1991

An investigation of the effect of a reverse engineering tool on the software maintenance effort

Leonie Menco Fernandes

Rochester Institute of Technology

Computer Science Department

AN INVESTIGATION OF THE EFFECT OF A REVERSE ENGINEERING TOOL ON THE SOFTWARE MAINTENANCE EFFORT

by

Leonie Menco Fernandes

A thesis, submitted to
The Faculty of the School of Computer Science and Information Technology, in partial fulfillment of the requirements for the degree of Master of Science in Computer Science.

Approved by:

_____
Professor Alan Kaminsky

_____
Professor William Stratton

_____
Professor Peter Anderson

_____
Professor Daniel Lawrence

January 28, 1991

3. Title of Thesis An Investigation of the Effect of a Reverse Engineering Tool on the Software Maintenance Effort

I Leonie Menco Fernandes hereby **deny** permission to the Wallace Memorial Library of RIT to reproduce my thesis in whole or in part.

Date 2/25/91

rev. 1/89

ABSTRACT

In order to determine the effectiveness and efficiency of software engineering aids in software program maintenance, productivity improvements -- as measured by the time to task completion -- for a representative sample of Software Engineers were assessed, with and without the use of a commercially available CASE tool. These improvements were also measured in relation to the level of experience of the Software Engineer and the complexity level of the program maintained.

Results show that the CASE tool appeared to have a significant impact on the productivity of the Software Engineers included in the study. The effect of the tool in terms of productivity, was positive for both experienced and inexperienced Software Engineers. This positive effect was apparent regardless of the complexity of the program.

Acknowledgments

This study and document were prepared with the assistance of others, who spent their time to ensure that it was a success. All their efforts were greatly appreciated.

The members of my committee offered me all the assistance I needed to complete this study. My deepest thanks are extended to Dr. William Stratton for spending many sessions with me throughout the thesis development. He answered all my questions and always made sure I stayed on the right path. The sessions I spent with Bill Stratton were intellectually stimulating and enjoyable. I learned far more during the course of this study than I originally expected to. Much of that knowledge and understanding were developed during my conversations with Dr. Stratton. A very special thanks is extended to Alan Kaminsky, who was the first to take an interest in the topic for my thesis. Although my original proposal was slightly modified, Alan took an interest and gave me the chance to pursue a topic I wanted to learn more about. He provided guidance and assistance to me through the entire study. Thanks is also extended to Dr. Daniel Lawrence who took the time to review my work throughout the thesis development. Although I never actually met Dan until my thesis defense, his efforts behind the scenes were critical in producing an accurate study and thesis document. I also extend thanks to Dr. Peter Anderson who guided me throughout all my course work at RIT. It was fitting for me to begin my RIT education and end my RIT education with the assistance of Peter Anderson.

The support of my parents and my husband were critical to the completion of my school work and my thesis. Their understanding, support, and patience were needed and greatly appreciated. They provided me with the desire and the confidence to pursue a dream -- the completion of a Master of Science Degree.

TABLE OF CONTENTS

CHAPTER I

STATEMENT OF THE PROBLEM

Introduction

The need for software services in the United States is continually increasing. All types and sizes of corporations and private organizations are realizing that automation is the key to success in a competitive market. Automation introduces high quality and consistency, as well as faster response to a customer's needs. For years, the software services industry has been responding to the needs of companies and individuals by developing software to automate the business of its customers.

The software services industry has turned its attention inward by applying automation techniques to its own business. It is automating the process used by Software Engineers to develop and maintain programs and systems. The development of software tools to assist Software Engineers in their work is necessary in order to produce high quality, complex systems in a short amount of time. These tools are referred to as CASE (Computer Aided Software Engineering) tools, and they are being developed by the software services industry for use by Software Engineers.

These CASE tools assist Software Engineers with the development of new systems by automating much of the Systems Development Life Cycle. The tools enforce proper design techniques and can automatically generate system code from familiar design constructs. The generated system code is accurate and logically correct at its

inception. Maintenance of these systems is greatly simplified by allowing Software Engineers to manipulate a design construct and have the CASE tool regenerate the system accordingly.

Software services companies are using and developing these tools in order to maintain a competitive edge over other software companies, both in the United States and around the world. Several companies have experienced a large productivity increase by their Software Engineers by using CASE tools (Baxter, 1987; Ameritech, 1987; Hartford Insurance, 1987). As a result, new systems and programs can be developed, enhanced and maintained very effectively.

CASE tools provide significant solutions to the software services industry by assisting in the reduction of the backlog of new system requests. However, until recently, these tools have not been able to address the backlog of system enhancements and maintenance tasks needed for existing systems. These existing systems are maintained through manual techniques, which allows room for error, as well as damage to the existing program structure if the system changes are not made carefully. The manual maintenance and support of these older systems is consuming the software services industry resources. Typically, 75% of the resources in software services companies is used to maintain existing systems (Digital Consulting, Inc., 1988; Datapro Research, 1984; Datamation, 1984). This percentage is rising as the amount of code across the world is steadily increasing and becoming more complex.

The software services industry is looking to CASE tools to assist with the maintenance effort of these older systems. Several different CASE tools have been developed to address the maintenance

effort. Uncertainty exists in the industry that these tools can reduce the amount of resources required to maintain systems (Zvegintzov, 1989). The alternative of the software services companies is to completely replace these older systems with new ones that are developed using automated design and system generation tools. This effort, referred to as re-engineering, is a very costly option for most software services organizations.

Problem

Tackling the maintenance problem that exists for most large corporations is difficult (Digital Consulting, Inc., 1988). Many older existing systems are needed to run the most basic functions for the software services customer, including financial, personnel, sales, manufacturing and engineering areas. For those systems that absorb large amounts of resources for maintenance, there are two choices: either re-engineer the entire system or look for ways to assist the Software Engineer with the maintenance effort.

The re-engineering alternative offers advantages and disadvantages (Bush, 1988). The primary advantage of re-engineering a system is that it can be developed, maintained and enhanced with the use of CASE development tools. The maintenance effort, after re-engineering, is significantly reduced (Digital Consulting, Inc., 1988). The primary disadvantages are the extensiveness of the resources required to complete such a task and the difficulty in assuring that all system functions, both documented and hidden, are replicated in a new system. Most older existing systems that have

been modified extensively, tend to have hidden functions that a customer is accustomed to using but may not necessarily be identified in a re-engineering effort. Another major disadvantage of the re-engineering alternative is that although there are CASE tools to help the forward development process of system generation from design specifications, there are no CASE tools that can automate the reverse engineering process of taking existing code and creating the design specification needed for the forward development.

The use of CASE tools to assist the Software Engineers with the maintenance effort is receiving a significant amount of attention from the software services industry. The attractiveness of this alternative is enhanced due to the fact that the technology now exists to immediately apply these tools within the software services industry. The CASE tools that address this scenario are referred to collectively as reverse engineering tools. The disadvantage of these tools is that they currently can address only existing software problems within the operation arena (see Diagram 1.1). Since these tools are relatively new to the CASE market, there is currently no statistical evidence to support the hypothesis that they can reduce the resources required to maintain existing systems. The software suppliers of these tools claim very high productivity gains based on the feedback from their software services industry customers (Digital Consulting, Inc., 1988). However, an informal review of this feedback suggests that it is primarily an estimate of the productivity savings from the customer. No substantial evaluation of actual resource savings or productivity gains has been done to date.

The technology necessary to perform the full re-engineering

cycle, including both the reverse and forward generation process, remains several years away according to industry announcements (Language Technology, 1988; Bachman, 1988; ViaSoft, 1988). Until that time, the use of maintenance-assistance tools may alleviate the large amounts of software services resources required to keep the one billion lines of existing code running correctly (Jones, 1987). Without the benefit of substantial investigation into the effects of these types of reverse engineering CASE tools on the maintenance effort, no one knows whether any productivity gain can be achieved in order to expedite the reduction of maintenance backlogs and resources.

Yet another alternative for the software services industry is to do nothing at this point in time. The trouble with this alternative is that according to software services industry projections, by the mid-1990's, the time of all available Software Engineers will be devoted to the maintenance of existing systems (Gartner Group, 1988; Bachman, 1988). The Air Force is predicting that by 1995, 25% of all males between the ages of 18 and 25 will be needed to support the maintenance of its existing software (Digital Consulting, Inc., 1988).

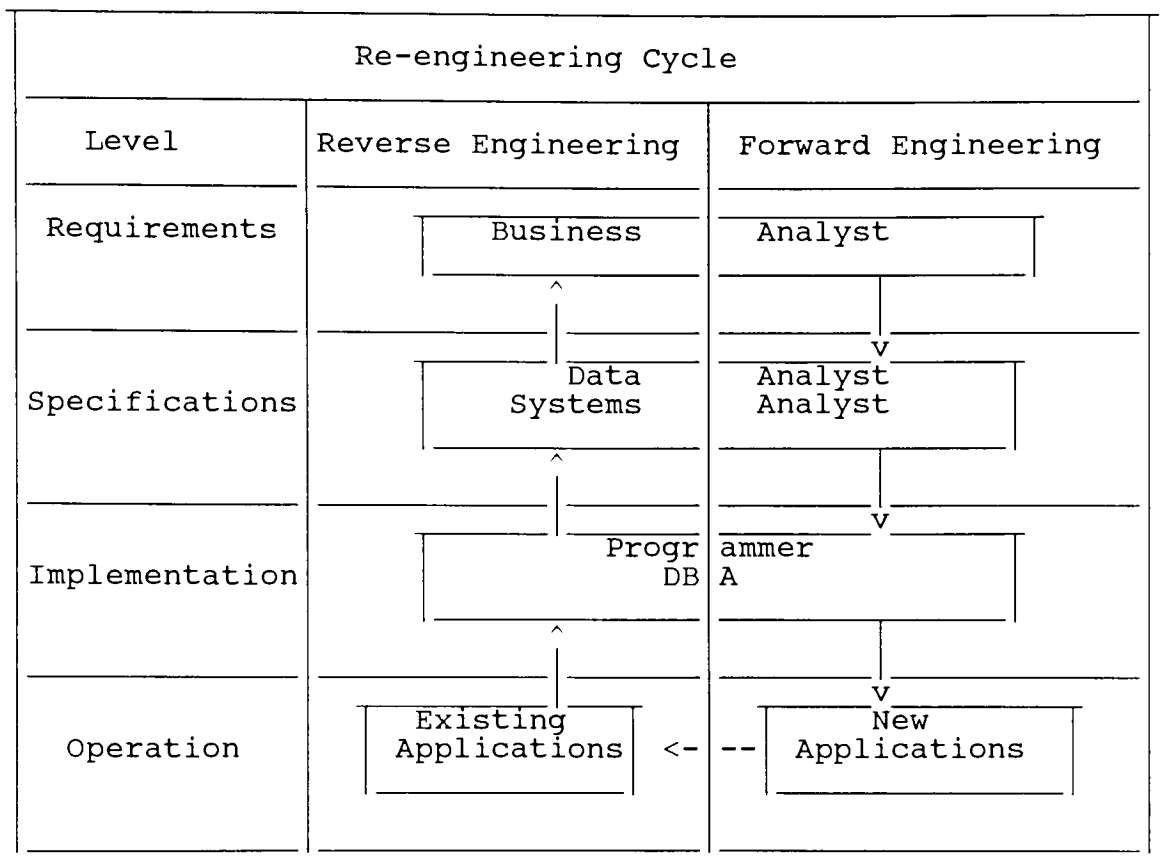

Diagram 1.1
Re-engineering Cycle

Source: Bachman Information Systems, Cambridge, MA.

Purpose of the Study

The use of re-engineering tools in the software services industry is expected to result in a dramatic increase in Software Engineer productivity. However, there is no substantial evidence to support such a conjecture. This study examines the software maintenance environment within a large software services company and investigates the effect of reverse engineering CASE tools on the productivity of Software Engineers.

Background

Software vendors that sell reverse engineering CASE tools to the software services industry claim large productivity gains in the maintenance phase of a systems life cycle. An informal review of these claims, though, reveals that most of them are based on brief interviews and on studies conducted with the reverse engineering CASE tools being used only on a trial basis (Baxter, 1988; Ameritech Services, 1988; Phillips Petroleum, 1988; Hartford Insurance Group, 1988). The users of these tools insist that a productivity increase has been achieved, yet no statistical evidence has been produced to verify these increases.

The evidence that has been produced by reverse engineering CASE tool investigations is of a comparison form. Actual maintenance effort using the tool was captured quantitatively, and the productivity gain was demonstrated by comparing the actual effort with an original estimated effort (Baxter, 1988). The validity of

the productivity gain measurement is strictly related to the validity of the original estimate of productivity. The estimating process is an art among Software Engineers, and the accuracy of their estimates is increased only on the basis of previous experiences. Unless evidence exists to support the accuracy of the original estimate, questions naturally arise regarding the validity of the productivity gains that are reported.

Ergonomics is also studied while investigating the use of reverse engineering CASE tools. Software Engineers who use the tool are asked to answer a questionnaire which includes questions related to ease of use, documentation, advantages and disadvantages of using the tool, and the estimated percent of productivity gain. These questionnaires tend to be highly subjective, based upon a Software Engineer's background and experience. Although the answers to these questions are helpful in studying the effects of reverse engineering CASE tools within the software services industry, they do not substantiate a productivity gain within an organization of maintenance Software Engineers (Baxter, 1988).

In summary, based on past investigations of reverse engineering CASE tools, all the information that is collected regarding past maintenance performance is subjective and not substantiated. In order to substantiate this information, time and resources must be given to the use of a reverse engineering CASE tool, with any productivity gain demonstrated by way of a comparison of actual resources expended before and after its introduction.

Derivation of a Model

The purpose of a model featuring the reverse engineering CASE tool is to evaluate the overall effectiveness of one of these tools on the performance of Software Engineers and to determine whether or not the effects are consistent. To that end, a group of sixteen Software Engineers was assembled for the investigation. Then, one dependent variable, time to complete a maintenance task, was measured as affected by three independent variables, namely, (1) experience level of the Software Engineer, (2) complexity of the program, and (3) the use of the reverse engineering CASE tool (see Diagram 1.2). The tool investigated in this model is a reverse engineering CASE tool that is claimed to assist Software Engineers with the analysis, testing and implementation of program changes.

The ANOVA model for this study is as follows:

$$Y_{ijkl} = \mu + E_i + T_j + (ET)_{ij} + P_k + (EP)_{ik} + (TP)_{jk} + (ETP)_{ijk} + e_{ijkl}$$

where E = experience
T = tool versus non-tool
P = program complexity
and i = 1 to 2
j = 1 to 2
k = 1 to 2
l = 1 to 3

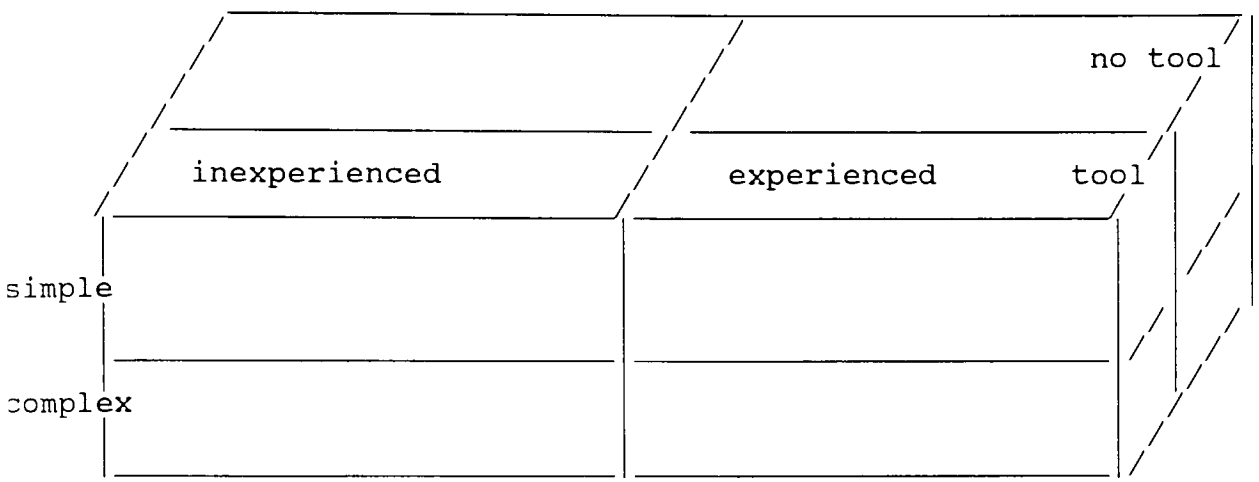

Diagram 1.2

The experience level of a Software Engineer was measured by the number of years that that individual had programmed in the software services industry. The complexity of the program was determined by McCabe's Cyclomatic Complexity measure (McCabe, 1976). The use of the reverse engineering CASE tool was indicated by a boolean numeral -- 1, if the tool was used; 0, if it was not used.

Research Questions

The model that features the reverse engineering CASE tool is based on a design for investigating the productivity gain from both experienced and inexperienced Software Engineers performing maintenance on complex and simple programs with and without the use of the tool. This model is the basis for an investigation of three relationships: (1) the relationship between reverse engineering CASE tools and productivity improvements in maintenance of a program, (2) the relationship between experience of a Software Engineer and the use of a reverse engineering CASE tool, and (3) the relationship between the complexity of a program and the use of the reverse engineering CASE tool.

The three research questions are:

Question 1. Does the use of the reverse engineering CASE tool reduce the amount of time required to maintain programs?

Does the reverse engineering CASE tool assist a Software Engineer with maintenance tasks? Can a Software Engineer fix or enhance a program in less time when using the

reverse engineering CASE tool to automate the maintenance activities than without it?

Question 2. Does the reverse engineering CASE tool equally equip experienced and inexperienced Software Engineers?

Does the reverse engineering CASE tool assist the experienced Software Engineer in maintaining a program to the same degree that it does an inexperienced Software Engineer? Does an inexperienced Software Engineer using the reverse engineering CASE tool have the same productivity as an experienced Software Engineer?

Question 3. Does the reverse engineering CASE tool work with equal effectiveness on simple and complex programs?

Does the reverse engineering CASE tool result in greater productivity gains for programs classified as "complex" than those classified as "simple"?

Definition of Terms

Analysis of Variance - A statistical method for comparing the average performances of two or more groups in order to determine whether or not statistically significant differences exist among these groups.

CASE (Computer Aided Software Engineering) - Software tools used to automate portions of the process used by Software Engineers to design, write and maintain programs.

Complex Program - A program that has a McCabe Cyclomatic Complexity measure of at least 10.

Design Constructs - Formalized and published diagramming symbols for graphically displaying the operation of a program, data, or system. (Examples: data flow diagram, entity relationship diagram, process model, flow chart.)

Design Techniques - Formalized and published techniques for graphically displaying the operation of a program or system.

Enhancements - Any change to a program or system that ultimately changes the functionality of that program or system.

Ergonomics - The ease of use and understandability of the software tool.

Forward Development Process - The process used by Software Engineers to write program code from design and analysis outputs.

Maintenance - Any change to a program or system that ultimately does not change the functionality of that program or system.

McCabe Cyclomatic Complexity - A measure of logical program complexity that is calculated using mathematical techniques.

Number of Years of Experience - Based on 40 hours/week, 5 days/week, working as a Software Engineer.

Re-engineering - The process used to perform both forward development and reverse engineering.

Re-engineering Tools - Software that assists the building or rebuilding of system functions with new or modified code.

Regenerate - Automatically generating program code from the system design after system alterations are made at the design level.

Resources - People who perform the Software Engineering work.

Reverse Engineering Process - The process used by Software Engineers to extract the system design from the program code (a subset of re-engineering).

Reverse Engineering Tools - Software that assists Software Engineers with systems and data analysis of existing programs (a subset of re-engineering tools).

Simple Programs - A program that has a McCabe Cyclomatic Complexity measure of less than 10.

Software Engineers - People who develop and maintain computer programs.

Software Services - Services and products delivered that are related to the development and maintenance of computer programs.

Software Services Industry - All companies, or components thereof, that develop, maintain and provide computer software.

Systems Development Life Cycle - The entire process used by Software Engineers that describes the development approach to systems from the inception of the idea, through analysis and design, through coding and testing, to maintenance.

System Generation Tools - Software tools that generate systems from ordinary design and analysis outputs.

Significance of the Study

As reverse engineering CASE tools enter the software market place, an evaluation method for determining their effectiveness, as well as an understanding of their benefit to different Software Engineers and programs, is needed by software services companies to make correct choices. This study examines whether or not the reverse engineering CASE tool can significantly assist a Software Engineer with maintenance programming tasks and establishes a method for future software evaluations.

Assumptions and Limitations

Some assumptions were made in defining the parameters of the analysis-of-variance model meant to address the research questions posed above. These assumptions relate to the measure of program complexity, the measure of Software Engineer experience, the reverse engineering CASE tool itself, and the set of maintenance tasks performed. Each assumption is discussed below.

> A program is classified as "complex" when the McCabe Cyclomatic Complexity measure is at least ten. A program that measures less than ten is classified as "simple".

The assumption that a program measuring ten or greater, based on McCabe's Cyclomatic Complexity measure, is "complex" was made on the basis of literature on this subject. Thomas Walsh (1979) reported on a study of the AEGIS Naval Weapon System which noted that programs with a complexity measure of ten or greater accounted for the majority of program errors. Schneidewind and Hoffman (1979) also observed a relationship between large (e.g., 5) McCabe complexity measures and program errors.

> A Software Engineer is considered to be experienced if he of she has at least six years of experience with programming in the software services industry. An inexperienced Software Engineer has three to five years of experience.

For purposes of this study, experience level was established by using the number of years that an individual has been programing. These assumptions about experience preclude a generalization of this study to all Software Engineers. The effect of the tool on performance, according to experience level, is a key measure as to the usefulness of the reverse engineering CASE tool.

> The selected reverse engineering CASE tool for this study is representative of this class of CASE tools.

There are many tools classified as reverse engineering CASE tools. They all address the same phase of the systems life cycle, and as such, it is reasonable to assume that the performance of any one of these CASE tools could be generalized to other CASE tools within this class. This assumption must be treated with caution, however, since research must also be conducted using other reverse engineering CASE tools in order to assure that any generalizations are valid.

The set of maintenance tasks performed by the Software Engineers are representative of the maintenance tasks performed throughout the software services industry, and the tasks themselves are similar in nature.

The maintenance tasks performed by Software Engineers in this study are actual maintenance tasks encountered in a large software services company. The variability that is inherent in these tasks was minimized as much as possible for the purposes of this investigation. A large number of maintenance tasks was included in order to get a better fix on this variation. It is assumed that the tasks themselves are standard in nature, but the problem with this assumption is that the results based on the use of the reverse engineering CASE tool could be jeopardized if the variability of the maintenance tasks is too great.

The Software Engineer's knowledge of the tool cannot be generalized.

If a Software Engineer is experienced in using this reverse engineering CASE tool, or if he or she has had any prior experience with this tool, this will not enhance his or her ability or productivity in doing a maintenance task without the aid of the tool.

Any "learning curve" that would suggest an increase in proficiency with the tool during the course of this study was eliminated to the extent possible.

An equal mastery of the tool was presumed for all Software Engineers participating in this research. All Software Engineers attended the same training session on the use of the tool, including a project workshop for each participant. Then, the correct completion of an identical project by all Software Engineers, on an individual basis, was taken to mean that all participants were at the same level of experience with the tool, and all would be equally proficient in its use.

CHAPTER II

REVIEW OF THE LITERATURE

The literature reviewed for this study varied from text books discussing the process of performing maintenance tasks, to the CASE tools available in the market place. The subject of re-engineering has been given very little attention in terms of published information. The topic itself is very new to the industry, and it is only beginning to gain recognition by software services companies.

Most literature available addresses the heart of the problem, which is the programs themselves. Approximately 80% of our existing code was written prior to any structured programming concepts being introduced (Schneidewind, 1987). If the original programs were not designed for easy maintenance, then they are very difficult to change without affecting other portions of the code. In addition, most of these systems do not have adequate and up to date documentation to support the code (Schneidewind, 1987). Even though these facts are obvious in the software services industry, it is neither economical nor feasible to throw away those older programs (Digital Consulting, Inc., 1988). Also, it is inevitable that program maintenance will always exist due to changes in customer demands, software upgrades and changes in operating environments (Digital Consulting, Inc., 1988). Several approaches to maintenance have been published, including code reviews by teams of Software Engineers, redesign and redefinition of the module or section of code that is intended to be changed, and structured testing techniques with

documented test plans that would ensure that no additional defects have been introduced in the code (Pressman, 1987; Schneidewind, 1987; Henry, 1981; Kafura, 1987). Several companies have introduced new development methodologies that enforce structured design and analysis techniques with design and code reviews, including complete documentation of all development outputs (Zvegintzov, 1989; Thebaut, 1987; Software Engineering Research Center, 1988). This systems life cycle has proven successful in reducing maintenance that is required on those systems. The fact remains, though, that many existing systems are old, unstructured and defect ridden (Schneidewind, 1987; Digital Consulting, Inc., 1988). For this reason, the software services industry is looking for automated methods to assist in maintenance.

First, and foremost, the Software Engineers must understand the functions of the program they are maintaining (Software Engineering Research Center, 1987; Borst 1979). Most often, this is done through analysis of the code, "walking" through the logic, following the branching through the code and changes in data as they progress. Once the code is understood, the correct portions of the code must be modified and tested (Interrante, 1988; Thebaut, 1988; Wilde, 1988). The automated methods available to assist Software Engineers with these tasks vary due to the variety of tasks that need to be performed to adequately maintain a program. These tools have been reviewed only from a functional standpoint, at most, and not in terms of their actual value in the software services environment (Language Technology, 1988; Software Engineering Research Center, 1988).

Some literature is available which discusses software renewal

and reverse engineering. Case studies have been performed by some companies to renew their software (Sneed, 1984; Software Engineering Research Center, 1988). Bertelsmann, the world's second largest publishing company, embarked on a software renewal project which cost two thirds of the original expense of the software development project. The renewal project was completed with the use of a CASE tool that aided the reverse engineering process. Without the tool, the renewal project would have been impossible. Their case study convinced them that automation was the only answer to reducing the maintenance backlogs found in most software services companies.

Automation is also the only solution to permanently removing bugs from defect ridden programs that are years old (Sneed, 1987). The problem is that the re-engineering CASE tools have not yet evolved to a state wherein they are completely usable in the industry (Digital Consulting, Inc., 1988; Zvegintzov, 1986). Re-engineering tools, as they now exist, cannot address the entire reverse engineering process. Once this process is automated, however, it is anticipated that three-year projects will be reduced to three-month projects through the use of expert systems that can take program code and extract the business rules and high-level designs for the systems (Bachman, 1988).

All of the literature addressing re-engineering agrees that full automation of the reverse engineering process is several years away (Digital Consulting, Inc., 1988; ViaSoft, 1988; Language Technology, 1988; Transform Logic, 1988). However, at the present time, progress can be made in limited domains by using automated analysis methods to assist the Software Engineers in maintenance tasks (Software

Engineering Research Center, 1988; Hartford Insurance Group, 1988; Digital Consulting, Inc., 1988). In the ideal world, a tool would be available for Software Engineers to ask questions regarding a program change that were as basic as the questions that the most experienced Software Engineer who ever worked on that program might ask (Bachman, 1988).

Due to the obvious immaturity of the reverse engineering CASE tools on the market today, not surprisingly, studies show that most software services companies are not willing to take the risk of investing in these new tools, fearful that there will be no return on their investment (Zvegintzov, 1988). The only research that has been done regarding the effectiveness of these currently available tools is vague at best (Zvegintzov, 1988). These studies have produced a wide variety of results, ranging from no impact on maintenance efforts to very significant impacts (Federal Software Management Support Center, 1987). A close-up look suggests that those companies experiencing a significant impact are estimating the additional costs they feel they would have expended had they not used the reverse engineering CASE tool (Language Technology, 1988; ViaSoft, 1988). No studies are available that make a comparison of actual maintenance expenditures before and after the introduction of a reverse engineering CASE tool in the Software Engineering area (Zvegintzov, 1989). Some studies indicate that the tools do have limited success in limited environments (Federal Software Management Support Center, 1987; Software Engineering Research Center, 1988; Digital Consulting, Inc., 1988). Typically, the most inexperienced programmers, working with more complex systems, exhibit the greatest

productivity improvements. The experienced programmers already have a good understanding of the programs they support, and a tool does little to assist them with analysis (Henry and Kafura, 1981).

CHAPTER III

METHOD

As noted earlier, the purpose of this study is to examine the effects of reverse engineering CASE tools on software maintenance tasks performed by Software Engineers. The construct used to investigate these effects involved group comparisons that would identify any significant differences. Ultimately, these comparisons were intended to demonstrate the kind of impact a reverse engineering tool can have whether it be a global impact within a software services company, a limited impact within certain environments, or no impact at all.

The research questions address the various environments that can be found in a software services company and their relationship to the performance of a CASE tool. The first question deals with the overall relationship of a CASE tool's performance and productivity improvements in the maintenance of programs and systems. The second question addresses potential differences in productivity improvements with a CASE tool between experienced and inexperienced Software Engineers when a CASE tool is used. The final question speaks to productivity improvements using a CASE tool on programs of varying complexity.

Population

The population for this study consists of the Software Engineers of the United States software services industry. This industry develops and maintains computer software.

Sample

The sample for this study consists of the Software Engineers of one United States software services company that develops and maintains programs and systems for a large corporation. Only one company, which is representative of the software services industry, was sampled due to practical considerations with regard to this thesis and data availability. (Note that the company chosen for inclusion in this study was not randomly selected; nor were the Software Engineers from within the company randomly chosen.)

Instrument

The instruments used to collect the data were a survey and a project management instrument (see Appendix A). The survey was completed by a Software Engineer as a maintenance task was completed. It noted the total time taken to complete the task, whether or not the CASE tool was used, and the complexity of the program on which the maintenance task was performed. The project management system was used to capture actual time spent on maintenance prior to the introduction of the tool.

Design

The comparison groups were formed by choosing participants that were classified as "experienced" and "inexperienced" in the terms defined earlier. These participants performed a variety of tasks on programs of differing complexity. Some of the tasks were performed with the CASE tool and other tasks were performed without it.

The participants in this study of the effects of the reverse engineering CASE tool were a group of sixteen Software Engineers. Eight of the sixteen Software Engineers were "experienced" and eight were "inexperienced". The maintenance tasks to be performed were divided up into four categories: simple program with aid of the tool, complex program with aid of the tool, simple program without aid of the tool and complex program without aid of the tool. Each of the eight inexperienced Software Engineers was randomly assigned to a group so that two were made to fall into each of the four categories. In the same way, the eight experienced Software Engineers were also randomly assigned to groups representing these four categories (Table 3.1).

Table 3.1 - Study Categories

	inexperienced	experienced
Category 1 - simple, tool	2	2
Category 2 - complex, tool	2	2
Category 3 - simple, no tool	2	2
Category 4 - complex, no tool	2	2

In order to randomly assign the eight inexperienced Software Engineers to the four categories, each one's name was placed in a hat. Placed in another hat was eight slips of paper, two slips of paper for each of the four categories described above. One by one, a name was drawn out of one hat and the category was drawn out of the second hat. This procedure was repeated for the eight experienced Software Engineers. In this manner, each of them, also, was assigned to one of the four research categories.

As the maintenance tasks arose, a Software Engineer would record information regarding his or her assigned category of work on the instrument survey (see Appendix A). Each Software Engineer recorded three maintenance tasks.

When all the recording was complete, there were twelve maintenance tasks associated with each of the four categories -- six of the tasks performed by inexperienced Software Engineers and six of the tasks performed by experienced Software Engineers. A total of 48 maintenance tasks were recorded.

The Hypotheses

The data analysis for answering the research questions is based on the following research hypotheses and statistical hypotheses:

Research Question 1: Does the use of a reverse engineering CASE tool reduce the amount of time required to maintain programs?

Research Hypothesis 1: Use of a reverse engineering CASE tool will reduce the amount of time required to maintain programs.

Statistical Hypothesis 1: Ha: $T_j <> 0$, for j=1 or j=2

Ho: $T_j = 0$, for j=1,2

Ha: The probability of the F value for the tool effect is less than or equal to 0.05 for this study.

Research Question 2: Does the reverse engineering CASE tool have the same effect on the performance of experienced or inexperienced Software Engineers?

Research Hypothesis 2: The use of a tool will have a greater effect on the performance of inexperienced Software Engineers than on experienced Software Engineers.

Statistical Hypothesis 2: Ha: $(ET)_{ij}=0$, for all i,j

Ho: $(ET)_{ij}<>0$, for some i,j

Ha: The interaction of experience and tool effect will result in an F value whose probability is less than or equal to 0.05.

Research Question 3: Does the reverse engineering CASE tool equally facilitate work on simple and complex programs?

Research Hypothesis 3: The use of the tool will result in a greater productivity gain for complex programs than for simple programs.

Statistical Hypothesis 3: Ha: $(TP)_{jk}<>0$, for some j,k

Ho: $(TP)_{jk}=0$, for all j,k

Ha: The interaction of program complexity and tool effect will result in an F value whose probability is less than or equal to 0.05.

CHAPTER IV

RESULTS

Results Summary

Based on the data collected and the analysis of variance that was subsequently performed, a reverse engineering CASE tool appears to have a significant impact on Software Engineers' productivity. The effect appears to be the same regardless of the Software Engineers experience. Apparently, this effect is also the same regardless of program complexity. Inspection of the variance associated with program complexity, however, reveals an interesting reversal. The variance increased when the tool was used with complex programs even though the time to complete a task with the tool was reduced. This reversal is probably due more to experimental control than to effect.

Although care was taken to control all software engineering groups when collecting the data, some unexpected anomalies did occur. Two Software Engineers, who had been assigned to the complex program category, performed maintenance on simple programs. The data collected by these individuals for the simple programs was eliminated from the analysis. Two other Software Engineers that were assigned to the simple program category, performed maintenance on complex programs. This data, too, was eliminated from the analysis. In addition, there were three cases in which the Software Engineer had been previously exposed to the program, but for different problems and reasons. These incongruities are not considered significant

enough to have effected the results of this study.

Descriptive Statistics

The descriptive statistics derived from the analysis of variance are shown in table 4.1.

Table 4.1 Descriptive Statistics for Reverse Engineering CASE Tool Use

Experience	Tool	Program	N	Time	Std. Dev.
Exp			24 (8 X 3)	130.00	84.7
Inexp			24 (8 X 3)	205.00	127.8
	No		24 (8 X 3)	216.25	117.7
	Yes		24 (8 X 3)	118.75	87.4
		Complex	24 (8 X 3)	216.25	104.7
		Simple	24 (8 X 3)	118.75	102.6
Exp	No		12 (4 X 3)	175.00	76.6
Exp	Yes		12 (4 X 3)	85.00	68.8
Inexp	No		12 (4 X 3)	257.50	139.2
Inexp	Yes		12 (4 X 3)	152.50	93.5
Exp		Complex	12 (4 X 3)	177.50	86.2
Exp		Simple	12 (4 X 3)	82.50	51.5
Inexp		Complex	12 (4 X 3)	255.00	110.4
Inexp		Simple	12 (4 X 3)	155.00	128.5
	No	Complex	12 (4 X 3)	277.50	77.9
	No	Simple	12 (4 X 3)	155.00	121.2
	Yes	Complex	12 (4 X 3)	155.00	93.0
	Yes	Simple	12 (4 X 3)	82.50	66.7
Exp	No	Complex	6 (2 X 3)	230.00	62.0
Exp	No	Simple	6 (2 X 3)	120.00	42.4
Exp	Yes	Complex	6 (2 X 3)	125.00	76.9
Exp	Yes	Simple	6 (2 X 3)	45.00	25.7
Inexp	No	Complex	6 (2 X 3)	325.00	64.1
Inexp	No	Simple	6 (2 X 3)	190.00	166.1
Inexp	Yes	Complex	6 (2 X 3)	185.00	104.6
Inexp	Yes	Simple	6 (2 X 3)	120.00	75.9

A visual inspection of the mean times shows general agreement with the research expectation. Independent of either Software Engineering experience level or program complexity, the effects of the tool are clear (a mean time of 216.25 without the tool versus a mean time of 118.75 with the tool).

A visual inspection of the Standard Deviations reveals some interesting shifts. Holding the complexity of the programs constant, the variance appears to decrease, for both experienced and inexperienced Software Engineers, when the tool is used. However, when the complexity of the program is considered, the variance actually increases with the use of the tool for complex programs. This is true for both experienced and inexperienced Software Engineers.

Detailed Summary

Question 1: Does the use of a reverse engineering CASE tool reduce the amount of time required to maintain programs?

Summary: The results of the analysis of variance indicate that the null hypothesis can be rejected. The use of a tool appears to have a significant impact on productivity as measured by the time to complete a task.

Elaboration: The results of the ANOVA are given in Tables 4.2 and 4.3.

Table 4.2 ANALYSIS OF VARIANCE STATISTICS

Dependent Variable: TIME TIME MEAN: 167.50

MEAN SQUARE FOR THE MODEL: 43585.71

MEAN SQUARE FOR THE ERROR: 7578.75

F Value = MEAN SQUARE(MODEL) / MEAN SQUARE(ERROR) = 5.75

PROBABILITY ASSOCIATED WITH F Value (PR > F) < .01

R-SQUARE (variation in the dependent variable) = 0.50

SQUARE ROOT OF MEAN SQUARE FOR THE ERROR = 87.06

Table 4.3 ANALYSIS OF VARIANCE - INTERACTION STATISTICS

	ANOVA Sum of Squares	F value	PR > F
Experience	67500	8.91	0.0048
Tool	114075	15.05	0.0004
Experience*Tool	675	0.09	0.7669
Program	114075	15.05	0.0004
Experience*Program	75	0.01	0.9213
Tool*Program	7500	0.99	0.3258
Experience*Tool*Program	1200	0.16	0.6928

As noted in Table 4.2, the F value for the model is highly significant (F value of 5.75, and PR > F is 0.0001). Table 4.3 shows that the tool contributes significantly to the effect (F of 15.05, and PR > F is 0.0004).

Question 2: Does the reverse engineering CASE tool have the same effect on the performance of experienced and inexperienced Software Engineers?

Summary: Here, the null hypothesis cannot be rejected. While the results reveal a strong "experience" effect, the tool appears to benefit both experienced and inexperienced Software Engineers equally.

Elaboration: Table 4.3 reveals that "experience" significantly contributes to the variance (F = 8.91, and PR > F is 0.0048). Yet the "experience*tool" effect results in an F value of only 0.09.

Question 3: Does the reverse engineering CASE tool equally facilitate work on simple and complex programs?

Summary: The null hypothesis here cannot be rejected either. Again, while the results show a very strong "program" effect, the tool appears to work equally as well for both complex and simple programs. As a side note, the variance actually increases with the use of the tool for complex programs, regardless of the experience level of the Software Engineer (see page 37).

Elaboration: Table 4.3 reveals that "program" complexity

contributed as much to the overall variance as did the tool (F = 0.0004, and PR > F is 15.05). This is not surprising given the mean times and standard deviations reported in Table 4.1. What is surprising is the indication that use of the tool actually increases the variance when complex programs are maintained! However, as with the "experience*tool" effect, the effect of the "tool*program" interaction is quite small (F = 0.99, and PR > F is 0.3258).

CHAPTER V

DISCUSSION

Discussion of the Findings

The hypothesis that a reverse engineering CASE tool can improve productivity appears to be confirmed within the limited confines of the environment in which this study was carried out. This productivity increase is equally as great regardless of Software Engineer experience or program complexity. (The productivity improvement found in this study averaged 55%.) The findings of this investigation agree with those of several previous research efforts discussed in CHAPTER II (Language Technology, 1988; Digital Consulting Inc., 1988; Viasoft, 1988; Transform Logic, 1988; Federal Software Management Support Center, 1987; Software Engineering Research Center, 1988; Jones, 1986).

The results of the analysis of variance indicated the null hypothesis that the reverse engineering CASE tool has the same effect on the performance of experienced and inexperienced Software Engineers could not be rejected. It appears, as far as this study is concerned, that the experience level of the Software Engineer does not contribute to the success of the tool. Apparently, within the specified environment, the tool can be used equally successfully by Software Engineers of varying experience levels. (It should be noted that the two groups for years of experience of the Software Engineers were mutually exclusive, as shown in Table 5.1, and the standard

deviation within each of the two groups was comparatively small.) Since the success of the tool does not appear to be limited to or restricted by the experience level of the Software Engineers using the tool, it may prove to be valuable to the software services industry.

Table 5.1

Summary of Results

	Simple Program	Complex Program	Inexp.	Experienced
	McCabe Complexity		Years	
Mean	8.38	143.88	3.75	8.00
Std. Dev.	.88	104.95	.89	1.20

The results of the analysis of variance indicated that the null hypothesis that the reverse engineering CASE tool equally facilitates work on simple and complex programs could not be rejected. According to this study, the complexity level of the program does not add to or detract from the success of the tool. It would appear that the tool can be used successfully with programs of varying complexity. (Again, as did experience and inexperience, the simple and complex programs formed mutually exclusive groups, as shown in Table 5.1.)

As noted earlier, the use of the tool seems to increase the variance when complex programs are maintained with the tool. Two possible explanations for this increase in the variance are suggested

here. First, the method used to measure the complexity of programs (McCabe Cyclomatic Complexity), resulted in considerable variation for complex programs. A glance at Table 5.1 reveals that the simple programs had a complexity standard deviation of .88, while the complex programs had a complexity standard deviation of 104.95. Due to the definition of complexity used in this study, a complex program could theoretically, have a complexity measure of 10 through infinity, since there was no defined upper limit to the complexity measure. Because the programs included in this study varied so much in complexity, the higher variance associated with using the tool may be due to the current definition of "complex" programs.

The second possible cause for the higher variance found when complex programs were maintained is the learning process. Learning to apply the tool to complex programs was, in itself, a formidable task. The initial training session may not have been sufficient for the application of the tool on complex programs. The learning curve for all participants in this study was overcome as much as possible by conducting a standard training class and workshop for all Software Engineers. However, the example used in this class was rather simple. It may be that when the tool was applied to more complex programs, the Software Engineers required additional learning time to handle the more complicated logic of these programs. In order to avoid such a variance reversal in any future duplication of this study, a more thorough and comprehensive training class and workshop may have to be organized.

Conclusions and Implications

There are two conclusions that might be drawn from the results of this study. First, reverse engineering CASE tools appear to effect productivity improvement in the maintenance environment. Second, with minor modification, the approach used in this study is a reasonable pattern for future studies of reverse engineering CASE tools or any other CASE tool that is intended to increase the productivity of Software Engineers.

There are significant implications for tools that can increase Software Engineering productivity when performing maintenance on existing programs. With the enormous backlog of maintenance and enhancements needed by many existing systems, reverse engineering CASE tools can assist in reducing the effort required. But tools of this type can only assist. Tools to automate the entire re-engineering cycle are critically needed if the older, more complex systems that exist today are to be replaced. Until that time, however, any tools that can aid Software Engineers in the difficult task of maintenance will be beneficial.

The implications of having some sort of mechanism to judge the effectiveness of CASE tools is also important. CASE tools will be playing a significant role in the future software services industry on a world-wide basis. Many companies are developing their own tools or purchasing the tools developed by other software suppliers. In either case, the goal is the same -- find a way to reduce the effort needed to maintain current systems so that more time and energy can

be devoted to the development of new systems. As a variety of CASE tools emerge on the market, having a structured and straightforward approach to determine which tool will give the best results is key to helping a software services company choose a tool. The method used in this study was neither complex nor difficult to implement. The results obtained from standard statistical procedures provided a means for determining our tool's effectiveness. This approach can be used with other reverse engineering CASE tools or any other CASE tool that is expected to increase productivity. The results are based on actual project work, as opposed to estimated work efforts that can be inaccurate. Although the decision to purchase or develop a CASE tool has to take other factors into consideration, the effectiveness of the tool to enhance productivity improvements is one of the main considerations. CASE tools that perform similar functions can be compared using the method in this thesis as well, so a software services company can choose the tool having the greatest impact. In summary, this method can be used by the software services industry as a reasonably effective and statistically conclusive means for assessing the relative impact of a CASE tool in its unique environment.

Recommendations for Further Research

There are many different types of CASE tools available today. Some of the tools have a very narrow focus, supporting one portion of the Systems Development Life Cycle (e.g., Design phase), and other tools are much more broad in scope, meant to support multiple

portions of the Life Cycle. Regardless of the CASE tool's function, further research should be done to confirm the tool's effectiveness in the software services industry.

Other reverse engineering CASE tools should be studied in order to generalize the results of this research. All reverse engineering tools are not the same, therefore we cannot conclude that any other reverse engineering CASE tool will produce the kind of results found in this study. Also, this study was performed in one software services company, in one geographic location. Further research would be beneficial to generalize the effects of reverse engineering CASE tools to different companies, in different locations, with different types of applications than described in this particular investigation. Research of that nature would allow any conclusions regarding the effectiveness of a reverse engineering CASE tool to be extended to a larger segment of the population.

Of the other CASE tools that exist, most of them address the Forward Development Process. Research regarding the effectiveness of these tools would also prove interesting. The effectiveness of these tools has never been conclusively, or even objectively, reported. Perhaps an approach similar to the one followed in this thesis could be used to more conclusively study these tools.

There is one last area that would benefit from future research. It was not addressed in this study but should be in conjunction with future CASE tool studies. It is the quality of the final product. Productivity, in and of itself, will not help the software services industry unless quality is built into the products and services produced. For a CASE tool to provide an advantage to the software

services industry, it must not only improve productivity, but must also ensure quality of the final product.

The measurement of quality is often subjective, since quality is defined by the individual receiving the product or service. Two concerns are implied here. First, the software services industry must understand what quality means to each one of its customers. Second, the software services industry must be able to measure its effectiveness in such a way that it satisfies the customer's definition of quality. The same type of issues surround CASE tools. A software services company looking to implement a CASE tool in its own environment needs to know not only the tool's level of effectiveness in increasing productivity, but also whether or not the end results after use of the CASE tool meet the requirements of the tool's user in producing a quality product. This quality could be measured using a statistical method. During the testing of the software, the number of software failures can be measured over elapsed CPU run-time for a program (Musa, 1989). These measurements can then be used, in turn, to compute a probability for software failure over any specified CPU run-time interval. Ultimately, this quantitative measurement would be used to determine whether or not the use of the CASE tool in question has the desired "quality" effect in terms of its ability to decrease the potential number of failures for a specified CPU run-time interval.

BIBLIOGRAPHY

Bachman, Charles, "A Case for Reverse Engineering", Datamation, Cahners Publishing Company, July 1988.

Basili, Victor, "Tutorial on Models and Metrics for Software Management and Engineering", IEEE Computer Society, Catalog no. EHO-167-7, Library of Congress no. 80-83085, 1980.

Borst, Curtis, Love, Milliman, and Sheppard, "Measuring the Psychological Complexity of Software Maintenance Tasks with the Halstead and McCabe Metrics", IEEE Computer Society, March 1979, vol 5. no. 2, pp 96-103.

Bush, Eric and Spencer, Nancy, Language Technology Corporation, "Software Re-Engineering Symposium", 1988.

Digital Consulting, Inc., "Software Re-Engineering Symposium", Fall 1988.

Federal Software Management Support Center, "Parallel Test and Productivity Evaluation of a Commercially Supplied Cobol Restructuring Tool", Office of Software Development and Information Technology, September 1987.

Fink, Arlene and Kosecoff, Jacqueline, "An Evaluation Primer", Sage Publications, Beverly Hills, 1978.

Halstead, Maurice, "Elements of Software Science", Elsevier North Holland Inc., New York, 1977.

Henry, Sallie and Kafura, Dennis, "Software Structure Metrics Based on Information Flow", IEEE Transactions on Software Engineering, vol. 7 no. 5, September 1981, pp. 510-518.

Hoffman and Schneidewind, "An Experiment in Software Error Data Collection and Analysis", IEEE Transactions on Software Engineering, Mary 1979, vol. 5 no. 3, pp. 276-286.

Holbrook, H.B. and Thebaut, S.M., "A Survey of Software Maintenance Tools That Enhance Program Understanding", Software Engineering Research Center, University of Florida, September 1987.

Interrante, Basrawala, "Reverse Engineering Annotate Bibliography", Software Engineering Research Center, University of Florida, January 1988.

Interrante, Thebaut, Wilde, "Reverse Software Engineering: Some Problems and Possibilities", August 1988, Software Engineering Research Center, University of Florida.

Jones, Capers, "An Overview of Software Productivity and Quality Measures", Quality Data Processing, July 1987, pp.8-10.

Jones, Capers, "Programming Productivity", McGraw Hill, 1986.

Kafura, Dennis and Reddy, Geereddy, "The Use of Software Complexity Metrics in Software Maintenance", March 1987, vol. 13 no. 3, IEEE Computer Society, pp. 335-343.

Kerlinger, Fred N., "Foundations of Behavioral Research", Holt, Rinehart and Winston Inc., New York, 1973, pp. 224-238.

McCabe, Thomas J., "A Complexity Measure", IEEE Transactions on Software Engineering, Vol. SE-2, no. 4, December 1976, pp. 308-320.

Musa, John D. and Ackerman, A. Frank, "Quantifying Software Validation: When to Stop Testing?", May 1989, IEEE Software, pp.19-27.

Pressman, Roger S. "Software Engineering: A Practitioner's Approach", McGraw-Hill, New York, 1987, pp432-466.

Schneidewind, N.F., "The State of Software Maintenance", March 1987, vol. 13 no. 3, IEEE Computer Society, pp.303-310.

Sneed, Harry M. "Software Renewal: A Case Study", IEEE Software, July 1984, vol. 1 no. 3, IEEE Computer Society, pp. 56-63.

Sneed, Harry and Jandrasics, Gabor, "Software Recycling", IEEE Transactions on Software Engineering, 1987, pp. 82-90.

Walsh, Thomas J., "A Software Reliability Study Using a Complexity Measure", AFIPS Conference Proceedings, Volume 48, 1979, National Computer Conference, New York, NY. pp. 761-768.

Zvegintzov, Nicholas, "Software Maintenance News", January 1988, vol 6 no 1, pp. 1-4.

Zvegintzov, Nicholas, "Software Maintenance News", August 1986, vol 4 no 8, pp. 1-2.

Zvegintzov, Nicholas, "Software Maintenance News", March 1989, vol 7 no 3, pp. 1-16.

Appendix A

Instrument Survey

Name _____ Date _____

Program Modified	Program Complexity	Tool yes/no	Time
_____	_____	yes or no	_____ hrs or min
_____	_____	yes or no	_____ hrs or min
_____	_____	yes or no	_____ hrs or min
_____	_____	yes or no	_____ hrs or min
_____	_____	yes or no	_____ hrs or min
_____	_____	yes or no	_____ hrs or min
_____	_____	yes or no	_____ hrs or min
_____	_____	yes or no	_____ hrs or min
_____	_____	yes or no	_____ hrs or min
_____	_____	yes or no	_____ hrs or min
_____	_____	yes or no	_____ hrs or min
_____	_____	yes or no	_____ hrs or min

Instructions:
Program Modified - name of the program modified
Program Complexity - complexity measure based on McCabe's Cyclomatic Complexity.
Tool yes/no - Did you use the tool for this program change?
Time - Record the amount of time, in hours or minutes, that you spent modifying this program.

APPENDIX B

SURVEY RESULTS

	Exp(in years)	Program Complexity			Tool Used	Time (minutes)		
Programmer A1	3	9	9	9	yes	240	60	180
Programmer B1	5	9	8	9	yes	120	60	60
Programmer C1	8	9	7	7	yes	60	20	35
Programmer D1	9	9	9	8	yes	15	80	60
Programmer A2	4	165	320	270	yes	240	270	300
Programmer B2	3	46	75	20	yes	60	60	180
Programmer C2	7	62	188	18	yes	180	240	60
Programmer D2	9	34	214	34	yes	120	120	30
Programmer A3	3	8	7	9	no	180	270	480
Programmer B3	3	9	6	8	no	60	120	30
Programmer C3	7	8	9	9	no	120	180	120
Programmer D3	10	9	9	8	no	150	60	90
Programmer A4	4	101	92	193	no	270	300	390
Programmer B4	5	212	63	104	no	420	270	300
Programmer C4	7	219	187	79	no	300	240	180
Programmer D4	7	79	405	203	no	150	300	210