audit Judgment Symposium* (February 1986). (For a revised version see O'Leary [1987]).

———, "Validation of Expert Systems: With Applications to Auditing and Accounting Expert Systems," *Decision Sciences*, Vol. 18, No. 3 (1987), pp. 468-486.

———, "Prototyping of Expert Systems as a Research Methodology," *Applied Expert Systems*, edited by E. Turban and P. Watkins (North-Holland, The Netherlands: Forthcoming, 1988).

———, and T. Munataka, "An Accounting Prototype Expert System," in *Artificial Intelligence in Accounting and Auditing: The Use of Expert Systems*, edited by M. Vasarhelyi, Markus Wiener (New York: Forthcoming, 1988).

Pressman, R., *Software Engineering* (New York: McGraw-Hill, 1987).

Reneau, J. and S. Grabski, "A Review of Research in Computer-Human Interaction and Individual Differences within a Model for Research in Accounting Information Systems," *The Journal of Information Systems* (Fall 1987), pp. 33-53.

Ross, S., "Statistical Estimation of Software Reliability," *IEEE Transactions on Software Engineering*, Vol. SE-11, No. 5 (May 1985), pp. 479-482.

Scheidewind, N., "The State of Software Maintenance," *IEEE Transactions on Software Engineering*, Vol. SE-13, No. 3 (March 1987), pp. 303-310.

Shen, V., T. Yu, S. Thebaut, and L. Paulsen, "Identifying Error-Prone Software—An Empirical Study,"*IEEE Transactions on Software Engineering*, Vol. SE-11, No. 4 (April 1985), pp. 316-317.

Shooman, M., *Software Engineering* (New York: McGraw-Hill, 1983).

Simon, H., "Whether Software Engineering Needs to be Artificially Intelligent," *IEEE Transactions on Software Engineering*, Vol. SE-12, No. 7 (July 1986), pp. 726-732.

Software International Corporation, "Setting Standards in Mainframe Applications Software," *The Journal of Systems and Software*, Vol. 6 (1986), pp. 295-305.

Soloway, E., and K. Ehrlich, "Empirical Studies of Programming Knowledge," *IEEE Transactions on Software Engineering*, Vol. SE-10 (September 1984), pp. 595-609.

Steinbart, P., "Materiality: A Case Study Using Expert Systems," *The Accounting Review* (January 1987), pp. 97-116.

Tripathi, S., Y. Huang, and S. Jajodia, "Local Area Networks: Software and Related Issues," *IEEE Transactions on Software Engineering*, Vol. SE-13, No. 8 (August 1987), pp. 872-883.

Velasco, F., "A Method for Test Data Selection," *The Journal of Systems and Software*, Vol. 7 (1987), pp. 89-97.

Vessey, I., and R. Weber, "Research on Structured Programming: An Empiricist's Evaluation," *IEEE Transactions on Software Engineering*, Vol. SE-10, No. 4 (July 1984), pp. 397-407.

Wasserman, A., P. Pircher, and D. Shewmake, "Building Reliable Interactive Information Systems," *IEEE Transactions on Software Engineering*, Vol. SE-12, No. 1 (January 1986), pp. 147-156.

Weber, R., "Database Models Research in Accounting: An Evaluation of Wholesale Distribution Software," *The Accounting Review* (July 1986), pp. 498-518.

———, "Toward a Theory of Artifacts: A Paradigmatic Base for Information Systems Research," *The Journal of Information Systems*, Volume One, Number Two (Spring 1987), pp. 3-19.

Yau, S. and J. Tsai, "A Survey of Software Design Techniques," *IEEE Transactions on Software Engineering*, Vol. SE-12, No. 6 (1986), pp. 713-721.

Zelkowitz, M., "Perspectives on Software Engineering," *Computing Surveys* (June 1978), pp. 197-216.