IMPACT OF DEVELOPER ATTITUDE ON SOFTWARE MAINTENANCE COST: A DESCRIPTIVE STUDY BASED ON THE TECHNOLOGY ACCEPTANCE MODEL

by

Boniface C. Nwugwo

A Dissertation Submitted in Partial Fulfillment Of the Requirements for the Degree of Doctor of Philosophy

Capella University

June, 2003
IMPACT OF DEVELOPER ATTITUDE ON SOFTWARE MAINTENANCE COST: A DESCRIPTIVE STUDY BASED ON THE TECHNOLOGY ACCEPTANCE MODEL

by
Boniface C. Nwugwo

has been approved
June, 2003

APPROVED:

FRANK DECARO, Ph.D., Faculty Mentor and Chair
WILLIAM REED, Ph.D., Committee Member
KURT LINBERG, Ph.D., Committee Member
JACK CICHY, Ph.D., Committee Member
WILLIAM LAING, MBA, Committee Member

ACCEPTED AND SIGNED:

FRANK DECARO, Ph.D.

_______________________________________________
Shelley R. Robbins, Ph.D.
Executive Director, School of Business
Abstract
This study examined the impact of the behavioral attitude of software engineers towards software process improvement practices on software maintenance cost. The study used the technology acceptance model to assess the relationship between software developers’ attitudes and software maintenance cost. Data was gathered using a 78-item, Web-based questionnaire that measured software attitude, organizational factors, project factors, process maturity and demographic factors. The results are based on self-reported data from 99 software developers and software project managers working in a Fortune 500 corporation, spanning over 20 projects. While the results showed strong relationships between the research model’s intervening variables, the findings did not show a significant correlation between software developer’s attitude towards software process improvement practices and software maintenance cost. The results show that software developers will adopt and use software process improvement best practices if they perceive them to be useful and easy to use. What is not clear is whether or not developers’ attitudes towards usage of software best practices have any impact on developer perceived software maintenance effort(cost).
Dedication

This work is dedicated to Francisca Ngozi “Gee” Nwugwo. Her encouragement, support, and steadfast love made it possible for me to start and finish a doctoral program while working full-time and helping raise four children under 7. I will forever remain indebted to her. May the Almighty God bless and continue to protect her for me, Amen.
Acknowledgements

I would like to thank some of the key people from the corporation where I worked for their support of this research: William Atkinson and Gail Evans for their support and encouragement to start a PhD program in the first place. Others include Robert Kuehl, Glenn Hyde, Gregg Rabidoux, Tim Brown, Robert Brown, and Rich Ruscio. Without their support I would not have been able to obtain the data to conduct the study that led to this report.

I also wish to acknowledge the helpful assistance of the members of my Dissertation Committee: Dr. Frank DeCaro, Dr. Kurt Linberg, Dr. Bill Reed, Dr. Jack Cichy, and Bill Laing. Although many people provided feedback throughout the dissertation process, the following people provided invaluable comments and assistance: Dr. Roger Pressman, Dr. Karl Wiegers, Mr. Capers Jones, and Dr. Alex Ikejiaku. I appreciate their help in reviewing and validating the instrument I used for this study. My special thanks goes to Beverly J. Bowen, a member of the Capella Mid-Atlantic PhD learners group, for her immeasurable review of the proposal manuscript for APA 5th edition compliance. Her suggestions guided me in writing the dissertation with respect to the APA format.
I would like to thank Dr. Frank DeCaro specially for being an excellent mentor. He was very helpful with his thoughtful suggestions and always enthusiastic about my research. He helped me when I needed it and I knew I could always count on his advice. His experience made the strain of the comprehensive and dissertation processes bearable.

I also acknowledge the compassion and best wishes I received throughout my journey from family members, friends, and work colleagues. Their encouraging words kept me going. My appreciation goes to my precious little ones -- Brian (7), Allen (6), Jaine (5), and Denzel (2) -- for their efforts in keeping me on my toes no matter how tired I was. Last but not the least, I want to acknowledge the most important person in my life, my wife, Francisca. Thank you for your support and constant reminder that “there is a light at the end of the tunnel”, and with your encouragement, I reached the end of that tunnel.
Table of Contents

Acknowledgements....................................................... iv
Table of Contents........................................................... vi
List of Tables.................................................................... ix
List of Figures................................................................... xi

CHAPTER 1 INTRODUCTION AND PROBLEM DEFINITION........ 1
  Introduction to the Study.................................................... 1
  Statement of the Problem.................................................. 3
  Background of the Problem............................................... 5
  Purpose of the Study....................................................... 8
  Theoretical basis for the study.......................................... 10
  The Research Model.......................................................... 18
  Research Questions and Hypotheses.................................. 26
  Significance of the Study.................................................. 28
  Definition of Terms.......................................................... 30
  Assumptions and Limitations............................................ 35
  Organization of the remainder of the study......................... 36

CHAPTER 2 LITERATURE REVIEW.................................. 38
  Introduction and Organizational Structure of the Chapter. 38
  General Theoretical Concepts on Behavior......................... 38
  Software Maintenance Process......................................... 40
  Review of Pertinent Literature......................................... 51
List of Tables

Table 1: Characteristics of the Pilot Test sample............. 70
Table 2: Developer Attitude Scale Questions and Purpose....... 72
Table 3: Developer Attitude Scale Questions and Purpose....... 73
Table 4: Developer Attitude Scale Questions and Purpose....... 74
Table 5: Organization Factor Scale Questions and Purpose...... 75
Table 6: Example of KPA Scale Questions....................... 77
Table 7: Project Factor Scale Questions....................... 78
Table 8: Project Factor Scale Questions....................... 79
Table 9: SMADAS Scale Cronbach’s Coefficients............... 81
Table 10: Breakdown of respondents by gender............... 100
Table 11: Years of Experience of the Study Sample............ 100
Table 12: Educational Level of the Study Sample.............. 100
Table 13: Developers’ Perception of their Organization’s
          Maintenance Attitude ....................................... 101
Table 14: Spearman’s Correlation Matrix of Study Constructs.. 105
Table 15: Perceived Ease of Use Construct..................... 106
Table 16: Perceived Usefulness Construct....................... 107
Table 17: Attitude Construct.................................. 108
Table 18: Actual Use Construct................................ 109
Table 19: Behavioral Intentions Construct...................... 110
Table 20: Real Business Impact Construct...................... 111
List of Figures

Figure 1. Theory of Reasoned Action................................. 14
Figure 2. Theory of Planned Behavior............................... 15
Figure 3. Technology Acceptance Model............................. 17
Figure 4. Extended TAM With RBI................................. 20
Figure 5. Scales for measuring the research model’s constructs 25
Figure 6. Intersection of development and maintenance processes ...................................................... 41
Figure 7. The Classic Waterfall Model.............................. 42
Figure 8. The Spiral Model of Development....................... 48
Figure 9. Power as a function of Sample Size..................... 67
Figure 10. Histogram of the Product Size......................... 102
Figure 11. Histogram for Total Effort.............................. 103
Figure 12. Results of Developer Attitude on SW Maintenance 105
Figure 13. Scatterplot of Effort and Process Maturity.......... 121
Figure 14. Scatterplot of Low PPMAT and High PPMAT projects .. 124
Figure 15. Type of Current Work by Software Developers...... 126
Figure 16. Percent of Time on Maintenance by Developers...... 127
Figure 18. Effort Distribution by Type of Maintenance........ 130
CHAPTER 1 INTRODUCTION AND PROBLEM DEFINITION

Introduction to the Study

The cost of each phase of the software development life cycle as a proportion of the overall software cost has been the subject of many studies in the past including IBM, IEEE, and DoD (Putnam, 1980; Roetzheim, 2000; Sneed, 1989; Walston & Felix, 1977; Wolverton, 1974). One common feature among these studies is that the software maintenance phase costs are disproportionately larger than any of the lifecycle phases.

Software maintenance is a task that every software development organization has to face after the exhilarating rush of getting the product out the door. What makes it particularly an important aspect of a software-based system is that it consumes between 40 percent and 80 percent of a typical product's total software life-cycle expenditures (Jones, 1998; Putnam & Myers, 1992; Sneed, 1989; Vallabhaneni, 1987). Moreover, software engineers spend over fifty percent of their time maintaining existing software in an operational mode (Vallabhaneni, 1987). Given this high cost of software maintenance, some organizations view their maintenance processes as an area in which to gain a competitive advantage (Moad, 1990).
Nothing explains this researcher’s concern better than the following quote:

Of course, we know much more about the phenomenon now. We've confirmed, over and over again, what the 1972 report first told us: approximately 50% of the DP budget in most organizations is devoted to maintenance. During the late 1980s, maintenance in some organizations climbed to as high as 80% of the total DP budget, but most of the recent surveys by software statisticians like Capers Jones, Howard Rubin and Larry Putnam indicate that it's retreated back to the 50% level again. The vast software metrics databases collected by Jones, Rubin, and Putnam also tell us that approximately 20% of the maintenance budget is typically spent fixing bugs that were introduced during the original development of the system (Yourdon, Variale, Rosetta, Steffen, & Rubin, 1994).

Whether these metrics are over or under estimated, the consensus among all the studies and supporting metrics is that maintenance costs are higher than any other software life-cycle activity.

Over the last two decades a lot of tools and techniques were developed and introduced for improving software development and maintenance. These include Structured Programming, Structured Analysis, Structured Inspections, Formal Methods, Computer Aided Software Engineering (CASE), Object Oriented design, and new testing techniques. Despite the availability of these various tools and techniques, software maintenance cost continues to consume more of the overall cost of software-based systems. That implies that there must be some other factor or factors other than tools, techniques and processes, and people
capability that cause software maintenance cost to be high. It seems that there is a correlation between software developers’ attitude (the importance they attach to the practices and processes that enhance software maintainability) and software maintenance cost. Therefore, this research examined the effect of software developers’ attitude towards software improvement processes on software maintenance cost.

In this first chapter, an overview of the need to study and understand the factors affecting an organization’s ability to reduce software maintenance cost is presented. The subsequent chapters provide a comprehensive review of major literature related to software maintenance, and nature of the study with the description of how the variables of the methodology were operationalized.

Statement of the Problem

One of the most significant challenges facing software development organizations today is how to reduce software development costs, especially the software maintenance cost aspect of the overall software life-cycle cost. Several studies (Boehm, 1987; Dion, 1993; Emam & Briand, 1997; Herbsleb, Carleton, Rosum, Siegel, & Zubrow, 1994; Humphrey, 1989; Humphrey, Snyder, & Willis, 1991; Paulk, 1996; Paulk, Weber,
Curtis, & Chrissis, 1995; Pipp, 2000; Sneed, 1989; Tvedt, 1996; Tvedt & Collofello, 1995) have reported that maintenance consumes a majority of the total cost of software-based systems. Although many tools, techniques and people processes have been developed and introduced for improving software development and maintenance, software development cost continues to skyrocket with the maintenance cost consuming over fifty percent of the overall cost. Therefore, there must be other reasons such as the software developers’ attitude that contribute to the high cost of software maintainability.

For any improvement to be made in reducing maintenance cost, companies need to know why it is very difficult for software organizations to reduce maintenance cost. If we know the reasons, then we can try to do something about them. So, as there is very little understanding of all the factors that influence organizations to reduce the cost of software maintenance, particularly, behavioral factors, it would be beneficial for software organizations to understand what those factors are. For that reason, the focus of this study was on determining if there is a correlation between software developers’ attitudes (or the importance they attach to the practices and processes that make software maintainable) and software maintenance cost. Therefore, the research is titled:
“Impact of Software Developers’ attitudes on software maintenance cost: A descriptive study based on the Technology Acceptance Model.”

Background of the Problem

For the past several years, the invisible software has been the driving force behind the information technology industry’s growth surpassing the visible hardware component in importance. The importance of software became more profound than hardware particularly in the last two decades. That importance attached to software came with it a hefty price. Today, software component constitutes up to seventy-five percent of the overall cost of a system (Standish-Group, 1995). A fairly large portion of this cost is attributable to software maintenance. It was not always this way. But as hardware and software advances have made it possible to build bigger software systems, the price of building these software systems have also skyrocketed. For that reason, a lot of focus was placed on tools and technology for software cost reduction. In the past decade, the primary focus in the IT implementation research has been on how potential users' perceptions of the information technology (IT) innovation influence its adoption. Why do users accept or reject information systems? How is user acceptance affected by
perceived usefulness, perceived ease of use, and attitude toward acceptance behavior? There has been no attempt to apply the technology acceptance model to software development in order to understand why software maintenance cost continues to go up. Monitoring the software developers’ attitudes and developing an understanding of the variables that affect developer attitudes will assist management in providing appropriate development environment in which software developers can succeed in building maintainable software, thereby reducing software maintenance cost.

This study utilized the Technology Acceptance Model (TAM) to investigate factors other than tools and technology that may influence software maintenance cost, particularly, the effect of software developers’ attitudes towards software improvement practices and techniques that have been proven to minimize maintenance cost. It was accomplished through survey questionnaires and evaluations. A survey was conducted to obtain software maintenance information from a sample of software development organizations through self-reporting (i.e., the people in the sample responded to a series of questions designed to elicit characteristics of the organization and the factors that impede or enhance software maintenance cost). The key measure for software maintenance cost is the average cost spent
on a software life cycle after the initial operational
capability of a program, for all projects in development for a
given calendar year. For the purposes of this study, ‘cost’ is
used synonymously with effort to mean the amount of person-hours
required to carry out a task. Unless otherwise stated, the terms
‘cost’ and ‘effort’ mean the same thing.

The population that was sampled comprised of software
developers and software project managers in software
organizations. For the purposes of this study, a software
organization means any software development group with twenty or
more people with responsibility to create software products (or
integration into a larger system) for commercialization. What
this means is that in a company with various divisions, there
could be several of such organizations. The sample organizations
were obtained from a Fortune 500 company. These organizations
varied in size ranging from thirty to more than one hundred
people. Using data from various organizations even though they
are from one mother company ensured that the sample was
representative of the population, that is those organizations
that develop computer-based systems for commercialization. To
protect confidentiality of the Fortune 500 company involved in
this study (herein referred to as “the corporation”), the
corporation is not named. Also for reasons of confidentiality,
neither the corporation’s specific software organizational
groups nor the projects within these groups was identified. The
corporation has a large staff, which is supported by many
software-intensive projects.

Purpose of the Study

There is no question about the importance of reducing
software maintenance costs for software organizations that truly
want to gain competitive advantage through pricing. An important
concept that highlights the role of competitive advantage in
today’s environment is the "value chain." This concept divides a
company's activities into the technologically and economically
distinct activities ("value activities") it performs to do
business. The value a company creates is measured by the amount
that buyers are willing to pay for a product or service. A
business is profitable if the value it creates exceeds the cost
of performing the value activities. Therefore, to gain
competitive advantage over its rivals, a company must either
perform these activities at a lower cost or perform them in a
way that leads to differentiation and a premium (Thompson Jr. &
Strickland III, 1999).

In order to appreciate the competitive advantage that
could be realized by reducing maintenance cost for the software
life cycle, computer-based systems development life cycle should be looked at as a “value chain”. A typical software life cycle consists of requirements and definition phase, system design phase, detailed design phase, implementation (coding, unit testing, and integration testing), system test phase, operation and maintenance. Each of these phases can be thought of as an activity in the value chain of building computer-based systems. These activities basically can be consolidated into just three activities, namely, planning, development (i.e., requirements and definition, system design, detailed design, coding, unit testing, and integration testing), and maintenance. Each of the three consolidated phases contributes to the overall total cost of building software. The cost of planning the software project accounts for ten percent of the total cost, development costs account for thirty percent, and maintenance costs account for sixty percent (Sneed, 1989). Assuming that this distribution is true and all studies and metrics databases support that it is, then a reduction in maintenance costs for any company would mean a certain competitive advantage for that company. The question is why can’t companies reduce maintenance cost even though maintenance activities consume more than fifty percent of the overall costs?
Therefore, the purpose of this study was to determine the factors that impede or enhance the ability of software organizations from reducing their software maintenance costs, specifically, the study examined the relationship between software developers’ attitudes towards software improvement best practices and software maintenance cost. The study sought to:

1. Understand the relationship between the behavioral attitudes of software developers towards software improvement processes and software maintenance costs.

2. Understand what accounts for the total maintenance workload – adaptive, corrective, perfective or preventive -- for software engineers.

3. Identify new factors that contribute to high software maintenance cost.

4. Determine how well maintenance is addressed in process improvement initiatives.

5. Make recommendations on the next step and how organizations should approach software development process improvement to reduce software maintenance cost.

Theoretical basis for the study

Software maintenance lacks a strong theoretical foundation. Even though important key concepts have been proposed in the past, there is no general agreement on the definition, the classification of maintenance activities, the modeling of the maintenance process, and even on name itself. This is confirmed by the fact that many software lifecycle models do not include
software maintenance as a separate activity. However, software lifecycle models describe the software development and maintenance processes. Quite a few of these lifecycle models have been developed and are in use. These include the classic lifecycle model also called the "waterfall model" (Boehm, 1976; Royce, 1970), the spiral model (Boehm, 1988), and corrective maintenance maturity model (CM³) (Kajko-Mattsson, 2000). Other related models include the Theory of Reasoned Action/Theory of Planned Behavior (Ajzen & Fishbein, 1980) and the Technology Acceptance Model (Davis, 1989).

The waterfall model is the oldest and most widely used method for software engineering. It integrates the maintenance phase as a separate activity that happens at the end of the lifecycle. However, over the past few years, some criticisms have been leveled against the paradigm causing even active supporters of the model to question its applicability in every situation. Some researchers have proposed a number of software life-cycle models partly or completely ignoring the phase of software maintenance. Most of these models do not provide clear guidelines on how to integrate maintenance into a development process. The top-down waterfall model has been widely accepted by the software community. It sees maintenance as a single phase in the post development chain.
The spiral model covers software development and software maintenance coherently. Although the model does not explicitly integrate the two processes, and does not explain which information is shared between the development process and the maintenance process, or how the information is interchanged between the two processes, one can argue that the two processes do mesh together after the first iteration. The spiral model has received considerable attention as an evolutionary development technique, using the waterfall model for each step emphasizing the risk analysis aspect of a software project.

Like an organism, organizations can be thought of as living systems that exist in wider environments on which they depend on for satisfying their various needs. One such need is an organization’s health and development (Morgan, 1997). The concept of organizational health and development along with other types of organizational needs raises the more general issues of organizational survival, organization-environment relations, and organizational effectiveness. An organization’s need to survive (i.e. lower cost and increase margin) can be likened to the behavioral theory of reasoned action.

The theory of reasoned action (TRA) provides a framework to study attitudes toward behaviors. It provides a construct that links individual beliefs, attitudes, intentions, and behavior
Impact of developer attitude

(Fishbein, Middlestadt, & Hitchcock, 1994). According to Fishbein et al (1994), the main factor that determines a person’s behavior is intent. A person’s intention to perform a behavior is comprised of attitude toward performing the behavior and subjective norm. The person’s attitude toward the behavior includes: behavioral belief, evaluations of behavioral outcome, subjective norm, normative beliefs, and the motivation to comply.

The theory of reasoned action (TRA) is often cited in the literature on behavior. It grew out of the 19th century when psychologists began to look at the term "attitude". Many theories that emerged during that era suggested that "attitudes could explain human actions" (Ajzen & Fishbein, 1980, p.13). In 1967, Icek Ajzen and Martin Fishbein collaborated in exploring ways to predict behaviors and outcomes. After reviewing all the studies, they developed a theory that links individual beliefs, attitudes, intentions, and behavior (Fishbein et al., 1994). Their construct, which became known as the Theory of Reasoned Action as shown in Figure 1, suggests that an individual’s behavior is determined by his/her intention to perform the behavior. This intention is in turn, a function of the individual’s attitude toward the behavior and his/her subjective norm (Ajzen & Fishbein, 1980). Their assumption was that
individuals are usually rational human beings who make systematic use of information available to them. In other words, people usually consider the implications of their actions before deciding to engage or not to engage in a given behavior (Ajzen & Fishbein, 1980).

But the TRA had some drawbacks. It works very well when applied to behaviors that are under the person's volitional control. However, if behaviors are not fully under the individual’s volitional control, even though the person may be highly motivated by his/her own attitudes and subjective norm, intervening environmental conditions may inhibit the person from performing the behavior. For this reason, a modification to the theory was needed and the result was the theory of planned behavior. Like the original theory of reasoned action, a central factor in the theory of planned behavior is the individual’s
intention to perform a given behavior. Intentions are assumed to capture the motivational factors that influence a behavior. They are indications of how hard people are willing to try, of how much of an effort they are planning to exert, in order to perform the behavior. As a general rule, the stronger the intention to engage in a behavior, the more likely it would be performed.

The Theory of Planned Behavior (TPB) as shown in Figure 2, is basically the TRA plus perceived behavioral control (Ajzen, 1991). Perceived behavioral control indicates that one’s motivation is influenced by how difficult the behaviors are perceived to be, as well as the perception of how successfully the person can, or cannot perform the activity.

![Figure 2. Theory of Planned Behavior](source: Ajzen, 1991, pp. 179-211)
The TPB postulates that a software developer’s intention to follow a structured process precedes the process behavior and will be determined by three things. (a) the individual’s evaluation of possible outcomes of process improvement behavior (attitudes toward behavior); (b) the individual’s perception of the expectations of others around him/her concerning process improvement; and (c) the degree to which one believes they can control their process improvement behavior. The perceived behavioral control is determined by two factors -- Control Beliefs and Perceived Power (Ajzen, 1991).

The Technology Acceptance Model (TAM) is an adaptation of the Theory of Reasoned Action (TRA) for explaining computer usage behavior (Davis, 1986, 1989). In the TAM, the attitudinal determinants of TRA are replaced by two distinct variables -- perceived ease of use (EOU) and perceived usefulness (U). Like TRA, TAM postulates that actual usage of software maintainability practices is determined by behavioral intention, but differs in that the intention is jointly determined by the person’s attitude toward using the process and perceived usefulness. As depicted in Figure 3, the TAM model identifies perceived ease of use, and perceived usefulness as key independent variables.
Perceived Ease of Use is defined as “the degree to which a person believes that using a particular system would be free of effort” (Davis, 1989, p. 320). Perceived Usefulness (U) is defined as “the degree to which a person believes that using a particular system would enhance his or her job performance” (Davis, 1989, p. 320). Behavioral Intention is the measure of the strength of one’s intention to perform a specified behavior (Fishbein & Ajzen, 1975, p. 288). Attitude is defined as the individual user’s positive or negative feelings (evaluative affect) about performing the target behavior (Fishbein & Ajzen, 1975, p. 216). Actual Use is measured in terms of frequency of system use (‘how often’) and the volume of system use (‘how much’) by the user.

Both the perceived usefulness and the perceived ease of use are modeled as having a significant impact on a user's attitude.
toward using the system (A). Behavioral intentions to use (BI) are modeled as a function of A and U, while BI determines actual use. BI has been consistently shown to be the strongest predictor of actual use (Davis, Bagozzi, & Warshaw, 1989; Taylor & Todd, 1995). Perceived usefulness is also indirectly influenced by perceived ease of use. The TAM includes the very important assumption that behavior is voluntary or at the discretion of the user. The TAM model has been tested in several studies of software use (Adams, Nelson, & Todd, 1992; Davis et al., 1989; Mathieson, 1991).

This study will apply the technology acceptance model to explain the developer’s attitudinal and normative influences on intentions to reduce software maintenance cost, a practice that would improve a company’s competitive position by lowering its overall cost of software-based systems. The other reasonable approaches in terms of formal models are the maintenance assistance capability for software, IEEE Standard 1219-1992 maintenance model, and a few more others. A broader literature review will examine the theoretical models in much detail.

The Research Model

The TAM has been used successfully for more than a decade in predicting and explaining the acceptance of many types of end
user computing systems in work or school settings. However, it has not been applied to the single-most expensive aspect of software-based system development -- software maintenance. It is vitally important to study the acceptance of software maintenance cost reduction as a competitive advantage since it eats up more than fifty percent of software costs and should have a tremendous impact on the profitability and economic survival of software organizations. Understanding more about the acceptance of maintenance cost reduction can lead to significant improvements in the design of software to increase its maintainability. Maintainability here is defined qualitatively as the ease with which software can be understood, corrected, adapted, and/or enhanced.

To understand the impact of software developer attitude on software maintenance cost, the researcher proposed an extension of the TAM as shown in figure 4. This model enabled the researcher to explore the type of relationship that exists between the independent variable (software developer attitude towards software process best practices) and the dependent variable (software maintenance cost). Therefore, the research model for this study is the TAM model, plus an extension called real business impact (RBI). The RBI contains the expected
business impact (such as reduction in cost of software maintenance) that can occur as a result of the user’s attitude.

As Figure 4 shows, the dependent variable – Developer’s Perceived Software Maintenance Cost (Effort) -- is depicted as one of the real business impacts that may be influenced by the developers’ attitudes and behaviors. Modeling the definition of Perceived Ease of Use (EOU) as defined by Fishbein & Ajzen (1975), for this study Perceived Ease of Use is the degree to which a software developer believes that using a particular software improvement best practices would require little or no effort. Perceived Usefulness is the degree to which a software developer believes that using software improvement best practices would enhance his or her job performance. Behavioral Intention is the
measure of the strength of the developer’s intention to use software improvement best practices. Attitude is the developer’s positive or negative feelings about using software improvement best practices. Actual Use is measured in terms of frequency of use of software improvement best practices and the volume of software best practices use by the developer. These constructs are measured using scales identified in Figure 5.

Drawing from the TAM, perceived usefulness of software process improvement and the developer’s perceived ease of use of software best practices are hypothesized to influence developer attitudes toward software engineering best practices. This in turn influences developer’s behavioral intentions to use those software best practices, which in turn influences software maintainability behavior, which then impacts software maintenance cost. In order to prove these, the research had to address some questions as presented in the next section. Figure 5 shows the scales that were used to measure the various constructs and how the variables were operationalized.

As the research model depicts, there are external variables that exert influence on actual usage of a system. For this study, there are several of these external variables including product characteristics, characteristics of the development team, the software development process, organizational and
environmental factors. These variables were addressed on the measurement scale.

Product characteristics include product size, product complexity, the type of application being maintained, and the programming language used. Product size (with the symbol PSIZ) is the software size measured in thousand of source lines of code (KSLOC). Product complexity (with the symbol CPLX) is the degree of difficulty of the product development as perceived by the developers. The type of application (with the symbol ATYP) captures the nature of the program the developers are maintaining. These could range from Accounting/Finance applications to Signal Processing and Software Development Tools. Programming language (with the symbol LANG) captures the programming language(s) used by the developers to maintain the software system.

Development team characteristics include the team members’ capability, team communication, cohesiveness and experience. Team members’ capability (with the symbol PCAP) rates the software developer’s efficiency, thoroughness and ability to communicate. Team communication and cohesiveness (with the symbol TCOM) rate the project team’s ability to communicate effectively and perform as a unit. Team experience (with the
symbol AEXP) rates the team’s familiarity or prior experience with the type of software system they are maintaining.

Software development process guides the software developers’ activities, the activities of the software quality assurance personnel and those of the project management. One of the most popular software development process activities is the CMM for Software (SW-CMM)-oriented practices. These include requirements management, change management, configuration management, problem tracking and corrective action management, product software design, coding, unit testing, and integration testing, software quality assurance and peer reviews. The SW-CMM has 5 maturity levels with a total of 18 key process areas (KPAs) with specific recommendations on how to progress on the KPAs to attain higher levels of maturity, of which organizations can choose to practice a subset of the KPAs in all the levels. These KPAs are also identified on the measurement scale with symbols KPA1 through KPA18.

Organizational and environmental factors that affect software maintenance cost include an organization’s commitment to reduce maintenance cost (ORG1 thru ORG6 on the measurement scale), commitment to software engineering methods and tools that help reduce effort (symbol TOOL on the measurement scale),
and commitment to having realistic schedules (symbol SCHD on the measurement scale).
Figure 5. Scales for measuring the research model’s constructs

**Perceived Ease of Use (EOU)**
22. In my opinion, software process improvement activities should not be avoided (DAT15).
24. Following established software development standard (such as analysis, design, and coding standards) to develop software does not prevent me from being creative (DAT17).

**Perceived Usefulness (U)**
13. Learning about process improvement models such as CMM or CMMI is not a waste of time (DAT6).
15. I think documenting my design reduces the amount of time I spend on rework (DAT8).
16. Documenting code makes it possible to spend less time doing rework (DAT9).
17. I think the time I spend documenting my design and code leads to reducing the time spent on maintenance later on (DAT10).
19. I think using software improvement processes adds value to my project (DAT12).
25. Employing software best practices (such as risk management, reviews/inspections, requirements management, metrics-based scheduling, defect tracking, size estimating, and so on) improves the overall quality of software (DAT18).
26. Software process improvement activities would increase my overall productivity (DAT19).

**Attitude Toward Using (A)**
27. Documented software development standards should be encouraged as much as possible (DAT20).
28. In my opinion, software maintenance is one of the most important activities in the software life cycle (DAT21).
29. Software maintenance cost can be reduced dramatically if software engineering best practices are followed (DAT22).

**Behavioral Intentions To Use (BI)**
20. I always follow my team’s chosen methods for software development instead of choosing my own way (DAT13).
21. I would like to learn more about software process improvement activities (DAT14).
23. I see software development standards as something I will always use in my implementation of software (DAT16).

**Actual Use (AU)**
7. Of all activities I perform, software design is one of the activities I enjoy doing most (DAT1).
8. Of all activities I perform, software coding is one of the activities I enjoy doing most (DAT2).
9. Of all activities I perform, software testing is one of the activities I enjoy doing most (DAT3).
10. I am familiar with the intent of software process improvement models such as the CMM or CMMI (DAT4).
11. I am familiar with the content of software process improvement models such as the CMM or CMMI (DAT5).
13. I usually document requirements before design and coding begin (DAT7).
17. I usually make a written plan before I start to solve a problem (DAT11).

**Real Business Impact (RBI)**
45. How many months has your team spent on development and testing of the software you described above since it was first released to the customer (NMOS)?
46. What is the average number of hours per person, per month spent by your team on this project since it was first released to the customer (NHRS)?
47. How many software developers (including the project manager) are or were on your project team (TSIZ)?
Research Questions and Hypotheses

The questions of interest are as follows:

Research Question 1: Is there a correlation between a software developer’s attitude towards software improvement best practices and the cost of software maintenance to the organization?

Related to this research question, the following hypotheses are proposed:

$H_{01}$: There is no correlation between software developer’s attitude towards software process improvement practices and software developer’s perceived software maintenance cost.

$H_{a1}$: As the developer’s perceived usefulness ($U$) of software process improvement and the developer’s perceived ease of use ($EOU$) of software best practices increase ($U$, $EOU$), developer attitude ($A$) and the developer’s behavioral intentions ($BI$) to use those software best practices will increase.

$H_{b1}$: As developer attitude ($A$) towards software best practices and the developer’s perceived usefulness ($U$) of software process improvement increase ($A$, $U$), the behavioral intentions ($BI$) of the developer to use software best practices will increase.

$H_{c1}$: As the developer’s behavioral intentions ($BI$) to use software best practices increase, the tendency of the developer to use software best practices ($AU$) will increase.

$H_{d1}$: A correlation between software developer’s attitude towards software process improvement practices and perceived software maintenance cost ($RBI$) will be higher for those with high actual usage of the software best practices than those with low usage of the software best practices.
Impact of developer attitude

Research Question 2: Is there a relationship between the process maturity of a software development organization, as defined by the Software Engineering Institute’s Capability Maturity Model (SEI-CMM) and software maintenance cost?

Related to this question, the following hypotheses are proposed:

$H_0^2$: The correlation between Process Maturity and software maintenance cost will not be higher for organizations with mature processes than for organizations with less mature processes for developing software.

$H_a^2$: The correlation between Process Maturity and software maintenance cost will be higher for organizations with mature processes than for organizations with less mature processes for developing software.

$H_b^2$: There is a negative correlation between organizations that recognize software maintenance cost as areas to gain competitive advantage and software maintenance cost.

Research Question 3: What percentage of time do software developers spend doing software maintenance?

Related to this question, the following hypotheses are proposed:

$H_0^3$: Software developers do not spend a significant amount of their time doing software maintenance than they admit.

$H_a^3$: Software developers spend a significant amount of their time doing software maintenance than they admit.

Research Question 4: What accounts for the total maintenance workload for an organization, is it Adaptive, Corrective, Perfective or Preventive maintenance?
Related to this question, the following hypotheses are proposed:

$H_0$: Majority of an organization’s software maintenance effort is not spent doing Corrective and Perfective maintenance.

$H_a$: Majority of an organization’s software maintenance effort is spent doing Corrective and Perfective maintenance.

For the study, the variables of particular interest were the independent variables (developer’s attitude towards software activities that reduce software maintenance cost, percent time spent on maintenance, process maturity of an organization) and the dependent variable (software maintenance cost). The researcher’s expectation was that after gathering and analyzing the information, the data would show that a strong relationship exists between software developers’ attitudes toward software process improvement practices and software maintenance cost.

Significance of the Study

According to a 1995 study of approximately 175,000 projects conducted by Standish Group researchers, in the United States, more than $250 billion is spent each year on developing information technology (IT) applications. The same study found that the average cost of a development project for a large company is $2,322,000; $1,331,000 for a medium company; and $434,000 for a small company (Standish-Group, 1995). And since software maintenance has been determined to cost over fifty
percent of the overall cost of a software-based system, it means that more than $125 billion is spent each year in the US doing software maintenance.

Imagine if software development organizations save 10 to 20 percent of the cost of software maintenance. For corporate America, that could mean over $20,000 to $40,000 in savings per project for a small organization; $60,000 to $120,000 in savings per project for a medium-sized organization; and $116,000 to $232,000 savings per project for a large organization. For the software industry as a whole, it would mean a whopping $12.5 billion to $25 billion in savings each year.

A major contribution of this research would be the identification of software developer attitude as a factor and the quantification of its impact on software maintenance cost. If the study shows a significant relationship between software developers’ attitudes and software maintenance cost, software development organizations may consider incorporating more training aimed at addressing the behavioral aspects of their greatest asset -- people. Another major contribution of the study would be the development of a new theoretical model for understanding the role of user’s attitude influences in TAM on business goals. In addition, an understanding of the software developers’ attitudes and the variables that affect developer
attitudes will assist management in providing appropriate environment in which software developers can work to produce software that will cost less to maintain.

Definition of Terms

Attitude is a person’s learned predisposition to behave in a consistently positive or negative manner regarding a given object (Ajzen, 1988; Francis, 1988). An individual’s attitude is determined by his/her belief that an action can have either positive or negative outcome.

Behavior is the translation of intention or perceived behavioral control into action (Ajzen & Fishbein, 1980). A person’s intent to perform a behavior depends upon that person’s attitude and the subjective norm (the social pressure that is perceived by a person to perform or not perform a certain behavior).

Capability maturity model (CMM®) is a five-level evolutionary framework for ranking organizations from level 1 to level 5 of software maturity. The CMM® for software delineates the characteristics of a mature, capable software process. The model also describes the progression from an immature, unrepeatable software process to a mature, well-managed software

The *Person-month* definition used here is the same definition found in the COCOMO II model. It consists of 152 hours of working time. This has been found to be consistent with practical experience with the average monthly time off because of holidays, vacation, and sick leave. To convert an estimate in person-months to other units, use the following:
Person-hours: multiply by 152; Person-days: multiply by 19;
Person-years: divide by 12.

*Process maturity* is defined as the extent to which an organization’s approach to software development and maintenance can be measured in terms of maturity against a recognized framework. The recognized framework of choice for this study is the Capability Maturity Model proposed by the Software Engineering Institute (SEI). The SEI maturity framework consists of five levels ranging from level 1, the initial level to level 5, the optimizing level.

*Software* is a list of instructions that tell a computer what to do. It encompasses the entire set of procedures and routines associated with the operation of a computer system (Pressman, 1997). The term was coined to differentiate these instructions from the physical components (*hardware*) of a
computer system. There are two main categories of software: system software and application software. System software, which includes the operating system and all the utilities that enable the computer to function, is the program needed to operate the computer. Application software is a program that enables users to do real work using the computer (e.g. word processors, spreadsheets, and database management systems).

A software developer is a person who builds (designs, codes, tests, debugs, documents, and maintains) a computer program (Linberg, 1999). For the purposes of this research, this definition refers to computer programmers, systems analysts, software engineers, and information systems analysts.

Software development process is the process employed by software developers to translate user needs into a set of software requirements, which are in turn translated into design, which are then implemented in code, and the code is tested, documented, and validated for operational use (IEEE-Std-610.12, 1991). The term “user” refers to the requester of the system as well as the people that will actually use it.

Software engineering is that part of the computer science discipline concerned with developing large software applications. Software engineering covers not only the technical aspects of building software systems, but also management
issues, such as directing programming teams, scheduling, and budgeting (Pressman, 1993). In short, it is the application of systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software (IEEE-Std-610.12, 1991).

Software lifecycle is defined as the various phases a software product goes through from conception till it is no longer available for use. The software life cycle typically includes the following phases -- requirements analysis, design, implementation (code, unit test, integration), system testing (validation), installation, operation, maintenance, and retirement of the system. Every phase of the cycle makes use of different engineering methods, delivers something to the next phase, and there are feedback loops. The software development process tends to run iteratively through these phases rather than linearly. Several models (spiral, waterfall etc.) have been proposed to describe this process.

Software maintenance is the performance of tasks needed to keep a software system operational and responsive after it has been accepted and placed into production (Vallabhaneni, 1987). It is defined based on activities that are undertaken after a software-based system is released for use. The three main types of activities are corrective, adaptive, and perfective

Adaptive maintenance describes the effort initiated to modify or upgrade software to properly work in a new environment. The changes are normally beyond the control of the software maintainer and often consists of changes to the rules, laws, and regulations that affect the system; hardware configurations; data formats, file structures; and system software (Vallabhaneni, 1987).

Corrective maintenance describes those changes that occur due to actual errors discovered in a system after release for use. This type of maintenance consists of activities that include diagnosis and correction of one or more errors (Vallabhaneni, 1987).

Perfective maintenance is the changes, insertions, deletions, extensions, and enhancements made to a system to meet the evolving needs of the system. As the software is used and become successful, suggestions for new capabilities and enhancements are received from users. To accommodate these requests, perfective maintenance is performed. Perfective
maintenance accounts for the majority of all effort expended on software maintenance (Pressman, 1987; Vallabhaneni, 1987).

*Preventive maintenance* occurs when software development organizations change their software to improve future maintainability of the software to provide a better basis for future enhancements (Pressman, 1987). Preventive maintenance is still not widely practiced in the software industry.

*Software maintenance cost* is the cost associated with performing tasks needed to keep a software system operational and responsive after it has been accepted and placed into production (Vallabhaneni, 1987).

A *software organization* is defined as any software development group with twenty or more people with the responsibility to create software products for commercialization or integration into a larger system.

Assumptions and Limitations

It was assumed that the sample chosen for the study would be interested in the accuracy of the results. It was also assumed that the study participants would be objective enough to complete the survey questionnaire truthfully. It was further assumed that the sample of software developers from the
corporation would be representative of the population of software developers in the corporation.

There were limitations that may or may not have affected the outcome of the study. The statistical generalizability of the study is limited to the population of software developers and software project managers of the corporation that was being studied. However, the researcher suggests that generalization can be extended to a much larger population, the entire population of software developers in the company.

There was also a time limitation for data collection. The researcher had only a window of five weeks to collect data. Other assumptions and limitations related to the study are discussed in Chapter 3.

Organization of the remainder of the study

The remaining two chapters will present the literature reviews and the methodology respectively. Chapter Two will discuss the pertinent literature reviews that have addressed the issue of software maintenance cost and its impact on software organizations. It will provide a rationale for the inclusion of the sections of the literature review that are in this chapter, relating the concepts, and connecting the focus of the study to
the larger concern. Chapter Three will present the research methodology that has been chosen for the study.
CHAPTER 2   LITERATURE REVIEW

Introduction and Organizational Structure of the Chapter

This chapter presents an abbreviated review of the pertinent literature as related to key topics of the proposed study. The chapter follows a progression from a broad focus on behavioral theory to a very specific focus on software developers’ attitudes about software maintenance. Before exploring specific literature on software developer attitude about software maintenance, this chapter will present a critical examination of the strengths and weaknesses of the related research designs, methodologies and theories and then provide a justification for the study.

General Theoretical Concepts on Behavior

Ajzen and Fishbein’s Theory of Reasoned Action (TRA) that was developed in the early 1970s is often one of the most cited theories in the literature for studying human attitudes and behavior and for developing appropriate interventions. The TRA is a model of the psychological processes that link individual beliefs, attitudes, intentions, and behavior (Fishbein & Ajzen, 1975; Fishbein et al., 1994). The TRA is composed of attitudinal, social influence, and intention variables that predict behavior. The theory asserts that an individual’s intention to perform a behavior is determined by the individual’s attitude toward the behavior and the subjective norm held by the individual.
This model supports a linear process in which the individual's actual behavior is ultimately affected by changes in an individual's behavioral and normative beliefs.

Another theory on behavior that is often cited is Ajzen’s Theory of Planned Behavior (TPB), which was added to the existing model of reasoned action to address the inadequacies of the TRA (Ajzen, 1991). According to the theory, there are three things that guide a software developer’s action: (a) the software developer’s beliefs about the likely outcome of following software engineering best practices to build software (behavioral beliefs), (b) the software developer’s perception of the expectations of others around him/her regarding software engineering best practices and the motivation to comply with those expectations (Normative beliefs), and (c) the degree to which the software developer believes that he/she can control the use of software best practices for building software (Control beliefs) (Ajzen, 1991). When taken separately, behavioral beliefs result in situations where an attitude toward the behavior is either favorable or unfavorable, normative beliefs produce perceived social pressure or subjective norm, and control beliefs result in perceived behavioral control. When combined, a person’s attitude toward a behavior, the person’s subjective norm, and the individual’s perception of behavioral
control lead to the formation of a behavioral intention (Ajzen, 1991).

The other theory on attitude and behavior that has been made popular in the last decade is Davis’ Technology Acceptance Model (Davis, 1986). The Technology Acceptance Model (TAM) uses TRA as a theoretical basis to specify causal linkages between the constructs: (1) Perceived Usefulness (U), Perceived Ease of Use (EOU), user’s attitude (A), behavioral intentions (BI) and actual computer best practice usage behavior. When the model is extended to predict the effect of attitude on a business goal, the behavioral and attitudinal processes determine the real business impact. This extension of TAM will attempt to enrich TAM's ability to explain technology acceptance and predict business impact.

Software Maintenance Process

Software maintenance process is essentially related to software development process, as some of the tasks are fundamental for both processes. Information from the development process is always needed in the maintenance process. However, some have suggested that software maintenance has a separate process of its own (Khan, Lo, Skramstad, & Khan, 2000).
Even though there are activities between the two processes that intersect as shown in Figure 6, there are three distinct, inter-related and interdependent components -- people involved, required knowledge, and supporting tasks -- of the software maintenance process (Khan et al., 2000).

The people involved are mainly the users of the system, the maintenance staff, and the original development staff. The required knowledge includes knowledge about the operational environment, design and program structure, as well as the domain of the software. The supporting tasks consist of maintenance requirements analysis, determination, program comprehension, localization and impact analysis, generating test data, and implementation plan (Khan et al., 2000).
As noted before, software maintenance process is essentially related to software development process because as some of the tasks are fundamental for both processes. For that reason, many of the lifecycle models described below do not include software maintenance as a separate activity.

The Classic Life-cycle "Waterfall" Model

Source: (NASA, 1999)

Figure 7. The Classic Waterfall Model
The classic life-cycle model (also known as “the waterfall model”) defined by Winston Royce (Royce, 1970) is still the most commonly used model for software development today, especially on large government systems, particularly by the Department of Defense (DOD). The waterfall model as shown in Figure 7 is an engineering method that is linear and sequential. The method has distinct goals for each phase of development. It assumes a step-wise approach whereby development progresses through different phases and each phase produces some outputs that become input into the next phase. There are usually five to six stages in this model of software development depending on how activities are broken down:

1. Concept. This phase is used to refine the initial idea for a software product. Specifications of the product are identified in this phase. The initial requirements are gathered from potential users, analyzed and documented. A prototype of the product is developed during this phase.

2. Requirements definition. This phase is used to gather and define the requirements of the system to be built. These are usually the features and services the new system will provide, its constraints and the goals of the software. Once these are established they have to be defined in such a way that they are usable in the next stage. Sometimes, a feasibility study is used to flesh out the requirements. The feasibility study might include questions like “should we buy or build the software?,” “what alternatives do we have?,” etc. Some people even call this phase the “conception” of a software product phase. This often the beginning of the software life cycle.
3. Systems Analysis & Design. In this phase the established requirements defined in the requirements definition phase are broken down into software and hardware requirements. The software requirements are analyzed. This phase also focuses on high level design (what programs are needed and how are they going to interact), low-level design (how the individual programs are going to work), interface design (what are the interfaces going to look like) and data design (what data will be needed).

4. Implementation (code and unit testing). In this phase, the design is translated into computer programs (code). Each program is called a unit, and unit testing is the verification that every unit meets its specification.

5. Integration and System testing. This phase focuses on combining all the units into an integrated system and the whole system is tested. When the integrated programs are successfully tested the software product is considered finished.

6. Operation and Maintenance. This phase is the post customer delivery stage, which most software products include in their development cycle. It involves correcting errors that have gone undetected during development and testing. Improvements and other forms of support are included in this phase. The phase is part of the life cycle of a software product, but is often forgotten or ignored during planning and estimating. It is also the most expensive phase of the software life cycle.

The Waterfall model is perhaps the most rigid of all software development models. Since it is documentation driven, changes usually are very hard to implement. The waterfall model is however, easier to manage than other models. It handles product and process control better than the evolutionary model or prototyping model. And even though the waterfall is a slower development model, it has fewer problems with debugging and
integration than the evolutionary (spiral) model or prototype model. However, unlike the spiral model or prototyping model, it does not handle user interfaces very well due to the difficulty in adding changes. The waterfall model is best suited for short-term projects. Although other models have been developed which are more usable for many modern projects, they still use the Waterfall model as a part of the model. And because the Waterfall model is designed towards cost-effective development of single products it can be used for single projects that are cost or time restricted because of the ease of managing it

The Spiral Model

The spiral model was developed to take advantage of the best features of both the classic waterfall model and prototyping, while adding risk analysis as a new element that is missing in both the waterfall model and the prototyping paradigms. The model is depicted by the spiral (see Figure 8), and is defined by four major activities represented by four quadrants that are labeled as planning, risk analysis, engineering, and customer evaluation (Pressman, 1993). Planning is for determining objectives, alternatives, and constraints. Risk analysis is for analyzing the alternatives, identification and/or resolution of risks. The engineering quadrant is for
developing the next level product while the customer evaluation quadrant is used to assess the results of engineering. There are two main features that distinguish the spiral from other models. One is a cyclic approach for evolving a system’s degree of definition and implementation while decreasing its degree of risk. The other is a set of anchor point milestones for ensuring stakeholder commitment to feasible and mutually satisfactory system solutions (Boehm & Hansen, 2000).

The cyclic nature of the model means that instead of building a completed product in one step, multiple cycles are performed with each cycle taking steps to reduce the most significant remaining risks. Starting at the center and working outward, every iteration around the spiral progresses the software system to a more complete version. During the first cycle around the spiral, the project’s objectives, alternatives, and constraints are defined. Risks are identified and analyzed. Based on the risk analysis, prototyping might be used to assist the developer and customer clarify uncertainties. If possible, simulations and other models may be used to define the problem and refine the requirements. The customer is then given the opportunity to evaluate the engineering work and makes suggestions for modification. Based on the input from the customer, the next phase of planning and risk analysis occurs.
The culmination of risk analysis results in a "go/no-go" decision. If it is determined that the risks are too great, the project can be terminated. However, in most cases, flow around a spiral path continues, with each path moving developers outward toward a more complete model of the system, and eventually, to the operational system itself. Each cycle around the spiral also requires engineering as the engineering quadrant shows. Such engineering activities can be accomplished using either the classic waterfall life-cycle or prototyping methods.

One of the difficulties with the original spiral model was its lack of intermediate milestones to serve as commitment points and progress checkpoints. Later updates to the model have remedied the problem by adding a set of anchor point milestones to the model. These are the Life Cycle Objectives (LCO), Life Cycle Architecture (LCA), and the Initial Operating Capability (IOC), which is the first released version of a software-based system (Boehm & Hansen, 2000). Each anchor point milestone is a specific combination of artifacts and conditions that must be attained at some point. These milestones propel the project toward completion and offer a means to compare progress between one project and another.
The LOC deals with the question of what the system should accomplish and the focus of LOC review is to ensure that at least one architecture choice is viable from a business perspective. As the piece that deals with what the structure of the system should be, its focus of review is to commit to a single detailed definition of the project. The LCA milestone serves a very important role. Part of its pass/fail criteria is
for project stakeholders to hold up projects attempting to proceed into evolutionary or incremental development without a life-cycle architecture (Boehm & Hansen, 2000). Each milestone serves as a stakeholder commitment point. For example, at LCO, the stakeholders commit to support building architecture, at LCA they commit to support initial deployment and at IOC they commit to support operations. These anchor point milestones collectively help avoid analysis paralysis, unrealistic expectations, requirements creep, architectural drift, COTS (Commercial off-the-shelf) shortfalls and incompatibilities, unsustainable architectures, and useless systems (Boehm & Hansen, 2000).

Maintenance Assistance Capability for Software

The Maintenance Assistance Capability for Software (MACS) (Desclaux & Ribault, 1991) focuses its activities on the process of understanding the existing application program. Its main phases are reverse engineering, modification management, and ripple effect analysis. Not only is it important to understand the facts about an application, but also necessary to understand the design decisions and rationale used. The goal of MACS is to provide a customizable software maintenance assistance system for both existing and in-progress software as well as a
supporting methodology (Desclaux & Ribault, 1991; Georges, 1992).

**The IEEE Standard 1219-1998**

The IEEE Standard 1219-1998, which redefined IEEE Standard 1219-1992, describes a process for managing and executing software maintenance activities. It addresses issues and problems of software maintenance and required actions and procedures, which can help organizations meet the growing demands of maintaining existing systems. It discusses the tools and techniques that can be used to improve the control of software maintenance activities. Managing the maintenance process and maintenance specific definitions will be included in the standard. However, in the IEEE standard, a clear interaction between the development and maintenance processes is lacking with respect to information from the software development process to maintenance, and the activities related to program understanding is not explicitly covered. The strength of this standard is the inclusion of the supporting issues such as planning, verification and validation, risk assessment, quality assurance, configuration management, and software metrics.
Impact of developer attitude

Review of Pertinent Literature

Historical Context

In the 1980s and the early 1990s, the software industry was declared to be in a crisis or as Robert S. Pressman put it, it had a “chronic affliction” (Pressman, 1987). The “software crisis” stemmed from a lack of disciplined approach for building software products. The result was massive software project failures, budget overruns, late deliveries and poor quality. Tools, structured methodologies and new higher level programming languages were developed in response to the crisis. Process improvement methods were developed and a host of other tools were introduced to address the problem. Despite all these remedies, the cost of developing software products continued to skyrocket and project failures are still rampant. According to Watts S. Humphrey, a careful examination of failed projects shows that they often fail for non-technical reasons (Humphrey, 1998). The most common problems are poor scheduling and planning or uncontrolled requirements. Poorly run projects also often lose control of changes or fail to use even rudimentary quality practices.

It was because of the "software crisis" that the U.S. Department of Defense (DoD) established the Software Engineering
Institute (SEI) in the first place. Under the auspices of the DoD, the SEI started its process improvement work in 1986 under the leadership of Watts S. Humphrey. After it was determined that poor management practices were the most frequent cause of software project failure, SEI first attacked project management problems. Usually, when the project is poorly managed, it is hard to improve anything else. The irony is that regardless of how logical the improvement is and regardless of growing evidence of its benefits, organizations do not launch successful process improvement programs until they have a compelling business reason to do so. And one of the most effective improvement motivators is customer pressure.

The U.S. Air Force commissioned the Software Engineering Institute to devise an improved method to select software vendors. When the SEI responded with the Capability Maturity Model (CMM®), the defense industry then had a compelling business reason to improve their software processes. The CMM® process maturity-level framework and related evaluation system help organizations understand their capabilities. As the CMM® became more widely used, the business logic for improvement also evolved. Today, many software organizations desire to establish
and maintain a strong competitive position as quality suppliers of software-intensive systems.

Despite all the improvements that have been made in the areas of tools, techniques, processes, and project management, one activity -- software maintenance -- seems to have taken a back seat. Although software maintenance consumes more than fifty percent of the overall cost of a software-based system, many software organizations have simply ignored it. It continues to receive the silent treatment and seen as an after thought instead of being given the special attention it deserves since it costs more. There must be good reason(s) why this is so but the reasons are either not known or the known reasons are outdated. There have been many studies on the impact of maintenance on the overall cost of software products as well as on the factors that influence maintenance cost, but these studies are more than twenty years old. In the software industry where technology has an average shelf life of eighteen months, depending on the same factors that were true more than twenty years ago might be part of the reason why organizations seem not to care about maintenance cost. Therefore, there should be a search for other factors, particularly, in the behavioral attitudes of the software professionals that fit with today’s
technologies and the way things are done. That will be the focus of this study.

Assessment of Previous Studies

Many studies have been conducted about software maintenance and a few studies have been published about the behavioral aspects of software personnel towards maintenance work (Basili et al., 1996; Basili & Robert W. Reiter, 1979; Morris & Dillon, 1997; Slaughter & Banker, 1996; Tan & Gable, 1998; Yeh & Jeng, 2002). However, nothing has been done with respect to using the Technology Acceptance Model to explain software developers’ attitudes towards software development activities that reduce maintenance costs.

A study conducted by Morris and Dillon (1997) explored the efficacy of the technology acceptance model in predicting software utilization among a set of potential users of the software. The study included all students enrolled in a beginning computer skills course, resulting in a sample of 101 potential users of Netscape. After subjects with prior experience using the World Wide Web or Netscape were eliminated, a final sample of 76 users were involved in the experiment. The study identified a number of interesting results. First, perceived usefulness and perceived ease of use were shown to
have significant positive influences on attitude toward using the application software. Second, both perceived usefulness and attitude toward using were also shown to exhibit significant positive influences on behavioral intentions to use the software application. Third, behavioral intentions to use were shown to have significant positive influence on actual use of the system. This was a well-designed study, however, the study was more about software usage as opposed to application of methods or best practices for developing software.

Another study by Slaughter and Banker (1996) involved two studies conducted in two organizations - a national mass merchandising retailer and a bank. One study examined the effects of software complexity on maintenance project effort, controlling for such factors as project team experience, software functionality modified, application size and quality, and other project factors. The other study attempted to connect maintenance outcomes to software practices by assessing the effect of software development tools and techniques on the complexity of the software application. The study provided a number of interesting results about software development practices and maintenance effort. Results of the study provided evidence that software development practices and maintenance effort are linked via their effects on software complexity.
Other findings of the study show that use of a software code generator is associated with increased software maintenance effort. Although the premise of the Slaughter and Banker (1996) study -- a large portion of maintenance costs result from poor software development practices -- is similar to that of the proposed research, the study focused on the software complexity aspect as opposed to the software developer's attitude towards software development practices.

In another study, Tan and Gable (1998) assessed the attitudes of software maintenance personnel towards maintenance work. The study's premise was that the attitudes of maintenance staff towards their work can have adverse effect on their performance and the quality of their work. The researchers compared the views of maintenance personnel in Singapore who either were actively involved in maintaining application systems or directly supervised the maintenance process with the views of maintenance managers. The results were in turn compared against findings from prior studies conducted in the United States. The study found that Singapore maintenance personnel had more adverse view of maintenance than their U.S. counterparts. Particularly, Singaporean maintenance personnel neither find their job challenging nor responsible. Their belief is that
their development colleagues are better paid with better chances of promotion.

Yeh and Jeng (2002) conducted a study that evaluated the influence of departmentalization and organizational position on software maintenance. The study applied quantitative empirical methods to investigate the influence of departmentalization on fulfillment opportunity, time allocations of activities, problem occurrences and management process in software maintenance. Results from the study showed that separate organizations demonstrate specialization in software maintenance, however, managers’ attitudes tend to aggravate the potential status difference for such organizations.

Basili and Reiter (1979) conducted one of the early studies about the effect of human factors in software development. Their investigation focused on the effects of two factors -- the size of the programming team deployed and the degree of methodological discipline employed. The study’s findings revealed some interesting programming aspects for which statistically significant differences do exist among development approaches of development teams. According to the study, disciplined teams required fewer computer runs and made fewer errors during software development than either individual programmers or ad hoc teams. The study’s results indicated that
disciplined methodology increased software reliability beyond that achieved by individual programmers or programming teams that employed ad hoc approaches.

All of these studies and reports have dealt in one way or another with the importance and benefits of software process improvement and its effect on productivity, quality, and maintainability of software. However, no study has dealt specifically with the behavioral impact of software developers on software maintenance cost. Hence, this research is an attempt to do just that, and that is to determine the relationship that exists between software developer’s attitudes towards process improvement activities for making software more maintainable and software maintenance cost.

Justification of the Topic

Reducing the cost of software maintenance benefits not only the software organization, but also the consumer. Some of the benefits of lower maintenance cost are:

1. Pricing below the competition. By controlling the conditions that lead to higher maintenance cost, a software development organization can spend less in terms of maintenance cost. This in turn enables the business to provide genuine value to customers by selling its software product at prices well below those of its competitors. Ultimately, the consumer wins.
2. Easy profit-creation with a few sales. It is a fact that software has to be maintained and changed from time to time due to the dynamic nature of business and life in general. It is also true that software maintenance has become a huge area in the computer industry that consumes more than its share of the overall cost. If a business or an organization spends less in expenses such as software maintenance, it makes it easier for the business to report profit. Even with very few sales, the business can easily record profits because the cost of doing business is very low.

3. Fewer upgrades for the customer. Fewer maintenance of a software-based system, which means fewer resources spent on the product translates to fewer upgrades for the customer. These upgrades often cause disruption in the customers’ operations, which could mean tens of thousands in loss of revenue.

4. Extended market life of the product. The software could be upgraded and maintained indefinitely. New, state-of-the-art technology can be applied to the software system every few years giving the system exciting new capabilities with each upgrade.

Despite mounting evidence of high cost of the software maintenance phase, many companies are still not doing enough to lower software maintenance cost. This is because they either don’t see it as an avenue to gain competitive advantage, or they don’t understand all the factors, particularly the human factors that impact software maintenance cost. This is why the main objective of this research will be to determine the effects of software developers’ attitudes on software maintenance cost. The outcome of the research will be a clear understanding of the
relationship that behavioral factors – such as software developer’s attitude – has with software maintenance cost.

In summary, this chapter presented the pertinent literature reviews that have addressed the issue of software maintenance cost and their impacts on software organizations. It also provided a rationale for the inclusion of the sections of the literature review that are in this chapter, related the concepts, and connected the focus of the study to the larger concern.
CHAPTER 3 METHODOLOGY

Introduction

This methodology chapter describes the research design and data collection methods that were used to investigate the relationship between software developers’ attitudes towards software process improvement activities and software maintenance cost. The research method used was the descriptive correlational study. A Web-based questionnaire was administered to a sample of software developers and software project managers in software development organizations of a Fortune 500, Multi-national Corporation in the United States of America. The variables that were studied include (a) the software developer’s attitude, (b) amount of time spent on software maintenance, (c) resources spent on software maintenance (i.e. cost of software maintenance or software maintenance effort), (d) level of process maturity, and (e) total maintenance workload. The respondents in this study were required to describe their application rather than an arbitrarily selected application. A further distinction of this study was the surveying of software development managers, who are often in a better position to answer questions concerning specific issues about software development activities and management. This chapter is divided into the following sections: description of methodology selected, design of the study, sample and population, instrumentation, data collection procedures, and data analysis.
Description of the Methodology

The chosen method for this research was the descriptive correlational study. The descriptive correlational method was used because the researcher was interested primarily in describing relationships among variables without necessarily seeking to establish causality. Descriptive studies tend to be simpler and easier to conduct than experimental or quasi-experimental studies. In a descriptive study, no attempt is made to change behavior or conditions -- the researcher simply measures things as they are. The aim of a descriptive study is often to make inferences about the population, based on descriptive statistics such as Means and Proportions (Crowl, 1993). In general, descriptive research designs can be classified as either qualitative or quantitative (Gersten, 2001). Quantitative descriptive designs yield numerical data on observable behaviors of samples, which are then subjected to statistical analysis (Crowl, 1993). Quantitative descriptive designs include but are not limited to surveys, ex-post facto designs, and case study. This research utilizes the survey design using self-reported questionnaires.
Design of the Study

Target Population

The target population is all software developers and software development managers with responsibility for product commercialization working in a Fortune 500 company. This Fortune 500 Company has an employee base of 73,000 worldwide with a U.S. employee population of approximately 28,600 people, of which approximately 21,700 are in Rochester, New York area. Its U.S. operations have about six divisions and each division has several software development organizations. Many of these organizations are engaged in the development of software. Based on the definition of a software organization, the researcher estimated that there are about 10-15 such organizations in the company in Rochester, with approximately 500-800 software developers and software managers.

A Web-based questionnaire was used rather than individual interviews for the purpose of increasing sample size. In addition, the questionnaire was less time consuming for the researcher as well as for the study participants. The researcher will analyze the data to determine if there is a correlation between software developer attitude and software maintenance cost. The researcher understands that even a strong correlation
does not necessarily mean that a cause-effect relationship exists.

The Sample

The sample included software developers and software development managers working in a Fortune 500 corporation located in the Rochester New York metropolitan area. The participants are experienced software professionals with at least 2 years of experience in the software development field. All software development organizations of the corporation in the Rochester area were included in the survey. These organizations have people from both genders and are multicultural.

Sampling Procedures and Sample Size

The researcher sent Web-based questionnaires to the entire population of software developers and software managers in the corporation in the Rochester, New York area. Each completed questionnaire was screened to eliminate those cases where the respondent failed to meet the criteria. The criteria were:

(a) the questionnaire must be properly filled out and
(b) the respondent must develop software or manage software project in his or her job.

In order to determine sample size for this study, a power analysis was carried out a priori, meaning that power is
determined during the design stage of the study. Given the three factors alpha, sample size and effect size, a fourth variable called beta can be calculated. Alpha ($\alpha$) is the probability of a type I error (i.e. rejecting a correct null hypothesis), while $1-\alpha$ is the probability of correctly accepting the null hypothesis. Beta ($\beta$) is the probability of a type II error (i.e. incorrectly rejecting the null hypothesis). The probability of correctly rejecting the null hypothesis is equal to $1 - \beta$, which is called power. The power of a statistical test is its ability to detect real differences among variables (Gillis & Jackson, 2002).

As there are many software programs today that can be used to do power analysis, this study used a power analysis program called Power and Precision version 2 by Biostat to determine sample size. One goal of the proposed study was to test the null hypothesis that there is no correlation between software developer’s attitude towards software process improvement practices and software maintenance cost. The criterion for significance (alpha) was set at 0.05. The test is 2-tailed, which means that an effect in either direction would be interpreted. As shown in Figure 9, with a proposed sample size of 85 the study would have power of 81.7% to yield a
statistically significant result. A sample size of 114 would have power of 91.4% to yield a statistically significant result.

This computation assumes that the correlation in the population is 0.30. The observed value will be tested against a theoretical value (constant) of 0.00. This effect was selected as the smallest effect that would be important to detect, in the sense that any smaller effect would not be of substantive significance. It is also assumed that this effect size is reasonable, in the sense that an effect of this magnitude could be anticipated in this field of research.

The planned sample size of 85 or 114 was larger than necessary to increase the likelihood that the sample would be representative of the target population of software developers and software managers. The sample size was also large enough to increase the likelihood that the five ranges for years at current position (0-2, 3-5, 6-8, 9-11, 12 and over) and the five ranges for years of experience in the software profession (0-2, 3-5, 6-8, 9-11, 12 and over) were represented.
Sampling Bias

Sampling bias (or the tendency to exclude some members of the sampling population and over represent others) is among the problems that face researchers. It may arise from a variety of sources including the failure to adhere to random sampling procedures, faulty measuring instruments (in terms of the specific questions used in a questionnaire) and omission of some subgroups of the population from the sampling frame and ultimately from the sample. In order to minimize such biases, the researcher made sure the sample for the pilot study was selected at random, using a simple random process. Although the participants of the pilot study all came from one software
development organization, they all worked on different projects that spanned three divisions of the corporation. So there experiences and project perspectives differ.

Selecting the actual study sample is another potential source of bias. However, it was the intention of the researcher to ensure that no subgroup of the target population was omitted. This was achieved by using a cluster sampling technique. Each software development organization was considered a cluster, and the entire population of each cluster was included in the sample.

Some bias is inevitable, especially when surveys of human populations are conducted, mainly because of such issues as non-response and the fact that it is impossible to design a survey questionnaire that is completely free of bias. However, it was the intention of the researcher to take measures to minimize bias in the survey. Examples of what could happen include refusals by subjects to participate, refusals by participants to answer certain questions, misunderstanding of questions by some of the respondents leading to incorrect answers. These sources of bias were minimized as follows:

1. Subject refusals to participate or to answer specific questions was reduced through various efforts of persistence, such as use of incentives and follow-up to remind the survey participants to respond.
2. A review by a panel of experts, pilot testing, and analysis of the pilot-test results minimized bias from question misunderstanding through careful design of survey questions. Feedback from the panel of experts revealed some problems with understanding of questions and those questions were re-written to make them clearer.

3. Based on recommendations by one of the experts, the researcher decided to conduct the survey using computers, specifically, the Internet. The reason for this was to minimize interview bias as well as improve response rate. Conducting the survey on the Internet eliminated interview bias as well as social class barriers that could have emanated from the interviewer.

4. Social interactions between participants and the researcher could influence the responses of the participant. This is one of the reasons the researcher chose the survey questionnaire as the instrument of choice. To further minimize or eliminate this type of bias, the researcher did not collect names or any other information that could compromise participant confidentiality.

Instrumentation

For lack of existing instrument that could be used for this particular study, the researcher designed, pilot-tested and validated an instrument for software maintenance and developer attitude survey (SMADAS). The questionnaire made use of the Likert Scale with multiple means of assessing the effect of developer attitude on software maintenance cost. The questionnaire included four sections: demographics, developer attitude, organization factor, and project/product factor. Refer to the appendix for additional information about the instrument.

The demographics section includes gender, type of work (software developer or software management), years in current
position, years of experience and level of education. Table 1 shows the characteristics of the participants that were sampled for the pilot test.

Table 1: Characteristics of the Pilot Test sample

<table>
<thead>
<tr>
<th>Years in current role</th>
<th>Percent</th>
<th>Years in the software profession</th>
<th>Percent</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-2 years</td>
<td>14%</td>
<td>0-2 years</td>
<td>0%</td>
</tr>
<tr>
<td>3-5 years</td>
<td>43%</td>
<td>3-5 years</td>
<td>14%</td>
</tr>
<tr>
<td>6-8 years</td>
<td>29%</td>
<td>6-8 years</td>
<td>29%</td>
</tr>
<tr>
<td>9-11 years</td>
<td>0%</td>
<td>9-11 years</td>
<td>14%</td>
</tr>
<tr>
<td>12+ years</td>
<td>14%</td>
<td>12+ years</td>
<td>43%</td>
</tr>
</tbody>
</table>

Note: Education: College degree (100%)

For the developer attitude section, a Likert-scale with questions as shown in Tables 2 through 4 was developed. Each question can be answered by selecting Strongly Disagree (SD), Disagree (D), Neutral (N), Agree (A), and Strongly Agree (SA). Points are assigned to each answer as follows: SA = 5, A = 4, N = 3, D = 2, SD = 1. The total possible score for the developer attitude portion is 110. Reliability of this scale was determined by a pilot test, in which a Cronbach’s alpha reliability coefficient of 0.92 was calculated.

Similarly, the organization factor section has Likert-scale questions as shown in Table 5. The questions are answered by selecting Strongly Disagree (SD=1), Disagree (D=2), Neutral
Impact of developer attitude

(N=3), Agree (A=4), and Strongly Agree (SA=5). The total possible score for the organization factor portion is 35. The same pilot test was used to compute a Cronbach’s alpha reliability coefficient of 0.93 for the organization factor scale. Additional questions in this section include whether or not the organization follows the Software Engineering Institute’s Capability Maturity Model for software (SW-CMM) process improvement, the organization’s overall process maturity level.
<table>
<thead>
<tr>
<th>SMADAS Developer Attitude Scale Question</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>8. Of all activities I perform, software design is one of the activities I enjoy doing most (DAT1).</td>
<td>Determination of job satisfaction.</td>
</tr>
<tr>
<td>9. Of all activities I perform, software-coding is one of the activities I enjoy doing most (DAT2).</td>
<td>Determination of job satisfaction.</td>
</tr>
<tr>
<td>10 Of all activities I perform, software-testing is one of the activities I enjoy doing most (DAT3).</td>
<td>Determination of job satisfaction.</td>
</tr>
<tr>
<td>11 I am familiar with the intent of software process improvement models such as the CMM or CMMI (DAT4).</td>
<td>Establishing an individual’s knowledge of software improvement processes.</td>
</tr>
<tr>
<td>12 I am familiar with the content of software process improvement models such as the CMM or CMMI (DAT5).</td>
<td>Establishing an individual’s knowledge of software improvement processes.</td>
</tr>
<tr>
<td>13 Learning about process improvement models such as CMM or CMMI is not a waste of time (DAT6).</td>
<td>Determination of an individual’s Perceived usefulness for SW improvement best practices.</td>
</tr>
<tr>
<td>14 I usually document requirements before design and coding begin (DAT7).</td>
<td>Determination of an individual’s actual use of SW design and coding best practices.</td>
</tr>
</tbody>
</table>
### Table 3: Developer Attitude Scale Questions and Purpose

<table>
<thead>
<tr>
<th></th>
<th>Question</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>15</td>
<td>I think documenting my design reduces the amount of time I spend on rework (DAT8).</td>
<td>Determination of an individual’s Perceived usefulness for SW design best practices.</td>
</tr>
<tr>
<td>16</td>
<td>Documenting code makes it possible to spend less time doing rework (DAT9).</td>
<td>Determination of an individual’s Perceived usefulness for SW coding best practices.</td>
</tr>
<tr>
<td>17</td>
<td>I think the time I spend documenting my design and code leads to reducing the time spent on maintenance later on (DAT10).</td>
<td>Determination of an individual’s Perceived usefulness for best practices for SW design and code.</td>
</tr>
<tr>
<td>18</td>
<td>I usually make a written plan before I start to solve a problem (DAT11).</td>
<td>Supporting an individual’s process-oriented behavior.</td>
</tr>
<tr>
<td>19</td>
<td>I think using software improvement processes adds value to my project (DAT12).</td>
<td>Perceived usefulness for SW improvement best practices.</td>
</tr>
<tr>
<td>20</td>
<td>I always follow my team’s chosen methods for software development instead of choosing my own way (DAT13).</td>
<td>Supporting an individual’s tendency not to follow procedures.</td>
</tr>
<tr>
<td>21</td>
<td>I would like to learn more about software process improvement activities (DAT14).</td>
<td>Establishing an individual’s intent to practice</td>
</tr>
<tr>
<td>22</td>
<td>In my opinion, software process improvement activities should not be avoided (DAT15).</td>
<td>Determination of an individual’s perceived ease of use of a software process or best practice.</td>
</tr>
<tr>
<td></td>
<td>Question</td>
<td>Purpose</td>
</tr>
<tr>
<td>---</td>
<td>--------------------------------------------------------------------------</td>
<td>---------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>23</td>
<td>I see software development standards as something I will always use in my implementation of software (DAT16).</td>
<td>Determination of an individual’s willingness to use software standards to develop software.</td>
</tr>
<tr>
<td>24</td>
<td>Following established software development standard (such as analysis, design, and coding standards) to develop software does not prevent me from being creative (DAT17).</td>
<td>Determination of an individual’s perceived ease of use for software best practices.</td>
</tr>
<tr>
<td>25</td>
<td>Employing software best practices (such as risk management, reviews/inspections, requirements management, metrics-based scheduling, defect-tracking, size-estimating, and so on) improves the overall quality of software (DAT18).</td>
<td>Determination of an individual’s perceived usefulness for software best practices.</td>
</tr>
<tr>
<td>26</td>
<td>Software process improvement activities would increase my overall productivity (DAT19).</td>
<td>Perceived Usefulness of Software Process improvement</td>
</tr>
<tr>
<td>27</td>
<td>Documented software development standards should be encouraged as much as possible (DAT20).</td>
<td>Supporting an individual’s tendency not to follow development standards.</td>
</tr>
<tr>
<td>28</td>
<td>In my opinion, software maintenance is one of the most important activities in the software life cycle (DAT21).</td>
<td>Establishing an individual’s attitude toward doing SW Maintenance activities</td>
</tr>
<tr>
<td>29</td>
<td>Software maintenance cost can be reduced dramatically if software engineering best practices are followed (DAT22).</td>
<td>Establishing an individual’s attitude toward doing SW Maintenance activities</td>
</tr>
</tbody>
</table>
### Table 5: Organization Factor Scale Questions and Purpose

<table>
<thead>
<tr>
<th>SMADAS Organization Factor Scale Question</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>30 My organization maintains a set of standard software processes for developing software (ORG1).</td>
<td>Determining an individual’s knowledge of the existence of an organization’s processes.</td>
</tr>
<tr>
<td>31 My organization encourages me to follow its standard software process to develop software (ORG2).</td>
<td>Establishing an individual’s perceived support for software process improvement.</td>
</tr>
<tr>
<td>32 My organization provides adequate funding to make software process improvement happen (ORG3).</td>
<td>Establishing an individual’s perceived support by the organization for software process improvement.</td>
</tr>
<tr>
<td>33 My organization has a group or individual(s) responsible for the organization’s software process activities (ORG4).</td>
<td>Establishing an individual’s perceived support by the organization for software process improvement.</td>
</tr>
<tr>
<td>34 In my opinion, my organization recognizes the need to reduce software maintenance cost (ORG5).</td>
<td>Establishing an individual’s perceived need by the organization to reduce software maintenance cost.</td>
</tr>
<tr>
<td>35 My organization sees software maintenance cost reduction as a way to gain competitive advantage (ORG6).</td>
<td>Establishing an individual’s perceived usefulness by the organization to reduce software maintenance cost.</td>
</tr>
<tr>
<td>36 My management values reducing maintenance cost below its current level for new products (ORG7).</td>
<td>Establishing an individual’s perceived usefulness by the organization to reduce software maintenance cost.</td>
</tr>
</tbody>
</table>
The project and product-related factors section includes a sub-section for assessing the degree to which a project exercises the various key processes that make up the Capability Maturity Model for Software. Due to the fact that not all software development organizations have gone through or have a plan to go through a SW-CMM based formal process maturity assessment, collecting data on Process Maturity solely on formal assessment would not yield good information for all organizations. For that reason, data on Process Maturity will be collected in two ways. The first way will be by selecting an overall maturity level based on some type of formal, independent or internal assessment. The second way is to ask respondents to rate Process Maturity of their project, by answering all of the SW-CMM Key process Area (KPA) questions, reflecting what actually happens on a project. Table 6 shows an example of the KPA questions.

The other sub-section of the project and product-related factors section also uses Likert-scale questions as shown in Tables 7 and 8 to assess respondents’ opinion about their projects. The questions are answered by selecting Strongly Disagree (SD=1), Disagree (D=2), Neutral (N=3), Agree (A=4), and Strongly Agree (SA=5). The total possible score for the organization factor portion is 50. A Cronbach’s alpha
reliability coefficient of 0.92 was calculated for the project factor scale.

Table 6: Example of KPA Scale Questions

<table>
<thead>
<tr>
<th>Key Process Area</th>
<th>Goals of each KPA</th>
<th>AA</th>
<th>FR</th>
<th>AH</th>
<th>OC</th>
<th>RE</th>
<th>NA</th>
<th>DK</th>
</tr>
</thead>
</table>
| **61 Requirements Management:** involves establishing and maintaining an agreement with the customer on the requirements for the software project (KPA1). | 1. System requirements allocated to software are controlled to establish a baseline for software engineering and management use.  
2. Software plans, products, and activities are kept consistent with the system requirements allocated to software. | ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ |
| **62 Software Project Planning:** establishes reasonable plans for performing the software engineering activities and for managing the software project (KPA2). | 1. Software estimates are documented for use in planning and tracking the software project.  
2. Software project activities and commitments are planned and documented.  
3. Affected groups and individuals agree to their commitments related to the software project. | ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ |

**LEGEND for the KPA Scale Questions:**

☐ Low, ☐ Medium, ☐ High, ☐ Excellent
AA = Almost Always (over 90% of the time) when the goals are consistently achieved and are well established in standard operating procedures.

FR = Frequently (about 60 to 90% of the time) when the goals are achieved relatively often, but sometimes are omitted under difficult circumstances.

AH = About Half (about 40 to 60% of the time) when the goals are achieved about half of the time.

OC = Occasionally (about 10 to 40% of the time) when the goals are sometimes achieved, but less often.

RE = Rarely If Ever (less than 10% of the time) when the goals are rarely if ever achieved.

NA = Does Not Apply when you have the required knowledge about your project or organization and the KPA, but you feel the KPA does not apply to your circumstances (e.g. Subcontract Management).

DK = Don’t Know when you are uncertain about how to respond for the KPA.

Table 7: Project Factor Scale Questions

<table>
<thead>
<tr>
<th>SMADAS Project Factor Scale Question</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>49 Project schedule was compressed so much that requirements were not fully analyzed and broken down into components before implementation began (SCHD).</td>
<td>Determination of the individual’s perception of realistic project schedule.</td>
</tr>
<tr>
<td>50 Software maintainability was clearly identified as an important quality attribute during software development (SMTY).</td>
<td>Determination of the project team’s perceived importance of software maintainability</td>
</tr>
<tr>
<td>51 In my opinion, there is good communication among the team members (TCOM).</td>
<td>Determination of team cohesion.</td>
</tr>
<tr>
<td>52 Tasks assigned to me are always delivered on or before their scheduled due dates (PEFF).</td>
<td>Determination of an individual’s effectiveness</td>
</tr>
</tbody>
</table>
Table 8: Project Factor Scale Questions

<table>
<thead>
<tr>
<th></th>
<th>Question</th>
<th>Evaluation</th>
</tr>
</thead>
<tbody>
<tr>
<td>53</td>
<td>I find myself spending a lot of time on rework (PCAP).</td>
<td>Evaluation of project team’s ability, efficiency, and thoroughness.</td>
</tr>
<tr>
<td>54</td>
<td>I had the appropriate experience/background for this project (PEXP).</td>
<td>Establishing an individual’s experience.</td>
</tr>
<tr>
<td>55</td>
<td>My project team is well experienced in the programming language(s) used to develop the software (LEXP).</td>
<td>Determination of project team’s programming language experience.</td>
</tr>
<tr>
<td>56</td>
<td>My project team is well experienced in the software tools used to develop the software (TOOL).</td>
<td>Determination of project team’s tools experience.</td>
</tr>
<tr>
<td>57</td>
<td>My project team has had prior experience with the type of software system described above (AEXP).</td>
<td>Determination of project team’s application experience.</td>
</tr>
<tr>
<td>58</td>
<td>The product is highly complex (CPLX). [Product Complexity is a continuum, where complexity of a product similar to a prior application is Low and a product that involves all new innovation is High]</td>
<td>Determination of project complexity.</td>
</tr>
</tbody>
</table>

Reliability and validity of the SMADAS

The instrument was tested for validity and reliability. Validity for this study was content validity, defined by Cooper and Schindler (2001) as “the extent to which the instrument provides adequate coverage of the investigative questions guiding the study” (p.211). A four-member panel of experts in the software development field was asked to validate the SMADAS...
instrument. These experts were asked to review the SMADAS scale to determine (a) any key attitude question the experts believe the software person must answer, (b) any key attitude question the experts believe should not be included in the scale, and (c) any wording change recommendations that will aid respondents to better understand the scale. The researcher carefully recorded the recommendations from each expert and made changes to the instrument to reflect those recommendations. The result is a more understandable and reliable instrument.

Reliability of the SMADAS scale was estimated in a pilot study using the Test-Retest method and the Coefficient alpha for internal consistency reliability. According to Cooper and Schindler (2001), the Test-Retest method is used to determine stability of the measurement scale. By obtaining consistent results with repeated measurements of the same people with the same instrument, the measure can be said to possess stability (Cooper & Schindler, 2001). The Coefficient alpha is usually more suitable for use with instruments whose items are not scored right or wrong. It is particularly useful for determining the reliability of attitude scales in which study participants are asked to respond on a continuum from strongly disagree to strongly agree (Cronk, 2002).
A Cronbach’s coefficient alpha was calculated for each of the four sub-scales. As Table 9 below shows, the alpha values for all sub-scales were significantly high, above 0.7 as recommended by (Litwin, 1995), suggesting the satisfactory nature of the internal consistency of the construct. Individual correlation for each specific question of the developer attitude sub-scale ranged from a low of –0.91 (question 15) to a high of 0.88 (question 30). For the organization factor sub-scale, individual correlation for each specific question ranged from a low of –0.13 (question 37) to a high of 0.91 (question 32). For the Process Maturity sub-scale, individual correlation for each specific question ranged from a low of –0.74 (question 66) to a high of 0.99 (question 59). For the project factor sub-scale, individual correlation for each specific question ranged from a low of –0.61 (question 48) to a high of 1.0 (question 55).

Table 9: SMADAS Scale Cronbach’s Coefficients

<table>
<thead>
<tr>
<th>Sub-Scale</th>
<th>Cronbach’s Alpha</th>
</tr>
</thead>
<tbody>
<tr>
<td>Developer Attitude (DATT)</td>
<td>0.92</td>
</tr>
<tr>
<td>Organization Factor (ORGF)</td>
<td>0.93</td>
</tr>
<tr>
<td>Process Maturity (PPMAT)</td>
<td>0.92</td>
</tr>
<tr>
<td>Project Factor (PROJA)</td>
<td>0.92</td>
</tr>
</tbody>
</table>
The Pilot Study for the SMADAS scale sample involved 10 software developers and software project managers from one of the corporation’s software development centers in the U.S. The tests were administered to the participants in November and December of 2002. In order to provide test, re-test reliability data, the researcher tested every participant twice, with interval of 4 weeks between sessions. In all, 7 people (6 software developers and 1 software development manager) responded. Permission to conduct the pilot study was obtained from the management of the corporation as well as from the study participants.

Data Collection Procedures

The research data collection was a structured self-reporting method utilizing a questionnaire as the instrument of choice. Data was collected for multiple software development projects. The data that was collected was grouped into four categories -- personnel, organizational, product, and project factors.

Personnel factors included the developer’s attitude towards software process improvement practices, attitude towards capability maturity model for software, and attitude towards the organization’s software development practices in general.
Organizational factors included an organization’s commitment to process improvement, whether or not an organization recognizes the need to reduce software maintenance cost and whether or not an organization sees software maintenance cost reduction as a way to gain competitive advantage.

Product factors included product size, complexity, and total cost since the first release to the customer. Project factors included the total number of months a project has been in maintenance (duration), effort (total number of hours spent on the project by the developer since its first release), process maturity level, software engineering methods and tools (tools used in making the software development more efficient). Other attributes included developer capability (developer ability, efficiency/thoroughness, and ability to communicate), team application experience (a rating dependent on the level of applications experience of the project team), language, and tool experience (the level of programming language and software tool experience of the project team).

Product size is the total thousand source lines of code generated for the project. This consists of total new lines of code developed, total lines of code adapted from previous
projects, and total lines of code that was developed but not used.

Process Maturity data was collected at both the organizational level and at the project level. Although an organization might be rated at one of the five levels of capability maturity model for software (SW-CMM), the respondents were still encouraged to answer all of the SW-CMM key process area (KPA) questions considering what actually happens on a project. The organizational level Process Maturity rating was determined by selecting an overall maturity level based on a formal assessment of the organization by an independent assessor. The second way to rate Process Maturity was to rate for each project, the percentage of compliance for each set of KPA goals. The goals for each KPA was considered in assigning a rating that reflect the percentage of compliance by the project. For this reason, Project Process Maturity would be computed as the average of all rated KPAs (“Does Not Apply” and “Don’t Know” do not count in the computation).

Effort, which is measured in Person Months, is the total number of months spent on the project by the developers since the first release of the software to the customer. This is calculated by multiplying the total number of months that has been spent to-date doing maintenance by the total number of
developers on the project. This is also what is considered as the cost. To put a dollar value to it would simply require expressing effort in Person Hours (i.e. multiplying effort in Person Months by 152 hours) and then multiplying the total person hours by the actual hourly rate. Effort here simply means software maintenance effort and unless otherwise stated, whenever effort is mentioned, it is assumed that software maintenance effort is intended.

Of course there are potential problems or biases that could arise such as respondents not providing the desired project data, the inherent bias of software developers against software maintenance. Although it might be the intention of study participants to answer all the questions as thoroughly and truthfully as possible, they may not provide the desired project data, not because they don’t want to, but because they may not have access to such information. Information such as the actual project size in terms of total lines of code and total effort may not be readily available or known to the software developers. To remedy this problem, the researcher redesigned questions to elicit the same information by grouping project size for example, into categories of very small, small, large, and very large projects and defining each category. Almost every software developer can tell if the project he/she is working on
falls under one of the categories. By so doing, the respondents will be able to provide the correct answers instead of leaving the questions blank.

Another possible bias is the tendency of software developers to identify more with new software development activities than with software maintenance. In most cases, software developers would not like to be identified with software maintenance activities, even though they may in fact be doing maintenance work. To minimize this type of bias, the researcher carefully defined the various types of software maintenance making it possible for respondents to indicate what percentage of their time they spend on each type of maintenance activity.

Data Analysis Procedures

Developer attitude data is ordinal in nature and therefore, statistical analysis techniques such as multiple linear regression, which requires interval or ratio data, cannot be rigorously applied. However, in most cases “valid results may be obtained with ordinal data” (Munro, 2001, p.246). A combination of descriptive and non-parametric techniques was adequate to establish the presence or absence of a statistically significant
correlation of developer attitude and software maintenance cost, and whenever possible, regressions were used.

Linear regression analyses were used to determine the relative importance of the set of independent variables, including the variable of interest, developer attitude. The standardized coefficient beta of each variable was then analyzed to determine the contribution of each independent variable (including developer attitude) to the overall maintenance cost.

In addition, the Spearman’s rank order correlation was performed to test the direction and strength of the relationship between the independent variable (developer attitude) and the dependent variable (software maintenance cost). The standard method for testing for the statistical significance of a correlation coefficient requires an assumption that may not hold in this case. Spearman’s rank-order correlation coefficient is an alternate measure of correlation that can be used to test whether the association between developer attitude and software maintenance cost is statistically significant. Like the standard correlation coefficient, the Spearman coefficient ranges between -1 and +1.
Computation of Project Process Maturity Number

For the purposes of analyzing the effect of process maturity on software maintenance cost, instead of the overall process maturity (PMAT) of an organization, the project process maturity (PPMAT) would be used since the data was collected at the project level. Another reason for not using PMAT is because not all organizations have been through the formal Software Engineering Institute’s CMM-Based Appraisal for Internal Process Improvement (CBA-IPI) assessment. If it were based on CBA-IPI assessment, most of the respondents would not know how to assess their organizations and would have no choice but to rate them at CMM Level 1.

PPMAT is determined by taking the average of all rated key process areas (KPAs). There are 18 possible KPAs. The KPAs that are rated as “Does Not Apply” or “Don’t Know” are not taken into consideration for PPMAT computation. The only problem is that this sometimes causes the total number of KPAs to be less than 18.

\[
PPMAT = \left( \frac{\sum KPA_i}{100} \right) \times \frac{1}{n}
\]

Where \( n \) is the total number of rated KPAs.
This computation means that PPMAT score ranges from 0 to 1, implying that the higher the score, the more mature a project’s process is. It was determined that the median rating for PPMAT is 0.5. For comparison sake, projects that score less than 0.5 are considered to be low on the process maturity scale, and those that score 0.5 and above are considered to be high on the maturity scale.

Assumptions

It was assumed that participants in the normative sample would be similar to the software developers that participated in the pilot study in terms of educational experience, professional work experience. A comparison of the pilot study results showed that the pilot study group scores were not significantly different from the scores of the actual study sample.

Limitations

This study relied on software developers’ self-reporting of effort data. This self-reporting of maintenance effort data could have introduced errors such that the researcher might not have gotten sufficiently good measures of the dependent variable (software maintenance cost) to see any correlation. To ensure that this was not so, the researcher attempted to obtain actual effort measures from one or more of the business units in the
target corporation to be used to crosscheck the self-reported measures for face validity. However, the researcher was only able to obtain corrective maintenance effort data for one of the projects. While some of the organizations with more mature processes maintained certain project data, including aggregate cost data for software development, software maintenance cost data is not being collected or maintained as a separate activity. The cost of software development is lumped together with software maintenance, perhaps an indication of the organizations’ attitude toward software maintenance.

Other limitations included the researcher not being in control of who received the survey and who didn’t. As the distribution lists of the various software organizations were controlled by the business units, there was no way the researcher could say for sure how many people did in fact receive the invitation to participate in the survey. All the researcher could do was hope and trust that the business units were indeed sending out the survey notifications when the researcher asked them to.

As a result of the researcher not being in control of the distribution list, no pre-notice letter was sent out to the participants to prepare them and explain what was to come.
Summary

In summarizing, this chapter presented the research design and methods that were used for data collection to investigate the relationship between software developer’s attitude towards software process improvement practices and software maintenance cost. This section also described the target population of interest, the sample, and the instrument used to conduct the survey, the data collection and the data analysis procedures.
CHAPTER 4 DATA COLLECTION, ANALYSIS AND FINDINGS

Introduction

This chapter presents the findings and analysis of the study. In an effort to determine the relationship between software developer’s attitude towards software best practices and software maintenance cost, both descriptive and correlational results are presented.

The research questions and the hypotheses that needed to be tested are presented followed by the procedures used in the study. The characteristics of the sampled software developers are then presented followed by a description of the data collected. Next, the actual data analyses of the results obtained from the Survey are then presented.

In the data analysis section, the results of the hypotheses testing are presented. This includes the correlation results between software developers’ attitudes towards software improvement best practices and the cost of software maintenance, results of the relationship between the process maturity of a software development organization and software maintenance cost, results of how much time software developers spend doing software maintenance are presented, and what accounts for the total maintenance workload for an organization. Chapter 4 concludes with a summary of the findings and discussions of the statistical findings.
Research Questions and Hypotheses

There were four research questions that needed to be answered and for each question, there were some proposed hypotheses. The following four questions and their associated hypotheses were tested to determine if there is a correlation between software developers' attitudes toward software best practices and software maintenance cost.

1. Is there a correlation between a software developer’s attitude towards software improvement best practices and the cost of software maintenance to the organization? The following hypotheses were tested to answer this question:

   \( H_0 \): There is no correlation between software developer’s attitude towards software process improvement practices and software developer’s perceived software maintenance cost.

   \( H_1 \): As the developer’s perceived usefulness (U) of software process improvement and the developer’s perceived ease of use (EOU) of software best practices increase (U, EOU), developer attitude (A) and the developer’s behavioral intentions (BI) to use those software best practices will increase.

   \( H_b \): As developer attitude (A) towards software best practices and the developer’s perceived usefulness (U) of software process improvement increase (A, U), the behavioral intentions (BI) of the developer to use software best practices will increase.

   \( H_c \): As the developer’s behavioral intentions (BI) to use software best practices increase, the tendency of the developer to use software best practices (AU) will increase.

   \( H_d \): A correlation between software developer’s attitude towards software process improvement practices and software maintenance cost (RBI) will be higher for those with high
actual usage of the software best practices than those with low usage of the software best practices.

2. Is there a relationship between the process maturity of a software development organization, as defined by the Software Engineering Institute’s Capability Maturity Model (SEI-CMM) and software maintenance cost? The following hypotheses were tested to answer this question:

$H_02$: There is no difference in the correlation between Process Maturity and software maintenance cost for organizations with mature processes and organizations with less mature processes for developing software.

$H_a2$: The correlation between Process Maturity and software maintenance cost will be higher for organizations with mature processes than for organizations with less mature processes for developing software.

$H_02$: There is a negative correlation between organizations that recognize software maintenance cost as areas to gain competitive advantage and software maintenance cost.

3. What percentage of time do software developers spend doing software maintenance? The following hypotheses were tested to answer this question:

$H_03$: Software developers do not spend a significant amount of their time doing software maintenance.

$H_a3$: Software developers spend a significant amount of their time on software maintenance than they admit.

4. What accounts for the total maintenance workload for an organization, is it Adaptive, Corrective, Perfective or Preventive maintenance? Related to this question, the following hypotheses are proposed:

$H_04$: Majority of an organization’s software maintenance effort is not spent doing Corrective and Perfective maintenance.
H₄: Majority of an organization’s software maintenance effort is spent doing Corrective and Perfective maintenance.

Study Procedures

The study used an online survey instrument that was sent to the respondents via e-mail. The secure survey link, a personal letter from the researcher to the participants and an introductory letter from the sponsoring director (see Appendix B) were sent via e-mail to each software development organization manager. Each manager then forwarded the e-mail containing the letters and the survey link to their software engineers (software developers and software project managers). A total of 430 software engineers received the invitation to participate in the study. The invitations explained the purpose of the study and the importance of their participation. The respondents were asked to click on the URL (universal or Uniform Resource Locator) link provided in the e-mail message, which linked to the Web-based online survey instrument. They were also given the required password needed to gain access to the survey.

To motivate software engineers to respond, they were invited to participate in an interesting software engineering study, as Software engineers are often too busy in their jobs to contribute to advances in software engineering research. They were reminded about valuable contributions that can be made by
practicing developers when they participate in such a research. After two weeks of the initial administration of the survey, another e-mail with the cover letters and the survey link was sent to the participants reminding and encouraging them to take the survey.

Although a monetary incentive was offered to respondents at the end of the survey, they were not made aware of it in the beginning, a recognition of the fact that money is not the top motivating factor for software engineers (Levy, 1998). Each person that completed the survey was given an option to provide a name and address to receive a free gift award of $10. Of those that completed the survey, only 26 percent (n = 39) of the respondents requested to receive the free gift. One person actually returned the $10 to the researcher with due thanks. Again, confirming what has been said about what motivates software engineers.

The respondents were also assured that theirs and the confidentiality of their organizations would be protected. It was indeed for that reason that the researcher was not given access to the distribution list of the participants. Respondents were also assured that the results would be reported in aggregate to guarantee their anonymity, and to ensure that, neither their names nor the organizations they belong to were
requested. Of the 430 participants that were surveyed, 152 responded, resulting in an effective response rate of 35.4%.

The online survey questionnaire itself was designed with the aid of a survey software package called the StatPac for Windows survey software. Although StatPac for Windows has integrated capability to do questionnaire design, sample selection, data entry and editing, Internet and email surveys, and integrated statistical analysis reporting, only its Internet survey design capability was utilized. It simplified the data collection process and the coding that would have taken place after data collection. As the recorded data was already coded, both coding and data entry did not have to be done again, thus saving the researcher at least 3-4 weeks of data entry and coding time. In addition, data entry error was no longer an issue as the software package was used to import the data as entered by the respondents directly into a spreadsheet and then into the SPSS software for analysis.

Characteristics of the Sample

The data collection method was based on a voluntary and self-administered Web-based questionnaire that was completed by the participants. As they answered the questions, their responses were recorded on the researcher’s Web server hosted by mercury.powweb.com. A survey link to the questionnaires was sent
to 430 Rochester-based software professionals in a Fortune 500 corporation. Within this population, 152 software professionals responded by completing the questionnaires resulting in a return rate of 35.4%.

Of the 152 respondents that completed the survey, ninety-six (96) responses were from software developers, twenty-four (24) represented responses from software project managers, and eighteen (18) were responses from people who considered themselves as both software developers and software project managers. Five (5) people responded as neither software developer nor software project manager, and nine (9) respondents did not provide an answer. Those fourteen questionnaires were discarded. In addition, questionnaires where respondents did not provide an answer or did not meet the established criteria were also eliminated. The established criteria were (a) the questionnaire must be properly filled out and (b) the respondent must develop software or manage software project in his or her job. Fifty-three (53) responses were thrown out for either of the reasons given above leaving a total number of 99 valid responses. Therefore, the effective valid response rate from the participants was 65.13% (99/152).

There was no prohibition from the corporation with respect to the type of information to collect from the software
professionals, but the researcher intentionally did not want to collect names and the respondents’ organizational units. This was to give participants enough confidence to answer the questions honestly without fear of reprisals.

The participants were very much representative of the corporation in terms of gender make up, level of education and professional work experience for this particular population of the corporation. As Table 10 shows, about twenty-six percent of the sampled software professionals are females, and sixty-seven percent are males. Comparing this characteristic with the corporation, gender representation in one division and a business unit parallel the representations seen in this study. In one division, as of January 1, 2003, 61.2% are males while 38.8% are females. Similarly, in the business unit, males make up 77% of the population and females represent 23% of the same population.

Less than sixteen percent of the respondents had been in their present position less than 2 years, and more than eighty-two percent of the software engineers have had more than 5 years total software development experience (see Table 11). Level of education for this group was also high, as Table 12 shows. More than eighty-eight percent of the respondents had at least 4-year college degree.
Table 10: Breakdown of respondents by gender

<table>
<thead>
<tr>
<th>Gender</th>
<th>Percent</th>
</tr>
</thead>
<tbody>
<tr>
<td>Female</td>
<td>26.3%</td>
</tr>
<tr>
<td>Male</td>
<td>67.1%</td>
</tr>
<tr>
<td>Missing</td>
<td>6.6%</td>
</tr>
</tbody>
</table>

Table 11: Years of Experience of the Study Sample

<table>
<thead>
<tr>
<th>Years in current Position</th>
<th>Percent</th>
<th>Years in the software profession</th>
<th>Percent</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-2 years</td>
<td>15.8%</td>
<td>0-2 years</td>
<td>3.9%</td>
</tr>
<tr>
<td>3-5 years</td>
<td>24.3%</td>
<td>3-5 years</td>
<td>7.9%</td>
</tr>
<tr>
<td>6-8 years</td>
<td>11.8%</td>
<td>6-8 years</td>
<td>5.9%</td>
</tr>
<tr>
<td>9-11 years</td>
<td>7.9%</td>
<td>9-11 years</td>
<td>7.9%</td>
</tr>
<tr>
<td>12+ years</td>
<td>34.2%</td>
<td>12+ years</td>
<td>68.4%</td>
</tr>
<tr>
<td>Missing</td>
<td>5.9%</td>
<td>Missing</td>
<td>5.9%</td>
</tr>
</tbody>
</table>

Table 12: Educational Level of the Study Sample

<table>
<thead>
<tr>
<th>Level of Education</th>
<th>Percent</th>
</tr>
</thead>
<tbody>
<tr>
<td>Some College</td>
<td>1.3%</td>
</tr>
<tr>
<td>2-year College degree</td>
<td>3.9%</td>
</tr>
<tr>
<td>4-year College degree</td>
<td>50.0%</td>
</tr>
<tr>
<td>Masters degree</td>
<td>38.8%</td>
</tr>
<tr>
<td>Ph.D degree</td>
<td>0%</td>
</tr>
<tr>
<td>Missing</td>
<td>5.9%</td>
</tr>
</tbody>
</table>
Table 13: Developers’ Perception of their Organization’s Maintenance Attitude

<table>
<thead>
<tr>
<th>Organizational Factor</th>
<th>Agree</th>
<th>Disagree or Neutral</th>
</tr>
</thead>
<tbody>
<tr>
<td>Organization Sees the need to reduce software maintenance cost</td>
<td>50.5%</td>
<td>49.5%</td>
</tr>
<tr>
<td>Organization recognizes software maintenance cost reduction as an area to gain competitive advantage</td>
<td>27.3%</td>
<td>72.7%</td>
</tr>
<tr>
<td>Organization values reducing software maintenance cost</td>
<td>38.4%</td>
<td>61.6%</td>
</tr>
</tbody>
</table>

Data Description

There are seventy-eight data items used in this research. Although the actual data recorded when respondents chose an answer were numerical values ranging from 1 to 5, all the data are qualitative in nature with the exception of Effort data.

The data came from over 20 projects in a Fortune 500 corporation. These projects covered Communications, Desktop Publishing, Engineering and Science, Imaging/Scanning, Information Management, Process Control, Signal Processing, Simulations, Testing Tools, Web Software and Command and Control applications. The data was on projects that have been released at least once to the client or customer for use. Most of the data came from recent projects (1990s and 2000s projects).
Product sizes range from less than 50 KSLOC to over 150 KSLOC, Figure 10.

The product size data has an average of 3.1, which means that most of the projects have a size between 75 KSLOC and 100 KSLOC, and a standard deviation of 1.41. Project effort ranges from 0.3 to 1300 Person Months, with an average of 100.6 person-months and a standard deviation of 196.99, Figure 11.
The data collected were on attitude predictor variables, organizational variables, project factor variables including the effort expended after the project had been released, and process maturity. Supporting information such as type of application was collected. No data was collected on reuse sizing data or on whether the software technology used on the project matched the application complexity. Data about whether the processes used to develop the software were mismatched to the type of application domain was not collected either. All of these data were collected through self-reporting by the software developers. These and other factors, such as the interpretation of qualitative ratings mean that the data are imprecise and the
researcher does not expect them to be precise because the study was only interested in establishing correlation.

Data Analysis

Data analysis was conducted with a combination of descriptive and non-parametric techniques to establish the presence or absence of a statistically significant correlation of developer attitude and software maintenance cost. Linear regression analysis was used to determine the relative importance of the set of independent variables. The standardized coefficient beta of each variable was analyzed to determine the contribution of each independent variable to the overall maintenance cost. In addition, the Spearman’s rank order correlation was performed to test the direction and strength of the relationship between developer attitude and software maintenance cost. Graphical analysis tools and statistical tools were used to aid in the analysis of the data.

The researcher purchased and used the SPSS® Student Version 10.0 for Windows for all the analyses that are described here. The data analyses included frequency distributions; scatter plots, correlation matrices and regression analyses. The correlation matrix for the study’s principal constructs using Spearman’s rank-difference method is shown in Table 14.
Table 14: Spearman’s Correlation Matrix of Study Constructs

<table>
<thead>
<tr>
<th></th>
<th>EOU</th>
<th>U</th>
<th>A</th>
<th>BI</th>
<th>AU</th>
<th>DATT</th>
<th>RBI</th>
</tr>
</thead>
<tbody>
<tr>
<td>Perceived Ease Of Use (EOU)</td>
<td>1.00</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Perceived Usefulness (U)</td>
<td>.60**</td>
<td>1.00</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Attitude Toward Using (A)</td>
<td>.50**</td>
<td>.57**</td>
<td>1.00</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Behavioral Intentions (BI)</td>
<td>.50**</td>
<td>.57**</td>
<td>.52**</td>
<td>1.00</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Actual Use (AU)</td>
<td>.28**</td>
<td>.42**</td>
<td>.31**</td>
<td>.37**</td>
<td>1.00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Developer Attitude (DATT)</td>
<td>.62**</td>
<td>.72**</td>
<td>.70**</td>
<td>.65**</td>
<td>.50**</td>
<td>1.00</td>
<td></td>
</tr>
<tr>
<td>Real Business Impact (RBI)</td>
<td>.13*</td>
<td>-0.01*.05*.05*.12*.06*</td>
<td>1.00</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

n=99; *p > .05, **p < .01 (2-tailed).

Figure 12. Results of Developer Attitude on SW Maintenance

Figure 12 shows the results for the portion of the research model that describes the contribution of each intervening
variable to the impact on software maintenance cost. The standardized coefficient beta of each variable is provided to show its contribution. In terms of the predictors for attitude toward using best practices \((R^2 = .45)\), both perceived ease of use and perceived usefulness contributed strongly to attitude toward using, but perceived ease of use was the most influential \((b=.63, \ p<.01)\), strongly supporting \(H_{a1}\). Tables 15 through 17 show Perceived Ease of Use, Perceived Usefulness and Attitude constructs respectively and how they were operationalized.

<table>
<thead>
<tr>
<th>Table 15: Perceived Ease of Use Construct</th>
</tr>
</thead>
<tbody>
<tr>
<td>Construct</td>
</tr>
<tr>
<td>-----------</td>
</tr>
<tr>
<td>Perceived Ease of Use (EOU)</td>
</tr>
</tbody>
</table>
Table 16: Perceived Usefulness Construct

<table>
<thead>
<tr>
<th>Construct</th>
<th>Description</th>
<th>Questions for Operationalization</th>
</tr>
</thead>
<tbody>
<tr>
<td>Perceived Usefulness</td>
<td>The degree to which a software developer believes that using software</td>
<td>13. Learning about process improvement models such as CMM or CMMI is not a waste of time (DAT6).</td>
</tr>
<tr>
<td>(U)</td>
<td>improvement best practices would enhance his or her job performance.</td>
<td>15. I think documenting my design reduces the amount of time I spend on rework (DAT8).</td>
</tr>
<tr>
<td></td>
<td></td>
<td>16. Documenting code makes it possible to spend less time doing rework (DAT9).</td>
</tr>
<tr>
<td></td>
<td></td>
<td>17. I think the time I spend documenting my design and code leads to reducing the time spent on maintenance later on (DAT10).</td>
</tr>
<tr>
<td></td>
<td></td>
<td>19. I think using software improvement processes adds value to my project (DAT12).</td>
</tr>
<tr>
<td></td>
<td></td>
<td>25. Employing software best practices (such as risk management, reviews/inspections, requirements management, metrics-based scheduling, defect tracking, size estimating, and so on) improves the overall quality of software (DAT18).</td>
</tr>
<tr>
<td></td>
<td></td>
<td>26. Software process improvement activities would increase my overall productivity (DAT19).</td>
</tr>
</tbody>
</table>
Table 17: Attitude Construct

<table>
<thead>
<tr>
<th>Construct</th>
<th>Description</th>
<th>Questions for Operationalization</th>
</tr>
</thead>
<tbody>
<tr>
<td>Attitude</td>
<td>The developer’s positive or negative feelings about using software improvement best practices.</td>
<td>27. Documented software development standards should be encouraged as much as possible (DAT20).</td>
</tr>
<tr>
<td></td>
<td></td>
<td>28. In my opinion, software maintenance is one of the most important activities in the software life cycle (DAT21).</td>
</tr>
<tr>
<td></td>
<td></td>
<td>29. Software maintenance cost can be reduced dramatically if software engineering best practices are followed (DAT22).</td>
</tr>
</tbody>
</table>

Actual use of software improvement best practices ($R^2 = .10$) was predicted by intentions to use ($b = .32$, $p < .01$), which validated $H_01$. Table 18 shows the Actual Use construct and how it was operationalized.
<table>
<thead>
<tr>
<th>Construct</th>
<th>Description</th>
<th>Questions Used for Operationalization</th>
</tr>
</thead>
</table>
| Actual Use    | Actual Use is measured in terms of frequency of use of software improvement best practices and the volume of software best practices use by the developer. | 7. Of all activities I perform, software design is one of the activities I enjoy doing most (DAT1).  
8. Of all activities I perform, software coding is one of the activities I enjoy doing most (DAT2).  
9. Of all activities I perform, software testing is one of the activities I enjoy doing most (DAT3).  
10. I am familiar with the intent of software process improvement models such as the CMM or CMMI (DAT4).  
11. I am familiar with the content of software process improvement models such as the CMM or CMMI (DAT5).  
13. I usually document requirements before design and coding begin (DAT7).  
17. I usually make a written plan before I start to solve a problem (DAT11). |

For the predictors of intention to use software improvement best practices ($R^2 = .43$), both perceived usefulness ($b=.58$, $p<.01$) and attitude toward using ($b=.59$, $p<.01$) were almost identical in their influences, however, attitude toward using was slightly more influential, strongly supporting $H_{b1}$. Table 19
shows Behavioral Intentions construct and how it was operationalized.

<table>
<thead>
<tr>
<th>Construct</th>
<th>Description</th>
<th>Questions Used for Operationalization</th>
</tr>
</thead>
<tbody>
<tr>
<td>Behavioral Intentions to Use (BI)</td>
<td>Behavioral Intention is the measure of the strength of the developer’s intention to use software improvement best practices.</td>
<td>20. I always follow my team's chosen methods for software development instead of choosing my own way (DAT13).</td>
</tr>
<tr>
<td></td>
<td></td>
<td>21. I would like to learn more about software process improvement activities (DAT14).</td>
</tr>
<tr>
<td></td>
<td></td>
<td>23. I see software development standards as something I will always use in my implementation of software (DAT16).</td>
</tr>
</tbody>
</table>

But contrary to what was hypothesized, perceived software maintenance cost ($R^2 = .01$) could not be determined by actual use of software improvement best practices ($b=.10$, $p>.05$), thereby not validating $H_{01}$. Table 20 shows the Real Business Impact construct and how it was operationalized.
Table 20: Real Business Impact Construct

<table>
<thead>
<tr>
<th>Construct</th>
<th>Description</th>
<th>Questions for Operationalization</th>
</tr>
</thead>
<tbody>
<tr>
<td>Real Business Impact (Developer Perceived Software Maintenance Cost)</td>
<td>The developer’s perceived reported time spent on his or her software project that is in maintenance.</td>
<td>45. How many months has your team spent on development and testing of the software you described above since it was first released to the customer (NMOS)?</td>
</tr>
<tr>
<td></td>
<td></td>
<td>46. What is the average number of hours per person, per month spent by your team on this project since it was first released to the customer (NHRS)?</td>
</tr>
<tr>
<td></td>
<td></td>
<td>47. How many software developers (including the project manager) are or were on your project team (TSIZ)?</td>
</tr>
</tbody>
</table>
had an average of 6-8 years on their current position and an average of 9-11 years experience in software development.

Tests of Hypotheses

Hypothesis 1

$H_0$: There is no correlation between software developer’s attitude towards software process improvement practices and software developer’s perceived software maintenance cost.

Table 14 shows correlation results for the effects of the study’s predictor variables on real business impact (software maintenance cost). A Spearman rho correlation coefficient was calculated for the relationship between a software developer’s attitude towards usage of software improvement best practices and cost of software maintenance (maintenance effort). A weak correlation that was not significant was found ($r (97) = .12$, $p > .05$). Even when the overall developer attitude (DATT) and software maintenance effort were compared, a weaker correlation that was not significant was found ($r (97) = .06$, $p > 0.5$).

Beyond what software developer’s attitude shares with other variables, it does not account for any of the variance in total project effort. Therefore, regarding the software maintenance and developer attitude scale, Software developer’s attitude towards software process improvement practices is not related to...
software maintenance cost. Hence, the first null hypothesis cannot be rejected.

Hypothesis 2

$H_a1$: As the developer’s perceived usefulness ($U$) of software process improvement and the developer’s perceived ease of use ($EOU$) of software best practices increase ($U$, $EOU$), developer attitude ($A$) and the developer’s behavioral intentions ($BI$) to use those software best practices will increase.

Table 21: Effect of $U$ and $EOU$ on Attitude ($A$)

<table>
<thead>
<tr>
<th>Variable</th>
<th>B</th>
<th>SE B</th>
<th>Beta ($\beta$)</th>
<th>t</th>
<th>Sig.</th>
</tr>
</thead>
<tbody>
<tr>
<td>(Constant)</td>
<td>0.97</td>
<td>0.31</td>
<td>3.13</td>
<td>.002</td>
<td></td>
</tr>
<tr>
<td>Perceived Usefulness ($U$)</td>
<td>0.35</td>
<td>0.09</td>
<td>0.40</td>
<td>3.79</td>
<td>.000</td>
</tr>
<tr>
<td>Perceived Ease of Use($EOU$)</td>
<td>0.31</td>
<td>0.10</td>
<td>0.33</td>
<td>3.16</td>
<td>.002</td>
</tr>
</tbody>
</table>

Note: $N = 99$, $R^2 = .45$, $p < .01$

Table 22: ANOVA Summary of Effect of $U$ and $EOU$ on Attitude ($A$)

<table>
<thead>
<tr>
<th>Model</th>
<th>Sum of Squares</th>
<th>df</th>
<th>Mean Square</th>
<th>F</th>
<th>Sig.</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Regression</td>
<td>2</td>
<td>12.07</td>
<td>39.27</td>
<td>.000</td>
</tr>
<tr>
<td></td>
<td>Residual</td>
<td>96</td>
<td>.31</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Total</td>
<td>53.65</td>
<td>98</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Note: Predictors: (Constant), $U$, $EOU$; Dependent Variable: $A$
A multiple linear regression was calculated to predict software developers’ attitudes toward using software best practices based on their perceived usefulness and perceived ease of use. A significant regression equation was found ($F\ (2,96) = 39.27, p < .001$) with an $R^2$ of 0.45. Software developers’ predicted Attitudes Toward Using software best practices is equal to $0.97 + 0.35(U) + 0.31(EOU)$, when perceived usefulness ($U$), perceived ease of use ($EOU$), and attitude toward using ($A$) are coded on a scale of 1 to 5, with 1 being the lowest and 5 being the highest. Developers’ attitudes increased 0.35 points
for each point increase of perceived usefulness, plus 0.31 points for each point increase of perceived ease of use on the Likert scale. Both perceived usefulness and perceived ease of use were significant predictors.

For the second part of this test, a multiple linear regression was also calculated to predict software developers’ behavioral intentions to use software best practices based on their perceived usefulness and perceived ease of use. A significant regression equation was also found ($F (2,96)= 33.04, p < .001$) with an $R^2$ of 0.41. Software developers’ predicted Behavioral Intentions (BI) to use software best practices is equal to $0.76 + 0.37(U) + 0.33(EOU)$, when perceived usefulness (U), perceived ease of use (EOU), and behavioral intentions to use (BI) are coded on a scale of 1 to 5, with 1 being the lowest and 5 being the highest. Developers’ attitudes increased 0.35 points for each point increase of perceived usefulness, plus 0.31 points for each point increase of perceived ease of use on the Likert scale. Both perceived usefulness and perceived ease of use were significant predictors.

These findings strongly support $H_{a1}$, thereby making it possible to accept the hypothesis as stated, and that is, as the developer’s perceived usefulness of software process improvement and the developer’s perceived ease of use of software best
practices increase, developer attitude and the developer’s behavioral intentions to use those software best practices increases as well.

Hypothesis 3

H₃: As developer attitude (A) towards software best practices and the developer’s perceived usefulness (U) of software process improvement increase (A, U), the behavioral intentions (BI) of the developer to use software best practices will increase.

Table 25: Effect of A and U on Behavioral Intentions (BI)

<table>
<thead>
<tr>
<th>Variable</th>
<th>B</th>
<th>SE B</th>
<th>Beta (β)</th>
<th>t</th>
<th>Sig.</th>
</tr>
</thead>
<tbody>
<tr>
<td>(Constant)</td>
<td>0.60</td>
<td>0.37</td>
<td>1.64</td>
<td>.104</td>
<td></td>
</tr>
<tr>
<td>Perceived Usefulness (U)</td>
<td>0.37</td>
<td>0.10</td>
<td>0.35</td>
<td>3.57</td>
<td>.001</td>
</tr>
<tr>
<td>Perceived Ease of Use (EOU)</td>
<td>0.42</td>
<td>0.11</td>
<td>0.38</td>
<td>3.90</td>
<td>.000</td>
</tr>
</tbody>
</table>

Note: N = 99, R² = .43, p < .01

Table 26: ANOVA Summary of Effect of A and U on BI

<table>
<thead>
<tr>
<th>Model</th>
<th>Sum of Squares</th>
<th>df</th>
<th>Mean Square</th>
<th>F</th>
<th>Sig.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Regression</td>
<td>28.41</td>
<td>2</td>
<td>14.21</td>
<td>35.43</td>
<td>.000</td>
</tr>
<tr>
<td>Residual</td>
<td>38.50</td>
<td>96</td>
<td>0.40</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Total</td>
<td>66.91</td>
<td>98</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Note: Predictors: (Constant), A, U; Dependent Variable: BI

A multiple linear regression was calculated to predict software developers’ behavioral intentions to use software best
practices based on their attitudes toward using software best practices and perceived usefulness. A significant regression equation was found \( F (2,96) = 35.43, p < .001 \) with an \( R^2 \) of 0.43. Software developers’ predicted Behavioral Intentions (BI) to use software best practices is equal to 0.60 + 0.37(U) + 0.42(A), when perceived usefulness (U), attitude toward using (A), and behavioral intentions to use (BI) are coded on a scale of 1 to 5, with 1 being the lowest and 5 being the highest. Developers’ behavioral intentions increased 0.37 points for each point increase of perceived usefulness, plus 0.42 points for each point increase of attitude toward using on the Likert scale. Both perceived usefulness and attitude toward using were significant predictors.

The findings strongly support \( H_b1 \), thereby making it possible to accept the hypothesis as stated, that is, as developer attitude towards software best practices and the developer’s perceived usefulness of software process improvement increase, the behavioral intentions of the developer to use software best practices increases too.

**Hypothesis 4**

\( H_{b1} \): As the developer’s behavioral intentions (BI) to use software best practices increase, the tendency of the developer to use software best practices (AU) will increase.
Table 27: Effect of Behavioral Intentions on Actual Use

<table>
<thead>
<tr>
<th>Variable</th>
<th>B</th>
<th>SE B</th>
<th>Beta (β)</th>
<th>t</th>
<th>Sig.</th>
</tr>
</thead>
<tbody>
<tr>
<td>(Constant)</td>
<td>3.00</td>
<td>0.23</td>
<td>13.12</td>
<td>.000</td>
<td></td>
</tr>
<tr>
<td>Behavioral Intentions (BI)</td>
<td>0.20</td>
<td>0.06</td>
<td>0.32</td>
<td>3.30</td>
<td>.001</td>
</tr>
</tbody>
</table>

Note: N = 99, R² = .10, p < .01

Table 28: ANOVA Summary of Effect of BI and AU

<table>
<thead>
<tr>
<th>Model</th>
<th>Sum of Squares</th>
<th>df</th>
<th>Mean Square</th>
<th>F</th>
<th>Sig.</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Regression</td>
<td>2.74</td>
<td>1</td>
<td>2.74</td>
<td>10.89</td>
<td>.001</td>
</tr>
<tr>
<td>Residual</td>
<td>24.43</td>
<td>97</td>
<td>0.25</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Total</td>
<td>27.17</td>
<td>98</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Note: Predictors: (Constant), BI; Dependent Variable: AU

A simple linear regression was calculated predicting software developers’ actual usage of software best practices based on their behavioral intentions. A significant regression equation was found (F (1,97) = 10.89, p < .001), with an R² of 0.10. Software developers’ predicted actual use of software best practices is equal to 3.0 + 0.20(BI), when behavioral intentions to use (BI) are coded on a scale of 1 to 5, with 1 being the lowest and 5 being the highest. On the average, software developers’ actual use of software best practices increased 0.20 points for each point increase of software developers’ behavioral intentions on the Likert scale.
The findings strongly support $H_01$, thereby making it possible to accept the hypothesis as stated, that is, as developer’s behavioral intentions (BI) to use software best practices increase, the tendency of the developer to use software best practices (AU) will increase as well.

Hypothesis 5

$H_01$: A correlation between software developer’s attitude towards software process improvement practices and perceived software maintenance cost (RBI) will be higher for those with high actual usage of the software best practices than those with low usage of the software best practices.

Table 29: Correlation Coeff. for DATT and RBI for Low AU

<table>
<thead>
<tr>
<th></th>
<th>DATT</th>
<th>RBI (Effort)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Developer Attitude (DATT)</td>
<td>1.00</td>
<td></td>
</tr>
<tr>
<td>Real Business Impact (RBI)</td>
<td>.14*</td>
<td>1.00</td>
</tr>
</tbody>
</table>

Note: $n=49$, *$p > .05$, (2-tailed).

Table 30: Correlation Coeff. for DATT and RBI for High AU

<table>
<thead>
<tr>
<th></th>
<th>DATT</th>
<th>RBI (Effort)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Developer Attitude (DATT)</td>
<td>1.00</td>
<td></td>
</tr>
<tr>
<td>Real Business Impact (RBI)</td>
<td>-.13*</td>
<td>1.00</td>
</tr>
</tbody>
</table>

Note: $n=50$, *$p > .05$, (2-tailed).
In order to make this comparison, some measures of central tendency, the mean, median, and range were calculated for the actual use variable. In addition, the minimum and maximum scores were also determined (Mean = 3.74, Median = 4, Range = 3, Minimum = 2, Maximum = 5). The data had to be split in half, one half representing those respondents with low actual usage of software best practices, and the other half representing those with high actual usage of software best practices. So using the median, the data was split in half 49/50 before calculating the correlation for each half.

Tables 29 and 30 show correlation results for the effects of developer attitude on real business impact (software maintenance cost). A Spearman rho correlation coefficient was calculated for the relationship between a software developer’s attitude towards usage of software improvement best practices and cost of software maintenance (maintenance effort) for those with low actual usage of the software best practices. A weak correlation that was not significant was found ($r (47) = .14$, $p > .05$). For those with high actual usage of software best practices, a Spearman rho correlation coefficient was also calculated. A weak negative correlation that was not significant was found ($r (48) = -.13$, $p > .05$).
Regarding the software maintenance and developer attitude scale, software developer’s attitude towards software process improvement practices is not related to software maintenance cost, which does not support $H_01$. Therefore, the hypothesis is rejected.

Hypothesis 6

$H_02$: There is no difference in the correlation between Process Maturity and software maintenance cost for organizations with mature processes and organizations with less mature processes for developing software.

![Scatterplot of Effort and Process Maturity](image)

Figure 13. Scatterplot of Effort and Process Maturity
Table 31: Correlation Coefficients for PPMAT and EFFORT

<table>
<thead>
<tr>
<th></th>
<th>PPMAT</th>
<th>RBI (Effort)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Project Process Maturity (PPMAT)</td>
<td>1.00</td>
<td></td>
</tr>
<tr>
<td>Real Business Impact/Total Effort (RBI)</td>
<td>-0.20*</td>
<td>1.00</td>
</tr>
</tbody>
</table>

Note: n=99, *p < .05, (2-tailed)

As shown in Table 31, a Spearman rho correlation coefficient was calculated for the relationship between project process maturity and cost of software maintenance (total software maintenance effort). A weak negative correlation was found ($r (97) = -.20, p < .05$). And as Figure 13 shows, the negative correlation indicates that the lower the project process maturity of an organization, the more effort it spends on software maintenance, not supporting $H_0^2$. Therefore, $H_0^2$ must be rejected. This is also in harmony with findings by other studies about the effects of process maturity on software development effort (Clark, 1997, 2000).

Hypothesis 7

$H_a^2$: The correlation between Process Maturity and software maintenance cost will be higher for organizations with mature processes than for organizations with less mature processes for developing software.

In order to make this comparison, the data had to be split between those respondents who scored lower than 0.5 on the PPMAT
scale and those who scored 0.5 and above, with 0.5 as the median. Table 32 shows the correlation results for the effects of PPMAT on software maintenance effort for the two groups.

Table 32: Comparison of Corr Coeff. for PPMAT and EFFORT

<table>
<thead>
<tr>
<th></th>
<th>PPMAT</th>
<th>RBI (Effort)</th>
<th>PPMAT</th>
<th>RBI (Effort)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Project Process Maturity (PPMAT)</td>
<td>1.00</td>
<td></td>
<td>1.00</td>
<td></td>
</tr>
<tr>
<td>Real Business Impact/Total Effort (RBI)</td>
<td>-.22*‡</td>
<td>1.00</td>
<td>.13†</td>
<td>1.00</td>
</tr>
</tbody>
</table>

Note: n=60, *p >.05, ‡p <.10, (2-tailed) (organizations with low PPMAT) n=39, †p >.05, (2-tailed) (organizations with high PPMAT)

As shown in Table 32, a Spearman rho correlation coefficient was calculated for the relationship between project process maturity and total software maintenance effort for those respondents who scored lower on the project process maturity scale and those who scored higher. For those who scored lower, a weak negative correlation that was not significant at the .05 level was found (r (58) = -.22, p > .05). However, the correlation was significant at the .10 level.
For those who scored higher on the project process maturity scale (see Figure 14), a weak positive correlation that was not significant was also found ($r (37) = .13, p > .05$). The correlation between PPMAT and Effort for organizations with less mature processes is actually higher ($r = -.22$) than for organizations with higher mature processes ($r = .13$) for developing software, which does not support $H_a2$.

Hypothesis 8
$H_b2$: There is a negative correlation between organizations that recognize software maintenance cost as areas to gain competitive advantage and software maintenance cost.
Table 33: Correlation Coeff. for Org Factors and EFFORT

<table>
<thead>
<tr>
<th></th>
<th>ORG5</th>
<th>ORG6</th>
<th>ORG7</th>
<th>EFFORT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Need to Reduce SW Maint Cost (ORG5)</td>
<td>1.00</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Recognizes SW Maint Cost As Areas To Gain Competitive Advantage (ORG6)</td>
<td>.76**</td>
<td>1.00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Values SW Maint Cost Reduction (ORG7)</td>
<td>.71**</td>
<td>.77**</td>
<td>1.00</td>
<td></td>
</tr>
<tr>
<td>Total SW Maint. Effort (EFFORT)</td>
<td>-.09†</td>
<td>.02†</td>
<td>-.06†</td>
<td>1.00</td>
</tr>
</tbody>
</table>

Note: n=99, **p < .01, †p > .05, (2-tailed)

As Table 33 shows, a Spearman rho correlation coefficient was calculated for the relationship between organizational factors and total software maintenance effort. While weak negative correlation that was not significant was found for other organizational factors (Org5 and Org7), a positive correlation that was not significant was calculated for organizations that recognize Software Maintenance Cost as an area to Gain Competitive Advantage (ORG6) (r (97)=.02, p>.05). This does not support H\textsubscript{b2}, so H\textsubscript{b2} is rejected.

Hypothesis 9

H\textsubscript{b3}: Software developers do not spend a significant amount of their time doing software maintenance.
Table 34: Frequency distribution of Type of Current Work

<table>
<thead>
<tr>
<th>Type Of Curr Work</th>
<th>Frequency</th>
<th>Percent</th>
</tr>
</thead>
<tbody>
<tr>
<td>New Development</td>
<td>26</td>
<td>26.3</td>
</tr>
<tr>
<td>Software Maintenance</td>
<td>4</td>
<td>4.0</td>
</tr>
<tr>
<td>Both New Development &amp; Maintenance</td>
<td>63</td>
<td>63.6</td>
</tr>
<tr>
<td>N/A</td>
<td>6</td>
<td>6.1</td>
</tr>
<tr>
<td>Total</td>
<td>99</td>
<td>100.0</td>
</tr>
</tbody>
</table>

Figure 15. Type of Current Work by Software Developers

From Table 34 and Figure 15, only 4 percent of the respondents considered themselves as doing purely software maintenance work, while 26 percent identified new software development as their main type of software work. Over 63 percent said that they do both. The highest percentage of respondents doing software maintenance could be at most 67.6 (4 + 63.6),
while the highest percentage of those that identified themselves as doing new software development could be 89.6% (26 + 63.6). So it appears that software developers do not spend a significant amount of their time on software maintenance.

Table 35: Frequency distribution of % Time on Maintenance

<table>
<thead>
<tr>
<th>Frequency</th>
<th>Percent</th>
<th>Percent</th>
</tr>
</thead>
<tbody>
<tr>
<td>0% - 20%</td>
<td>54</td>
<td>54.5</td>
</tr>
<tr>
<td>21% - 40%</td>
<td>20</td>
<td>20.2</td>
</tr>
<tr>
<td>41% - 60%</td>
<td>16</td>
<td>16.2</td>
</tr>
<tr>
<td>61% - 80%</td>
<td>5</td>
<td>5.1</td>
</tr>
<tr>
<td>81% - 100%</td>
<td>4</td>
<td>4.0</td>
</tr>
<tr>
<td>Total</td>
<td>99</td>
<td>100.0</td>
</tr>
</tbody>
</table>

Figure 16. Percent of Time on Maintenance by Developers
Also from Table 35 and Figure 16, over 54 percent of respondents identified themselves as spending zero to twenty percent of their time on maintenance. Less than 46 percent said they spend more than 20 percent of their time on software maintenance. Both pieces of information support $H_03$ making it possible not to reject the hypothesis as stated by $H_03$.

Hypothesis 10

$H_a3$: Software developers spend a significant amount of their time on software maintenance than they admit.

As Tables 34 and 35 show, software developers do not spend majority of their time on software maintenance, which does not support $H_a3$ as stated. Therefore, $H_a3$ has to be rejected.

Hypothesis 11

$H_04$: Majority of an organization’s software maintenance effort is not spent doing Corrective and Perfective maintenance.
Figure 17. Types of Maintenance Performed by Organizations

Table 36: Frequency distribution of Types of Maintenance

<table>
<thead>
<tr>
<th>Amount of Time Spent on Different Types of Maintenance (in %)</th>
<th>Corrective Maint</th>
<th>Adaptive Maint</th>
<th>Perfective Maint</th>
<th>Preventive Maint</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 - 20%</td>
<td>74.70</td>
<td>73.70</td>
<td>37.40</td>
<td>78.80</td>
</tr>
<tr>
<td>21 - 40%</td>
<td>17.20</td>
<td>17.20</td>
<td>34.30</td>
<td>18.20</td>
</tr>
<tr>
<td>41 - 60%</td>
<td>6.10</td>
<td>7.10</td>
<td>17.20</td>
<td>1.00</td>
</tr>
<tr>
<td>61 - 80%</td>
<td>1.00</td>
<td>2.00</td>
<td>6.10</td>
<td>1.00</td>
</tr>
<tr>
<td>81 - 100%</td>
<td>1.00</td>
<td>0.00</td>
<td>4.00</td>
<td>1.00</td>
</tr>
</tbody>
</table>
As Figure 17 and Table 36 show, 25.3% of respondents said that more than 20 percent of the software maintenance work they do are corrective maintenance. 26.3% said that more than 20 percent of the work they do are adaptive maintenance. 61.6% said that more than 20 percent of the work they do are Perfective maintenance. 21.2% attribute more than 20 percent of the work they do to preventive maintenance. But as a proportion of overall effort expended on maintenance, the distribution as shown in Figure 18 is Preventive Maintenance (16%), Corrective Maintenance (19%), Adaptive Maintenance (19%), and Perfective Maintenance (46%). Based on this, majority of the maintenance
work effort that organizations do is spent on Adaptive, Corrective and Perfective maintenance, which does not support $H_0 4$. Therefore, $H_0 4$ must be rejected.

Hypothesis 12

$H_a 4$: Majority of an organization’s software maintenance effort is spent doing Corrective and Perfective maintenance.

Based on Figure 24, the study respondents spend equal amount of effort on corrective and adaptive maintenance, with perfective maintenance taking most of their software maintenance effort. For this reason, the statement that majority of the software maintenance effort that organizations do is spent on Corrective and Perfective maintenance is true, which supports $H_a 4$ as stated, and therefore must not be rejected.

Summary of Findings

This study evaluated the impact of software developers’ attitudes on software maintenance cost utilizing self-reported data collected from 99 software developers and software project managers representing six divisions of a Fortune 500 company. The study involved a descriptive correlational design using a well-tested model (Technology Acceptance Model) that is rooted in a well-established theory of social psychology, the Theory of Reasoned Action. All software developers and software project managers completed a 78-item, Web-based questionnaire that
measured four factors: influence of personnel attitude, organizational influences, influence of project characteristics, influence of process maturity of an organization, and demographics.

Four questions were answered, which resulted in twelve hypotheses being tested to determine various unknowns, which included: (1) Is there a correlation between software developer’s attitude towards software improvement best practices and the cost of software maintenance? (2) Does a relationship exist between the process maturity of a software development organization, and software maintenance cost? (3) What percentage of time do software developers spend doing software maintenance? and (4) Where do organizations spend most of their total maintenance time, is it in Adaptive, Corrective, Perfective or Preventive maintenance? These four questions resulted in twelve hypotheses that were tested to answer the questions including determining if a correlation exists between software developers’ attitudes towards software engineering best practices and software maintenance cost.

**Hypotheses**

There were a total of twelve hypotheses that were tested, four of them were null hypotheses. The first hypothesis, which
is a null hypothesis states: There is no correlation between software developer’s attitude towards software process improvement practices and software developer’s perceived software maintenance cost. The relationship between software developer’s attitude towards usage of software improvement best practices and the software developer’s perceived software maintenance effort was tested using Spearman rho correlation coefficient. A weak correlation that was not significant was found ($r (97)=.12, p > .05$), so this null hypothesis is not rejected.

The second hypothesis states: As the developer’s perceived usefulness ($U$) of software process improvement and the developer’s perceived ease of use (EOU) of software best practices increase ($U$, EOU), developer attitude (A) and the developer’s behavioral intentions (BI) to use those software best practices will increase. There were two parts to this test -- (a) predicting software developers’ attitudes toward using software best practices based on their perceived usefulness and perceived ease of use, and (b) predicting software developers’ behavioral intentions to use software best practices based on their perceived usefulness and perceived ease of use. A multiple linear regression was calculated and in both cases, significant
regression equations were found. So the second hypothesis was accepted.

The third hypothesis states: As developer attitude \((A)\) towards software best practices and the developer’s perceived usefulness \((U)\) of software process improvement increase \((A, U)\), the behavioral intentions \((BI)\) of the developer to use software best practices will increase. A multiple linear regression was calculated to predict software developers’ behavioral intentions to use software best practices based on their attitudes toward using software best practices and perceived usefulness. A significant regression equation was found supporting the hypothesis. So, the third hypothesis is accepted.

The fourth hypothesis states: As the developer’s behavioral intentions \((BI)\) to use software best practices increase, the tendency of the developer to use software best practices \((AU)\) will increase. A simple linear regression was calculated to predict software developers’ actual usage of software best practices based on their behavioral intentions. A significant regression equation was found supporting the hypothesis, so the fourth hypothesis was accepted.

The fifth hypothesis states: A correlation between software developer’s attitude towards software process improvement practices and perceived software maintenance cost \((RBI)\) will be
higher for those with high actual usage of the software best practices than those with low usage of the software best practices. This test required that the data be split representing those respondents with low actual usage of software best practices, and those with high actual usage of software best practices. So using the median, the data was split in half 49/50 before calculating the correlation for each half. A Spearman rho correlation coefficient was calculated for each half. A weak correlation that was not significant was found for both, so the fifth hypothesis had to be rejected. This finding is also consistent with the first null hypothesis.

The sixth hypothesis, which is also the second null hypothesis states: There is no difference in the correlation between Process Maturity and software maintenance cost for organizations with mature processes and organizations with less mature processes for developing software. To test this, a Spearman rho correlation coefficient was calculated for the relationship between project process maturity and total software maintenance effort. A weak negative correlation, but nonetheless a correlation, was found also confirming what other studies have concluded about the effects of process maturity on software development effort. The negative correlation indicates that the lower the project process maturity of an organization, the more
effort it spends on software maintenance, not supporting the second null hypothesis, so the null hypothesis was rejected.

The seventh hypothesis states: The correlation between Process Maturity and software maintenance cost will be higher for organizations with mature processes than for organizations with less mature processes for developing software. This test required that the data be split between those respondents who scored lower than 0.5 on the project process maturity (PPMAT) scale and those who scored 0.5 and above, with 0.5 as the median. For those who scored lower, a weak negative correlation that was not significant at the .05 level was found. However, the correlation was significant at the .10 level. Those who scored higher on the project process maturity scale also had a weak correlation that was not significant. The correlation between PPMAT and Effort for organizations with less mature processes was actually higher than for organizations with higher mature processes for developing software, which did not support the hypothesis. So the seventh hypothesis was rejected.

The eight hypothesis states: There is a negative correlation between organizations that recognize software maintenance cost as areas to gain competitive advantage and software maintenance cost. For this test, a Spearman rho correlation coefficient was calculated to test the relationship
between organizational factors and total software maintenance effort. A negative correlation that was not significant was found for other organizational factors, and a positive, non-significant correlation was calculated for organizations that recognize Software Maintenance Cost as an area to Gain Competitive Advantage. This did not support the hypothesis and was therefore rejected.

The ninth hypothesis, which is the third null hypothesis states: Software developers do not spend a significant amount of their time doing software maintenance. A simple frequency distribution showed that only 4 percent of the respondents considered themselves as doing purely software maintenance work, while 26 percent identified new software development as their main type of software work. Over 63 percent said that they do both. Based on the results, the highest percentage of respondents that could possibly be doing software maintenance as their main type of work would be 67.6%, while the highest percentage of those that could possibly be doing new software development would be 89.6%. Based on that, it appeared that majority of software developers do not spend a significant amount of their time on software maintenance. So this null hypothesis is not rejected.
The tenth hypothesis states: Software developers spend a significant amount of their time on software maintenance than they admit. Using the same results obtained for the ninth hypothesis, it was concluded that software developers do not spend majority of their time on software maintenance, which did not support the hypothesis as stated. So, the tenth hypothesis is rejected.

The eleventh hypothesis, which is the fourth null hypothesis states: Majority of an organization’s software maintenance effort is not spent doing Corrective and Perfective maintenance. For this test, a simple frequency distribution was adequate to test it. As a proportion of overall effort expended on maintenance, it was found that software developers spend 16% of their effort on Preventive Maintenance, 19% on Corrective Maintenance, 19% on Adaptive Maintenance, and 46% on Perfective Maintenance. Based on this, majority of the maintenance work effort that organizations do is spent on Adaptive, Corrective and Perfective maintenance. Since the proportion of effort spent on Adaptive and Corrective Maintenance are equal, which was not in agreement with the null hypothesis, so the fourth null hypothesis is rejected.

The twelfth hypothesis states: Majority of an organization’s software maintenance effort is spent doing
Corrective and Perfective maintenance. Using the same results that were applied to the eleventh hypothesis, it is clear that majority of the maintenance work effort that organizations do is spent on Corrective and Perfective maintenance. So, the twelfth hypothesis is accepted.

Discussion of Statistical Findings

Some interesting and important findings were found from this study. While actual usage of software engineering best practices is theorized (Hypothesis 1) to have some relationship with software maintenance cost, the empirical observations on software developers’ attitudes towards usage of best practices did not show a significant relationship. No definitive explanation for this apparent lack of relationship was found. It is possible that the full effect of software developer attitude is yet to be determined because of some inherent biases that may have affected this study.

One possible bias is the reliance of the study on self-reporting of software maintenance effort data by the software developers. For example, the non-significant correlation of actual use of software best practices and software maintenance cost may have been caused by the respondents’ lack of knowledge of how much time they actually spend on software maintenance activities. When the self-reported effort data for one project
was compared to the actual effort data for that project, there was a significant difference between what was self-reported and the actual. While respondents indicated that they spend 19% of their time on corrective maintenance, the actual corrective maintenance effort for one project only showed 5%.

Regarding Hypothesis 5, the correlation between software developer’s attitude towards software process improvement practices and software maintenance cost was almost identical for those with low actual usage of the software best practices and those with high usage of the software best practices. This result may have been because the two groups were not separated from the beginning and then surveyed separately. But then, that would have been almost impossible to do as no organization would admit upfront that their actual usage of software best practices is low.

With respect to Hypothesis 7, the expectation was that the correlation between process maturity and software maintenance cost would be significant even at the .05 (95%) level for those organizations whose project processes are more mature. Instead, a significant correlation was found at the .10 (90%) level. Again, this finding may also be because the two groups were not delineated from the beginning and surveyed separately.
Regarding Hypothesis 8, while negative, non-significant correlation was found for other organizational factors, a positive, non-significant correlation was calculated for organizations that recognize Software Maintenance Cost as an area to Gain Competitive Advantage. One explanation for this finding could be that software practitioners do not perceive that their organizations recognize software maintenance cost reduction as an area to gain competitive advantage.

With respect to Hypothesis 10, software developers indeed do not spend a significant amount of their time on software maintenance, at least that is what the result shows. But this finding could have been biased as well based on software developers’ perception of what software maintenance is. Although four different software maintenance categories were defined to help software developers distinguish software maintenance activities and new software development activities, preconceived notions about software maintenance could have played a role here.

Regarding Hypotheses 11 and 12, it was shown that contrary to popular opinion, respondents in these software development organizations do actually spend as much time on adaptive maintenance as they do on corrective maintenance. Most of the maintenance effort is spent doing perfective maintenance. For
this study, the average distribution of effort across maintenance change types is 19% Correction, 19% Adaptation, 46% Perfection, and 16% Prevention.

In terms of external validity or generalizability, this study has not fully addressed all the issues using software developer populations whose understanding of software maintenance compared to the general population may limit the extent of the study’s external validity. For example, the non-significant correlation of software developer’s attitude and software maintenance effort might have been because most software developers like to be identified with new software development instead of software maintenance, which may have affected the way they reported the time they spent on software maintenance.
Summary

This study examined the effects of software developers’ attitudes on software maintenance effort. The study was a five-week data gathering that involved self-reporting of individual attitudes towards software improvement best practices and projects data. The instrument used was a software maintenance and developer attitude survey (SMADAS) that was designed, pilot-tested and validated by the researcher. The questionnaires made use of the Likert Scale with multiple means of assessing the effect of developer attitude on software maintenance cost. The questionnaire comprised of four sub-scales, with each scale representing a component of software developer attitude and software maintenance cost.

Twelve hypotheses (four null hypotheses and 8 alternative hypotheses) were tested to answer four research questions using a combination of descriptive and non-parametric techniques. The first hypothesis is not rejected. There was no significant correlation between software developer’s attitude towards software process improvement practices and software developer’s perceived software maintenance cost. This result was not anticipated because it was thought that software developers who engaged in software improvement best practices would be creating maintainable software, which would lead to less effort being spent on software maintenance.
The second hypothesis states: As the developer’s perceived usefulness (U) of software process improvement and the developer’s perceived ease of use (EOU) of software best practices increase (U, EOU), developer attitude (A) and the developer’s behavioral intentions (BI) to use those software best practices will increase. The second hypothesis is not rejected. A significant regression equation for predicting attitudes towards using software best practices and behavioral intentions to use those best practices was found, given developer’s perceived usefulness and ease of use.

Table 37: Summary of primary findings

<table>
<thead>
<tr>
<th>Test</th>
<th>Sig.</th>
</tr>
</thead>
<tbody>
<tr>
<td>H₀₁: Not Rejected</td>
<td>r(97) = .12, p &gt; .05</td>
</tr>
<tr>
<td>Hₐ₁: Not Rejected</td>
<td>F(2, 96) = 39.27, **p &lt; .001; R² = 0.45 (U+EOU, A); F(2, 96) = 33.04, **p &lt; .001; R² = 0.41 (U+EOU, BI);</td>
</tr>
<tr>
<td>H₀₂: Not Rejected</td>
<td>F(2, 96) = 35.43, **p &lt; .001; R² = 0.43 (A+U, BI)</td>
</tr>
<tr>
<td>Hₐ₂: Not Rejected</td>
<td>F(1, 97) = 10.89, **p &lt; .001; R² = 0.10 (BI, AU)</td>
</tr>
<tr>
<td>H₀₃: Rejected</td>
<td>r(47) = .14, p &gt; .05; r(48) = -.13, p &gt; .05;</td>
</tr>
<tr>
<td>Hₐ₃: Rejected</td>
<td>r(47) = -.20, *p &lt; .05</td>
</tr>
<tr>
<td>H₀₄: Rejected</td>
<td>r(58) = -.22, p &gt; .05, †p &lt; .10; r(37) = .13, p &gt; .05</td>
</tr>
<tr>
<td>Hₐ₄: Rejected</td>
<td>r(97) = .02, p &gt; .05</td>
</tr>
<tr>
<td>H₀₅: Not Rejected</td>
<td>NDW (89.6%) &gt; MW (67.6%)</td>
</tr>
<tr>
<td>Hₐ₅: Rejected</td>
<td>NDW (89.6%) &gt; MW (67.6%)</td>
</tr>
<tr>
<td>H₀₆: Not Rejected</td>
<td></td>
</tr>
<tr>
<td>Hₐ₆: Not Rejected</td>
<td></td>
</tr>
</tbody>
</table>

Note: The highlighted tests are the null hypotheses. NDW = New Development Work, MW = Maintenance Work. S = Significant, NS = Not Significant. ** = p < .01, * = p < .05, † = p < .10.
The third hypothesis states: As developer attitude (A) towards software best practices and the developer’s perceived usefulness (U) of software process improvement increase (A, U), the behavioral intentions (BI) of the developer to use software best practices will increase. This hypothesis is not rejected. A significant regression equation was found supporting the hypothesis.

The fourth hypothesis states: As the developer’s behavioral intentions (BI) to use software best practices increase, the tendency of the developer to use software best practices (AU) will increase. The fourth hypothesis is not rejected. A significant regression equation was also found supporting the hypothesis.

The fifth hypothesis states: A correlation between software developer’s attitude towards software process improvement practices and perceived software maintenance cost (RBI) will be higher for those with high actual usage of the software best practices than those with low usage of the software best practices. The fifth hypothesis is rejected. A non-significant, weak correlation was found. This finding is also consistent with the first null hypothesis.

The sixth hypothesis (also the second null hypothesis) states: There is no difference in the correlation between Process Maturity and software maintenance cost for organizations
with mature processes and organizations with less mature processes for developing software. The second null hypothesis is rejected. A weak negative correlation was found indicating that the lower the process maturity of an organization, the more effort its developers spend on software maintenance.

The seventh hypothesis states: The correlation between Process Maturity and software maintenance cost will be higher for organizations with mature processes than for organizations with less mature processes for developing software. This test required that the data be split between those respondents who scored lower than 0.5 on the project process maturity (PPMAT) scale and those who scored 0.5 and above, with 0.5 as the median. For those who scored lower, a weak negative correlation that was not significant at the .05 level was found. However, the correlation was significant at the .10 level. Those who scored higher on the project process maturity scale also had a weak correlation that was not significant. The correlation between PPMAT and Effort for organizations with less mature processes was actually higher than for organizations with higher mature processes for developing software, which did not support the hypothesis. So the seventh hypothesis was rejected.

The eight hypothesis states: There is a negative correlation between organizations that recognize software maintenance cost as areas to gain competitive advantage and
Impact of developer attitude software maintenance cost. For this test, a Spearman rho correlation coefficient was calculated to test the relationship between organizational factors and total software maintenance effort. A negative correlation that was not significant was found for other organizational factors, and a positive, non-significant correlation was calculated for organizations that recognize Software Maintenance Cost as an area to Gain Competitive Advantage. This did not support the hypothesis and was therefore rejected.

The ninth hypothesis (which is also the third null hypothesis) states: Software developers do not spend a significant amount of their time doing software maintenance. The third null hypothesis is not rejected. It was found that the highest percentage of respondents (89.6%) identified themselves as doing new software development compared to 67.6% that identified their work as software maintenance.

The tenth hypothesis states: Software developers spend a significant amount of their time on software maintenance than they admit. This hypothesis is rejected. It was concluded that software developers spend majority of their time on new software development activities and less time on software maintenance activities.

The eleventh hypothesis (also the fourth null hypothesis) states: Majority of an organization’s software maintenance
effort is not spent doing Corrective and Perfective maintenance. The fourth null hypothesis is rejected. It was found that software developers spend majority of their time doing Corrective (19%) and Perfective (46%) software maintenance, with Adaptive (19%) and Preventive (16%) Maintenance very close behind.

The twelfth hypothesis states: Majority of an organization’s software maintenance effort is spent doing Corrective and Perfective maintenance. The twelfth hypothesis is not rejected. Using the same results that were applied to the eleventh hypothesis, it was concluded that majority of the maintenance work effort that organizations do is spent on Corrective and Perfective maintenance.

Conclusions

The main conclusions from this study are as follows:

1. The software developers and software project managers that responded to this study did not report significant impact of developer attitudes towards software improvement best practices on the software developer’s perceived software maintenance cost. Although all the intervening variables showed strong interrelationship, none showed any significant correlation with software maintenance effort. There are several reasons that could have contributed to this result, including product characteristics, software development processes used, the dynamics of the development team, and other environmental factors. However, the most likely reason for the non-significant finding in the correlation between developer attitude and software maintenance cost is the reliance of the study on self-reporting of effort data by the developers. It appears that self-reporting of software maintenance effort data by
software developers was not realistic enough to show true relationship between developer attitude and software maintenance cost. This conclusion is based on the analysis of the maintenance effort data reported for one project and the actual corrective maintenance effort data for that project. When the actual corrective maintenance effort data was compared to the self-reported effort data, the actual corrective maintenance effort data was just 5% of the overall effort data reported by the software developers. Given that software developers had reported that 19% of their time is spent on corrective maintenance, and the actual effort indicates 5%, one could not conclude one way or the other the reliability of the self-reported effort data.

2. Software developers and software project managers would adopt and use software improvement best practices if they perceive them to be useful and easy to use. It was shown that both perceived ease of use (b=.63, p<.01) and perceived usefulness (b=.61, p<.01) contributed strongly to attitude toward using software improvement best practices, with perceived ease of use being the most influential. In addition, perceived usefulness (b=.58, p<.01) and attitude toward using (b=.59, p<.01) were almost identical in their influences on intention to use software improvement best practices, however, attitude toward using was slightly more influential.

3. A relationship exists between an organization’s software process maturity level and software maintenance cost. A negative correlation between an organization’s project process maturity and software maintenance effort, indicating that the lower an organization scores on the project process maturity scale, the more effort it spends on software maintenance. This conclusion reinforces the research done by Clark (1997) related to effect of process maturity on effort, although that study was primarily about software development effort.

4. Software developers spend as much time on corrective and perfective maintenance as they do on adaptive and perfective maintenance. The sampled software developers reported that 19% of their effort is spent on Corrective Maintenance, 19% on Adaptive Maintenance, and 46% on Perfective Maintenance. Despite that many software developers do not like to identify with software
maintenance activities, 67.7% of work performed by the respondents are about software maintenance. Most of the maintenance works are about perfection, correction and adaptation.

5. This study identified software developer attitude as a possible factor and also attempted to quantify its effect on software maintenance cost. However, the study did not show a significant relationship between software developer attitude and software maintenance cost. Although the study failed to show a connection between software developer attitude and software maintenance cost, it was a good first step in quantifying the developer behavioral aspect of software development. Recommendations on how to improve the study are provided later in this Chapter.

Implications for Practice

This research has provided answers to the research questions including showing that software developer attitude towards process improvement practices for at least this study population, does not have any effect on software maintenance cost. It suggests that software development organizations are not paying close attention to software maintenance activities as most respondents indicated that their organizations did not value reducing software maintenance cost (61.6%) or see software maintenance as an area to gain competitive advantage (72.7%).

It was out of scope for this study to obtain the details to explain or understand why over 61% of the software developers felt that their organizations do not value reducing software maintenance cost. The researcher could only assume that organizations have either not clearly communicated their
software maintenance cost reduction intentions to these software
developers or the organizations indeed have not identified
maintenance cost as a strategic area to reduce cost. In either
case, this finding suggests that software organizations should
seriously review their software maintenance activities,
determine how much software maintenance is costing them and make
improvements where necessary.

This research indicates that 67.6% of the respondents
perform some type of software maintenance. These software
maintenance activities are further broken down as follows:
Perfective maintenance (46%), Corrective maintenance (19%),
Adaptive maintenance (19%), and Preventive maintenance (16%).
This is crucial for software development organizations as
majority of software developers identify more with new
development activities than with software maintenance
activities. However, as this research showed, over 67% of
software developers do engage in software maintenance
activities. Since software developers do indeed spend a lot of
time on maintenance, designing for maintainability becomes a
virtue for these development organizations.

This research used the Technology Acceptance Model and
extended it for this study. While the overall real business
impact could not be predicted using the extended model, the
variables that affect developer’s attitude were examined and
shown to predict actual use of the software engineering best practices. Using software engineering best practices to design and build software products is a good indicator of a process mature organization. And as this study showed, there is a negative correlation between project process maturity and software maintenance effort. That implies that the higher a software organization is on the project process maturity scale, the less it spends on software maintenance effort.

Recommendations

This research showed that organizations that exhibit high project process maturity tend to spend less time on software maintenance cost (effort). This study also concluded that software developers and software project managers would adopt and use software improvement best practices if they perceive them to be useful and easy to use. Based on these findings, it is recommended that organizations engage in software process improvement practices that are useful and easy to use to increase their project process maturity levels.

Based on the results of this study, it is also the recommendation of the researcher that the questionnaire be reviewed and re-calibrated for applicability to measuring software developer attitude prior to future implementation on software developers. Several responses (35%) were thrown out for
incomplete response. These included people that did not meet the criteria of being either a software developer or software project manager, as well as properly filling out the survey. Several responses were rejected because respondents either refused to provide the data that makeup software maintenance effort (i.e., duration in months, number of hours per person-month, project team size), or they did not understand the question. Many responses were also thrown out because questions about project process maturity were not provided. Although there may not be anything wrong with the instrument, it is recommended that it be reviewed again before any future use.

This research indicated that no significant correlation exists between software developer attitude towards software improvement best practices and the software developer’s perceived software maintenance cost. This finding could have been due to self-reporting of effort data by software developers. The researcher recommends that any future research in this area be based on actual effort data maintained by software development organizations. The researcher further recommends that a sample of specific projects be identified. Effort data would then be gathered from those specific projects and the survey should then be administered to the software developers that work on those projects. This would eliminate the
inherent bias associated with self-reporting of effort data by study participants.

Although the sample from the population of software developers was adequate for the findings to be statistically significant, the respondents were all from a single metropolitan region. The researcher recommends that future researches on this topic be extended to other regions to improve generalization.

Recommendations for Further Research

This research simply opened the door for investigations into the attitudinal and behavioral impact of software developers on software maintenance cost. This study has attempted to show software developer attitude as an antecedent to software maintenance cost. However, the self-reported effort data did not support that hypothesis. Therefore, an important focus of any future study in this area is to base effort data gathering on direct reporting of actual effort data by the software organizations.

Another thing that can be done to accurately assess the effect of software developer attitude on software maintenance cost would be to conduct an experiment and observe the software developers at work and use actual effort data reported by the participating organizations.
Other important areas for further studies include evaluating the impact of software developer attitudes on software development productivity and exploring the effect of software developer attitude on performance and software quality.
REFERENCES


Impact of developer attitude


Impact of developer attitude


Impact of developer attitude


This pilot survey is the final step of the researcher’s effort to validate this survey instrument to be used in the evaluation of the impact of developers’ attitudes on software maintenance cost for his doctoral dissertation in Information Technology Management. Although your participation in this study is completely voluntary, your response will contribute immensely to the significance of this research. To ensure confidentiality, no name has been requested in the survey and your responses will remain confidential.

The actual time required to complete this survey is approximately 15-20 minutes. When you finish, please fold the survey in half, staple it and drop it in a mailbox as it is already self-addressed and stamped. Please return the completed survey by January 8, 2003. Thank you in advance for your assistance.
Thank you for participating in this survey. The researcher would like to know how long it took you to complete this questionnaire. If you are interrupted while answering these questions, please note the elapsed time and deduct it from the overall time it took to complete the survey. Thanks again.

This survey contains 4 parts. It is estimated to take about 15 to 20 minutes to complete.

Read each statement and then check the box or circle the answer that best describes you or shows how you feel.

PART 1: DEMOGRAPHIC QUESTIONS

Instructions: Circle the answer that best describes you.

1. My gender? 1=Female, 2=Male  

2. A Developer is a person who builds (designs, codes, tests, debugs, documents, and maintains) computer programs. A Software Project Manager is a person who plans, monitors, and controls people and processes involved in the creation of software products. Based on these definitions, how would you describe yourself? 0=Neither 1=Developer 2=Software Project Manager 3=Both

3. Number of years in your current position? 1=0-2 yrs, 2=3-5 yrs, 3=6-8 yrs, 4=9-11 yrs, 5=12+ yrs

4. How many years have you been in the software profession? 1=0-2 yrs, 2=3-5 yrs, 3=6-8 yrs, 4=9-11 yrs, 5=12+ yrs

5. What type of software work are you currently doing? 1=new development, 2=maintenance, 3=both, 4=not applicable.

6. Percentage of my time spent doing software maintenance activities: 1=0-20%, 2=21-40%, 3=41-60%, 4=61-80%, 5=81-100%
### PART 2: PERSONNEL ATTITUDE QUESTIONS

**Instructions:** Select one level of agreement for each statement to indicate how you feel.

<table>
<thead>
<tr>
<th></th>
<th>SD</th>
<th>D</th>
<th>N</th>
<th>A</th>
<th>SA</th>
</tr>
</thead>
<tbody>
<tr>
<td>7.</td>
<td>I enjoy doing my work.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8.</td>
<td>Learning about process improvement models such as CMM is a waste of time.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9.</td>
<td>I think documenting my design makes me spend more time to finish my project assignment.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10.</td>
<td>I think documenting my code makes me spend more time to finish my project assignment.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11.</td>
<td>I think it is important to document requirements before design or coding begins.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12.</td>
<td>I make a plan before I start to solve a problem.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>13.</td>
<td>I see the use of software process as adding value to my project.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>14.</td>
<td>I choose my own way without imitating or worrying about methods used by other members of my team.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>15.</td>
<td>I would like to learn more about software process improvement activities.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>16.</td>
<td>In my opinion, software process improvement activities are too much trouble.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>17.</td>
<td>I see software development standards as something I will rarely use in my implementation of software.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>18.</td>
<td>Using a standard to develop software prevents me from being creative.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>19.</td>
<td>Employing software best practices improves the overall quality of software.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>20.</td>
<td>Software process improvement activities would increase my overall productivity.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>21.</td>
<td>Documented process standards are an evil that should be avoided as much as possible.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>22.</td>
<td>In my opinion, software maintenance is the most important activity in the software life cycle.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>23.</td>
<td>In my opinion, software maintenance is the least important activity in the software life cycle.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### PART 3: ORGANIZATION QUESTIONS

**Instructions:** Select one level of agreement for each statement to indicate how you feel about your organization.

<table>
<thead>
<tr>
<th></th>
<th>SD</th>
<th>D</th>
<th>N</th>
<th>A</th>
<th>SA</th>
</tr>
</thead>
<tbody>
<tr>
<td>24.</td>
<td>My organization maintains a set of standard software processes for developing software.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>25.</td>
<td>My organization encourages me to follow its standard software process to develop software.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>26.</td>
<td>My organization provides adequate funding to make software process improvement happen.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>27.</td>
<td>My organization has a group or individual(s) responsible for the organization’s software process activities.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>28.</td>
<td>In my opinion, my organization recognizes the need to reduce software maintenance cost.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>29.</td>
<td>My organization sees software maintenance cost reduction as a way to gain competitive advantage.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Circle the answer that best describes your organization’s software process improvement capability.

30. My organization follows the Software Engineering Institute’s Capability Maturity Model as a process for Software Improvement. Yes No

31. My organization’s overall Process Maturity Level is (1=CMM Level 1; 2=CMM Level 2; 3=CMM Level 3; 4=CMM Level 4; 5=CMM Level 5, NA = Not Applicable). 1 2 3 4 5 NA

32. My answer to the above question is based on (BOAS): (1= Internal Audit, 2 = Mini-Assessment, 3 = Interim Process Assessment, 4 = Software Capability Assessment, 5 = Software Process Assessment, NA = Not Applicable. 1 2 3 4 5 NA

### PART 4: PROJECT AND PRODUCT QUESTIONS

#### 4.1: Your Project Information

**Instructions:** Provide the required project data and circle one level of agreement for each statement that indicates the amount of your time spent on these.

33. Name or describe a project or sub-system that you’re working on (or have worked on in the past 6 months) that has been released to the customer for use. ________________

34. What percentage of your time do you spend making changes that occur due to actual changes that occur due to actual problems discovered after the system was released for use? 1=0-20 %, 2=21-40%, 3=41-60%, 4=61-80%, 5= 81-100% 1 2 3 4 5

35. What percentage of your time do you spend on modification or upgrading of the software so that it will work properly in a new 1 2 3 4 5
36. What percentage of your time do you spend on making changes or enhancements to the software to meet the evolving needs of the system? 1=0-20%, 2=21-40%, 3=41-60%, 4=61-80%, 5=81-100%

37. What percentage of your time do you spend on improving the software for future maintainability to provide a better basis for future enhancements? 1=0-20%, 2=21-40%, 3=41-60%, 4=61-80%, 5=81-100%

38. Provide the effort in Person Months associated with development and test of the software component you described above since the project was released for use.

39. Indicate the average number of hours (per person month) spent to-date by your team on this project since it was first released to the customer.

40. How many software developers (including the project manager) are on your project team?

41. Please provide the size measures (the total lines of code) for the project or sub-system that you described above. Indicate if the count was for logical SLOC or physical SLOC. If both are available, please submit both types of counts. If neither type of count applies to the way the code was counted, please describe the method.
**Instructions:** Select one level of agreement for each statement to indicate how you feel about your project.

| SD = Strongly Disagree,  D = Disagree,  N = Neutral,  A = Agree,  SA = Strongly Agree |
|-----------------------------------------------|---|---|---|---|
| 42. Project schedule was compressed so much that requirements were not fully decomposed before implementation began. | SD | D | N | A |
| 43. Software maintainability was clearly identified as an important quality attribute during software development. | SD | D | N | A |
| 44. In my opinion, there is good communication among the team members. | SD | D | N | A |
| 45. Tasks assigned to me are always delivered on or before their scheduled due dates. | SD | D | N | A |
| 46. I find myself spending a lot of time on rework. | SD | D | N | A |
| 47. My project team is well experienced in the language and tools used to develop the software. | SD | D | N | A |
| 48. My project team has had prior experience with the type of software system described above. | SD | D | N | A |
| 49. The product is highly complex. (Note: Product Complexity is a continuum, where complexity of a product similar to a prior application is Low and a product that involves all new innovation is High) | SD | D | N | A |

Go To Next Page ➔
### 4.2: Your Project Key Process Area (KPA) Evaluation*

Enough information is provided in the following table so that you can assess the degree to which a KPA was exercised on your project. Each KPA is briefly described and its goals are given. Check the box that best describes your project team’s software practices. The response categories are explained below:

- **AA = Almost Always** (over 90% of the time) when the goals are consistently achieved and are well established in standard operating procedures.
- **FR = Frequently** (about 60 to 90% of the time) when the goals are achieved relatively often, but sometimes are omitted under difficult circumstances.
- **AH = About Half** (about 40 to 60% of the time) when the goals are achieved about half of the time.
- **OC = Occasionally** (about 10 to 40% of the time) when the goals are sometimes achieved, but less often.
- **RE = Rarely If Ever** (less than 10% of the time) when the goals are rarely if ever achieved.
- **NA = Does Not Apply** when you have the required knowledge about your project or organization and the KPA, but you feel the KPA does not apply to your circumstances (e.g. Subcontract Management).
- **DK = Don’t Know** when you are uncertain about how to respond for the KPA.

<table>
<thead>
<tr>
<th>Key Process Area</th>
<th>Goals of each KPA</th>
<th>AA</th>
<th>FR</th>
<th>AH</th>
<th>OC</th>
<th>RE</th>
<th>NA</th>
<th>DK</th>
</tr>
</thead>
</table>
| 50. Requirements Management: involves establishing and maintaining an agreement with the customer on the requirements for the software project (KPA1). | • System requirements allocated to software are controlled to establish a baseline for software engineering and management use.  
  • Software plans, products, and activities are kept consistent with the system requirements allocated to software.                                                                                                                                                                                                                                                                                            |    |    |    |    |    |    |    |
| 51. Software Project Planning: establishes reasonable plans for performing the software engineering activities and for managing the software project (KPA2). | • Software estimates are documented for use in planning and tracking the software project.  
  • Software project activities and commitments are planned and documented.  
  • Affected groups and individuals agree to their commitments related to the software project.                                                                                                                                                                                                                                                                 |    |    |    |    |    |    |    |
| 52. Software Project Tracking and Oversight: provides adequate visibility into actual progress so that management can take corrective actions when the software project’s performance deviates significantly from the software plans (KPA3). | • Actual results and performances are tracked against the software plans.  
  • Corrective actions are taken and managed to closure when actual results and performance deviate significantly from the software plans.  
  • Changes to software commitments are agreed to by the affected groups and individuals.                                                                                                                                                                                                                                                                 |    |    |    |    |    |    |    |
### Key Process Area

<table>
<thead>
<tr>
<th>53. Software Subcontract Management: involves selecting a software subcontractor, establishing commitments with the subcontractor, and tracking and reviewing the subcontractor’s performance and results (KPA4).</th>
<th><strong>Goals of each KPA</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>• The prime contractor selects qualified software subcontractors.</td>
<td></td>
</tr>
<tr>
<td>• The prime contractor and the software subcontractor agree to their commitments to each other.</td>
<td></td>
</tr>
<tr>
<td>• The prime contractor and the software subcontractor maintain ongoing communications.</td>
<td></td>
</tr>
<tr>
<td>• The prime contractor tracks the software subcontractor’s actual results and performance against its commitments.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>54. Software Quality Assurance: provides management with appropriate visibility into the process being used by the software project and of the products being built (KPA5).</th>
<th><strong>Goals of each KPA</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>• Software quality assurance activities are planned.</td>
<td></td>
</tr>
<tr>
<td>• Adherence of software products and activities to the applicable standards, procedures, and requirements is verified objectively.</td>
<td></td>
</tr>
<tr>
<td>• Affected groups and individuals are informed of software quality assurance activities and results.</td>
<td></td>
</tr>
<tr>
<td>• Senior management addresses noncompliance issues that cannot be resolved within the software project.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>55. Software Configuration Management: establishes and maintains the integrity of the products of the software project throughout the project’s software life cycle (KPA6).</th>
<th><strong>Goals of each KPA</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>• Software configuration management activities are planned.</td>
<td></td>
</tr>
<tr>
<td>• Selected software work products are identified, controlled, and available.</td>
<td></td>
</tr>
<tr>
<td>• Changes to identified software work products are controlled.</td>
<td></td>
</tr>
<tr>
<td>• Affected groups and individuals are informed of the status and content of software baselines.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>56. Organization Process Focus: establishes the organizational responsibility for software process activities that improve the organization’s overall software process capability (KPA7).</th>
<th><strong>Goals of each KPA</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>• Software process development and improvement activities are coordinated across the organization.</td>
<td></td>
</tr>
<tr>
<td>• The strengths and weaknesses of the software processes used are identified relative to a process standard.</td>
<td></td>
</tr>
<tr>
<td>• Organization-level process development and improvement activities are planned.</td>
<td></td>
</tr>
</tbody>
</table>
### Key Process Area

<table>
<thead>
<tr>
<th></th>
<th>Goals of each KPA</th>
</tr>
</thead>
<tbody>
<tr>
<td>57. <strong>Organization Process Definition:</strong> develops and maintains a usable set of software process assets that improve process performance across the projects and provides a basis for cumulative, long-term benefits to the organization (KPA8).</td>
<td></td>
</tr>
<tr>
<td></td>
<td>- A standard software process for the organization is developed and maintained.</td>
</tr>
<tr>
<td></td>
<td>- Information related to the use of the organization’s standard software process by the software projects is collected, reviewed, and made available.</td>
</tr>
<tr>
<td>58. <strong>Training Program:</strong> develops the skills and knowledge of individuals so they can perform their roles effectively and efficiently (KPA9).</td>
<td></td>
</tr>
<tr>
<td></td>
<td>- Training activities are planned.</td>
</tr>
<tr>
<td></td>
<td>- Training for developing the skills and knowledge needed to perform software management and technical roles is provided.</td>
</tr>
<tr>
<td></td>
<td>- Individuals in the software engineering group and software-related groups receive the training necessary to perform their roles.</td>
</tr>
<tr>
<td>59. <strong>Integrated Software Management:</strong> integrates the software engineering and management activities into a coherent, defined software process that is tailored from the organization’s standard software process and related process assets (KPA10).</td>
<td></td>
</tr>
<tr>
<td></td>
<td>- The project’s defined software process is a tailored version of the organization’s standard software process.</td>
</tr>
<tr>
<td></td>
<td>- The project is planned and managed according to the project’s defined software process.</td>
</tr>
<tr>
<td>Key Process Area</td>
<td>Goals of each KPA</td>
</tr>
<tr>
<td>-----------------------------------------------------</td>
<td>------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
</tbody>
</table>
| **60. Software Product Engineering:** integrates all the software engineering activities to produce and support correct, consistent software products effectively and efficiently (KPA11). | - The software engineering tasks are defined, integrated, and consistently performed to produce the software.  
- Software work products are kept consistent with each other.                                                                                         |
| **61. Intergroup Coordination:** establishes a means for the software engineering group to participate actively with the other engineering groups so the project is better able to satisfy the customer’s needs effectively and efficiently (KPA12). | - The customer’s requirements are agreed to by all affected groups.  
- The commitments between the engineering groups are agreed to by the affected groups.  
- The engineering groups identify, track, and resolve Intergroup issues.                                                                               |
| **62. Peer Review:** removes defects from the software work products early and efficiently (KPA13). | - Peer review activities are planned.  
- Defects in the software work products are identified and removed.                                                                                   |
| **63. Quantitative Process Management:** controls the process performance of the software projects quantitatively (KPA14). | - The quantitative process management activities are planned.  
- The process performance of the project’s defined software process is controlled quantitatively.  
- The process capability of the organization’s standard software process is known in quantitative terms.                                                |
| **64. Software Quality Management:** involves defining quality goals for the software products, establishing plans to achieve these goals, and monitoring and adjusting the software plans, software work products, activities, and quality goals to satisfy the needs and desires of the customer and end user (KPA15). | - The project’s software quality management activities are planned.  
- Measurable goals for software product quality and their priorities are defined.  
- Actual progress toward achieving the quality goals for the software products is quantified and managed.                                              |
| **65. Defect Prevention:** analyzes defects that were encountered in the past and takes specific actions to prevent the occurrence of those types of defects in the future (KPA16). | - Defect prevention activities are planned.  
- Common causes of defects are sought out and identified.  
- Common causes of defects are prioritized and systematically eliminated.                                                                                 |
### Technology Change Management

66. **Involves identifying, selecting, and evaluating new technologies, and incorporating effective technologies into the organization (KPA17).**

- Incorporation of technology changes is planned.
- New technologies are evaluated to determine their effect on quality and productivity.
- Appropriate new technologies are transferred into normal practice across the organization.

### Process Change Management

67. **Involves defining process improvement goals and, with senior management sponsorship, proactively and systematically identifying, evaluating, and implementing improvements to the organization’s standard software process and the projects’ defined software processes on a continuous basis (KPA18).**

- Continuous process improvement is planned.
- Participation in the organization’s software process improvement activities is organization wide.
- The organization’s standard software process and the projects’ defined software processes are improved continuously.


**THE END.**
This survey is the final step of the researcher’s effort to evaluate the impact of developers’ attitudes on software maintenance cost for his doctoral dissertation in Information Technology Management. Although your participation in this study is completely voluntary, your response will contribute immensely to the significance of this research. To ensure confidentiality, no name has been requested in the survey and your responses will remain confidential.

The actual time required to complete this survey is estimated to be between 15 and 20 minutes. Please complete the survey by April 28, 2003. Thank you in advance for your assistance.
The survey contains 4 parts. It is estimated to take about 15 to 20 minutes to complete.

Read each statement and then check the box or circle the answer that best describes you or shows how you feel.

PART 1: DEMOGRAPHIC QUESTIONS

Instructions: Circle the answer that best describes you.

1. What is your gender (GEND)? 1=Female, 2=Male --------------- 1 2

2. A Software Developer is a person who builds (designs, codes, tests, debugs, documents, and maintains) computer programs. A Software Project Manager is a person who plans, monitors, and controls people and processes involved in the creation of software products. Based on these definitions, how would you describe yourself (RESP)? 0=Neither 4=Both
   1=Developer 2=Software Project Manager 3=Both

3. Number of years in your current role (PYRS)? 1=0-2 yrs, 2=3-5 yrs, 3=6-8 yrs, 4=9-11 yrs, 5=12+ yrs
   1 2 3 4 5

4. How many years have you been in the software profession (SYRS)? 1=0-2 yrs, 2=3-5 yrs, 3=6-8 yrs, 4=9-11 yrs, 5=12+ yrs
   1 2 3 4 5

5. What type of software work are you currently doing (TWRK)?
   (1=new development; 2=maintenance, 3=both, 4=not applicable).
   1 2 3 4

6. Percentage of my time spent doing software maintenance activities (PTIM): 1=0-20%, 2=21-40%, 3=41-60%, 4=61-80%, 5=81-100%
   1 2 3 4 5

7. How would you describe your highest level of education (EDUC)? (1 = High School; 2 = Some College; 3 = Two-year degree; 4 = Four-year degree; 5 = Masters degree; 6 = PhD)
   1 2 3 4 5 6

Go To Next Page ➔
PART 2: PERSONNEL ATTITUDE QUESTIONS

Instructions: Select one level of agreement for each statement to indicate how you feel.

<table>
<thead>
<tr>
<th></th>
<th>SD</th>
<th>D</th>
<th>N</th>
<th>A</th>
<th>SA</th>
</tr>
</thead>
<tbody>
<tr>
<td>8. Of all activities I perform, software design is one of the activities I enjoy doing most (DAT1).</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9. Of all activities I perform, software-coding is one of the activities I enjoy doing most (DAT2).</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10. Of all activities I perform, software-testing is one of the activities I enjoy doing most (DAT3).</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11. I am familiar with the intent of software process improvement models such as the CMM or CMMI (DAT4).</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12. I am familiar with the content of software process improvement models such as the CMM or CMMI (DAT5).</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>13. Learning about process improvement models such as CMM or CMMI is not a waste of time (DAT6).</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>15. I think documenting my design reduces the amount of time I spend on rework (DAT8).</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>16. Documenting code makes it possible to spend less time doing rework (DAT9).</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>17. I think the time I spend documenting my design and code leads to reducing the time spent on maintenance later on (DAT10).</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>18. I usually make a written plan before I start to solve a problem (DAT11).</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>19. I think using software improvement processes adds value to my project (DAT12).</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>20. I always follow my team's chosen methods for software development instead of choosing my own way (DAT13).</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>21. I would like to learn more about software process improvement activities (DAT14).</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>22. In my opinion, software process improvement activities should not be avoided (DAT15).</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>23. I see software development standards as something I will always use in my implementation of software (DAT16).</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>24. Following established software development standard (such as analysis, design, and coding standards) to develop software does not prevent me from being creative (DAT17).</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>25. Employing software best practices (such as risk management, reviews/inspections, requirements management, metrics-based scheduling, defect-tracking, size-estimating, and so on) improves the overall quality of software (DAT18).</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>26. Software process improvement activities would increase my overall productivity (DAT19).</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>27. Documented software development standards should be encouraged as much as possible (DAT20).</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>28. In my opinion, software maintenance is one of the most important activities in the software life cycle (DAT21).</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>29. Software maintenance cost can be reduced dramatically if software engineering best practices are followed (DAT22).</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Go To Next Page ➔
PART 3: ORGANIZATION QUESTIONS

Instructions: Select one level of agreement for each statement to indicate how you feel about your organization.

<table>
<thead>
<tr>
<th>Statement</th>
<th>SD</th>
<th>D</th>
<th>N</th>
<th>A</th>
<th>SA</th>
</tr>
</thead>
<tbody>
<tr>
<td>30. My organization maintains a set of standard software processes for developing software (ORG1).</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>31. My organization encourages me to follow its standard software process to develop software (ORG2).</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>32. My organization provides adequate funding to make software process improvement happen (ORG3).</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>33. My organization has a group or individual(s) responsible for the organization’s software process activities (ORG4).</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>34. In my opinion, my organization recognizes the need to reduce software maintenance cost (ORG5).</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>35. My organization sees software maintenance cost reduction as a way to gain competitive advantage (ORG6).</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>36. My management values reducing maintenance cost below its current level for new products (ORG7).</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Circle the answer that best describes your organization’s software process improvement capability.

<table>
<thead>
<tr>
<th>Question</th>
<th>Response</th>
</tr>
</thead>
<tbody>
<tr>
<td>37. My organization follows the Software Engineering Institute’s Capability Maturity Model as a process for Software Improvement (CMMP).</td>
<td>Yes</td>
</tr>
<tr>
<td>38. My organization’s overall Process Maturity (PMAT) Level is (1=CMM Level 1; 2=CMM Level 2; 3=CMM Level 3; 4=CMM Level 4; 5=CMM Level 5, NA=Not Applicable).</td>
<td>1</td>
</tr>
<tr>
<td>39. My answer to question #38 is based on (BOAS): (1= Internal Audit, 2 = Mini-Assessment, 3 = Interim Process Assessment, 4 = Software Capability Assessment, 5 = Software Process Assessment, NA = Not Applicable.</td>
<td>1</td>
</tr>
</tbody>
</table>
PART 4: PROJECT AND PRODUCT QUESTIONS

4.1: Your Project Information

Instructions: Provide the required project data and circle one level of agreement for each statement that indicates the amount of your time spent on these.

40. Name or describe a project or sub-system that you’re working on (or have worked on in the past 6 months) that has been released to the customer for use (PROJ).

41. What is (are) the programming language(s) used on this project, e.g., Ada, C, C++, Java, XML, COBOL, etc. (LANG)?

42. What type of application is this project attempting to perform (ATYP)? Select only one.
   [ ] Accounting/Finance  [ ] Command and Control  [ ] Communications
   [ ] DBMS  [ ] Desktop Publishing  [ ] Document Imaging
   [ ] Engineering and Science  [ ] Imaging/Scanning  [ ] Information Management
   [ ] Operating Systems  [ ] Process Control  [ ] Simulation
   [ ] Signal Processing  [ ] Software Development Tools

43. What percentage of your time do you spend making changes that occur due to actual problems discovered after the system was released for use (CORR)?
   1 = 0-20%, 2 = 21-40%, 3 = 41-60%, 4 = 61-80%, 5 = 81-100%

44. What percentage of your time do you spend on modification or upgrading of the software so that it will work properly in a new environment (ADAP)?
   1 = 0-20 %, 2 = 21-40%, 3 = 41-60%, 4 = 61-80%, 5 = 81-100%

45. What percentage of your time do you spend on making changes or enhancements to the software to meet the evolving needs of the system (PERF)?
   1 = 0-20 %, 2 = 21-40%, 3 = 41-60%, 4 = 61-80%, 5 = 81-100%

46. What percentage of your time do you spend on improving the software for future maintainability to provide a better basis for future enhancements (PREV)?
   1 = 0-20%, 2 = 21-40%, 3 = 41-60%, 4 = 61-80%, 5 = 81-100%

47. How many months has your team spent on development and testing of the software you described above since it was first released to the customer (NMOS)?

48. What is the average number of hours per person, per month spent by your team on this project since it was first released to the customer (NHRS)?

49. How many software developers (including the project manager) are or were on your project team (TSIZ)?

50. How big is the project (PSIZ)?
   1 = Very Small Size < 50 KSLOC; 2 = Small Size >50 KSLOC and <= 75 KSLOC; 3 = Medium Size > 75 KSLOC and <= 100 KSLOC; 4 = Large Size > 100 KSLOC and <= 150 KSLOC; 5 = Very Large Size > 150 KSLOC. Circle one.
Impact of developer attitude

Instructions: Select one level of agreement for each statement to indicate how you feel about your project.

<table>
<thead>
<tr>
<th>Statement</th>
<th>SD</th>
<th>D</th>
<th>N</th>
<th>A</th>
<th>SA</th>
</tr>
</thead>
<tbody>
<tr>
<td>51. Project schedule was compressed so much that requirements were not fully analyzed and broken down into components before implementation began (SCHD).</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>52. Software maintainability was clearly identified as an important quality attribute during software development (SMTY).</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>53. In my opinion, there is good communication among the team members (TCOM).</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>54. Tasks assigned to me are always delivered on or before their scheduled due dates (PEFF).</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>55. I find myself spending a lot of time on rework (PCAP).</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>56. I had the appropriate experience/background for this project (PEXP).</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>57. My project team is well experienced in the programming language(s) used to develop the software (LEXP).</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>58. My project team is well experienced in the software tools used to develop the software (TOOL).</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>59. My project team has had prior experience with the type of software system described above (AEXP).</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>60. The product is highly complex (CFLX). (Product Complexity is a continuum, where complexity of a product similar to a prior application is Low and a product that involves all new innovation is High)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
4.2: Your Project Key Process Area (KPA) Evaluation*

Enough information is provided in the following table so that you can assess the degree to which a KPA was exercised on your project. Each KPA is briefly described and its goals are given. Check the box that best describes your project team’s software practices. The response categories are explained below:

- **AA = Almost Always** (over 90% of the time) when the goals are consistently achieved and are well established in standard operating procedures.
- **FR = Frequently** (about 60 to 90% of the time) when the goals are achieved relatively often, but sometimes are omitted under difficult circumstances.
- **AH = About Half** (about 40 to 60% of the time) when the goals are achieved about half of the time.
- **OC = Occasionally** (about 10 to 40% of the time) when the goals are sometimes achieved, but less often.
- **RE = Rarely If Ever** (less than 10% of the time) when the goals are rarely if ever achieved.
- **NA = Does Not Apply** when you have the required knowledge about your project or organization and the KPA, but you feel the KPA does not apply to your circumstances (e.g. Subcontract Management).
- **DK = Don’t Know** when you are uncertain about how to respond for the KPA.

<table>
<thead>
<tr>
<th>Key Process Area</th>
<th>Goals of each KPA</th>
<th>AA</th>
<th>FR</th>
<th>AH</th>
<th>OC</th>
<th>RE</th>
<th>NA</th>
<th>DK</th>
</tr>
</thead>
<tbody>
<tr>
<td>61. Requirements Management:</td>
<td>involves establishing and maintaining an agreement with the customer on the</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>requirements for the software project (KPA1).</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• System requirements allocated to software are controlled to establish a</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>baseline for software engineering and management use.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Software plans, products, and activities are kept consistent with the system</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>requirements allocated to software.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>62. Software Project Planning:</td>
<td>establishes reasonable plans for performing the software engineering activities</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>and for managing the software project (KPA2).</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Software estimates are documented for use in planning and tracking the</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>software project.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Software project activities and commitments are planned and documented.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Affected groups and individuals agree to their commitments related to the</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>software project.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>63. Software Project Tracking and Oversight:</td>
<td>provides adequate visibility into actual progress so that management can take</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>corrective actions when the software project’s performance deviates significantly</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>from the software plans (KPA3).</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Actual results and performances are tracked against the software plans.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Corrective actions are taken and managed to closure when actual results and</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>performance deviate significantly from the software plans.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Changes to software commitments are agreed to by the affected groups and</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>individuals.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Key Process Area

#### Goals of each KPA

<table>
<thead>
<tr>
<th>Key Process Area</th>
<th>Goals of each KPA</th>
</tr>
</thead>
</table>
| **64. Software Subcontract Management:** involves selecting a software subcontractor, establishing commitments with the subcontractor, and tracking and reviewing the subcontractor’s performance and results (KPA4). | - The prime contractor selects qualified software subcontractors.  
- The prime contractor and the software subcontractor agree to their commitments to each other.  
- The prime contractor and the software subcontractor maintain ongoing communications.  
- The prime contractor tracks the software subcontractor’s actual results and performance against its commitments. |
| **65. Software Quality Assurance:** provides management with appropriate visibility into the process being used by the software project and of the products being built (KPA5). | - Software quality assurance activities are planned.  
- Adherence of software products and activities to the applicable standards, procedures, and requirements is verified objectively.  
- Affected groups and individuals are informed of software quality assurance activities and results.  
- Senior management addresses noncompliance issues that cannot be resolved within the software project. |
| **66. Software Configuration Management:** establishes and maintains the integrity of the products of the software project throughout the project’s software life cycle (KPA6). | - Software configuration management activities are planned.  
- Selected software work products are identified, controlled, and available.  
- Changes to identified software work products are controlled.  
- Affected groups and individuals are informed of the status and content of software baselines. |
<table>
<thead>
<tr>
<th>Key Process Area</th>
<th>Goals of each KPA</th>
</tr>
</thead>
<tbody>
<tr>
<td>**67. **Organization Process Focus:</td>
<td>- Software process development and improvement activities are coordinated across</td>
</tr>
<tr>
<td></td>
<td>the organization.</td>
</tr>
<tr>
<td></td>
<td>- The strengths and weaknesses of the software processes used are identified</td>
</tr>
<tr>
<td></td>
<td>relative to a process standard.</td>
</tr>
<tr>
<td></td>
<td>- Organization-level process development and improvement activities are planned.</td>
</tr>
<tr>
<td>**68. **Organization Process Definition:</td>
<td>- A standard software process for the</td>
</tr>
<tr>
<td></td>
<td>organization is developed and maintained.</td>
</tr>
<tr>
<td></td>
<td>- Information related to the use of the organization’s standard software process</td>
</tr>
<tr>
<td></td>
<td>by the software projects is collected, reviewed, and made available.</td>
</tr>
<tr>
<td>**69. **Training Program:</td>
<td>- Training activities are planned.</td>
</tr>
<tr>
<td></td>
<td>- Training for developing the skills and knowledge needed to perform software</td>
</tr>
<tr>
<td></td>
<td>management and technical roles is provided.</td>
</tr>
<tr>
<td></td>
<td>- Individuals in the software engineering group and software-related groups</td>
</tr>
<tr>
<td></td>
<td>receive the training necessary to perform their roles.</td>
</tr>
<tr>
<td>**70. **Integrated Software Management:</td>
<td>- The project’s defined software process is a tailored version of the</td>
</tr>
<tr>
<td></td>
<td>organization’s standard software process.</td>
</tr>
<tr>
<td></td>
<td>- The project is planned and managed according to the project’s defined software</td>
</tr>
<tr>
<td></td>
<td>process.</td>
</tr>
<tr>
<td>**71. **Software Product Engineering:</td>
<td>- The software engineering tasks are defined, integrated, and consistently</td>
</tr>
<tr>
<td></td>
<td>performed to produce the software.</td>
</tr>
<tr>
<td></td>
<td>- Software work products are kept consistent with each other.</td>
</tr>
<tr>
<td>Key Process Area</td>
<td>Goals of each KPA</td>
</tr>
<tr>
<td>----------------------------------------</td>
<td>--------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>72. Intergroup Coordination:</strong></td>
<td>• The customer’s requirements are agreed to by all affected groups.</td>
</tr>
<tr>
<td>establishes a means for the software</td>
<td>• The commitments between the engineering groups are agreed to by the affected groups.</td>
</tr>
<tr>
<td>engineering group to participate actively with the other engineering groups so the project is better able to satisfy the customer’s needs effectively and efficiently (KPA12).</td>
<td>• The engineering groups identify, track, and resolve Intergroup issues.</td>
</tr>
<tr>
<td><strong>73. Peer Review:</strong> removes defects</td>
<td>• Peer review activities are planned.</td>
</tr>
<tr>
<td>from the software work products early and efficiently (KPA13).</td>
<td>• Defects in the software work products are identified and removed.</td>
</tr>
<tr>
<td><strong>74. Quantitative Process Management:</strong> controls the process performance of the software projects quantitatively (KPA14).</td>
<td>• The quantitative process management activities are planned.</td>
</tr>
<tr>
<td><strong>75. Software Quality Management:</strong></td>
<td>• The process performance of the project’s defined software process is controlled quantitatively.</td>
</tr>
<tr>
<td>involves defining quality goals for the software products, establishing plans to achieve these goals, and monitoring and adjusting the software plans, software work products, activities, and quality goals to satisfy the needs and desires of the customer and end user (KPA15).</td>
<td>• The process capability of the organization’s standard software process is known in quantitative terms.</td>
</tr>
<tr>
<td><strong>76. Defect Prevention:</strong> analyzes defects that were encountered in the past and takes specific actions to prevent the occurrence of those types of defects in the future (KPA16).</td>
<td>• Defect prevention activities are planned.</td>
</tr>
<tr>
<td>• Common causes of defects are sought out and identified.</td>
<td>• Common causes of defects are prioritized and systematically eliminated.</td>
</tr>
<tr>
<td>Key Process Area</td>
<td>Goals of each KPA</td>
</tr>
<tr>
<td>-----------------------------------------------</td>
<td>-----------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>77. Technology Change Management:</td>
<td>• Incorporation of technology changes is planned.</td>
</tr>
<tr>
<td>involves identifying, selecting, and evaluating new technologies, and incorporating effective technologies into the organization (KPA17).</td>
<td>• New technologies are evaluated to determine their effect on quality and productivity.</td>
</tr>
<tr>
<td></td>
<td>• Appropriate new technologies are transferred into normal practice across the organization.</td>
</tr>
<tr>
<td>78. Process Change Management:</td>
<td>• Continuous process improvement is planned.</td>
</tr>
<tr>
<td>involves defining process improvement goals and, with senior management sponsorship, proactively and systematically identifying, evaluating, and implementing improvements to the organization’s standard software process and the projects’ defined software processes on a continuous basis (KPA18).</td>
<td>• Participation in the organization’s software process improvement activities is organization wide.</td>
</tr>
<tr>
<td></td>
<td>• The organization’s standard software process and the projects’ defined software processes are improved continuously.</td>
</tr>
</tbody>
</table>


THE END.
APPENDIX C: APPROVAL LETTERS FOR PARTICIPATION IN THE STUDY

To: EK Software Professionals
From: William Atkinson

Dear Software Professional:

Software engineers are often too busy in their jobs to contribute to advances in software engineering research. This is unfortunate because there are valuable contributions that can be made by practicing developers. Here is an opportunity to tap that potential! Boniface Nwugwo, an employee of the company and software professional, is embarking on an interesting study of the impact of software developer attitude on software maintenance cost. The Product Software Council has sanctioned his study and he asks your assistance. Please take a few minutes to complete the survey following the specified link.

I emphasize the confidentiality of your responses. The company will have access to the summary results of the study but not to individual surveys.

Thanking you in advance for your cooperation.
Bill.

Dear Colleague:

I am a Kodak employee and software professional in CI-Output Systems Development’s Washington Development Center. I am also working on my doctoral dissertation in Information Technology Management at the Capella University School of Business. I am researching the impact of software developer attitudes on software maintenance cost. My sincere hope is that my research will identify and quantify the correlation between developer attitudes and software maintenance cost. An understanding of the variables that affect developer attitudes towards software development practices can assist management in providing appropriate development environment in which software developers can succeed in building maintainable software, which will in turn help reduce software maintenance cost.

Although your participation in the study is absolutely voluntary, your response will contribute immensely to the significance of my research. To ensure confidentiality, your business unit has not given me access to the distribution list and the survey will not ask for your name or the organization you belong to. The survey will take a total of 10-15 minutes to complete. Please follow the link below to provide your response by April 21, 2003. When you click on the link, you will be required to enter a password to gain access to the survey. The password is also provided for you below.

http://www.gowithcssg.com/cgi-bin/Smadas1.pl
Password: xxxx

There are no foreseeable risks for participating in this study. Yours and the confidentiality of your organization shall be maintained. A summary of the results of my study will be available on my Web page at: http://www.gowithcssg.com/surveys/boniface/status.htm by end of May 2003.
Dear Colleague:

Please pardon me if you're receiving this notice a second time. A couple of weeks ago, you were invited to participate in a study. If you have already completed the survey, please disregard this notice and read no further. If you're yet to complete the survey, I still need your assistance and your input to the study is very crucial.

As I told you before, I am a Kodak employee and software professional in CI-Output Systems Development’s Washington Development Center. I am also working on my doctoral dissertation in Information Technology Management at the Capella University School of Business. I am researching the impact of software developer attitudes on software maintenance cost. My sincere hope is that my research will identify and quantify the correlation between developer attitudes and software maintenance cost. An understanding of the variables that affect developer attitudes towards software development practices can assist management in providing appropriate development environment in which software developers can succeed in building maintainable software, which will in turn help reduce software maintenance cost.

Although your participation in the study is absolutely voluntary, your response will contribute immensely to the significance of my research. To ensure confidentiality, your business unit has not given me access to the distribution list and the survey will not ask for your name or the organization you belong to. The survey will take a total of 10-15 minutes to complete. Please follow the link below to provide your response by April 21, 2003. You will also be eligible for a free gift if you complete the survey on or before the deadline of April 21, 2003. When you click on the link below, you will be required to enter a password to gain access to the survey. The password is also provided for you here:

http://www.gowithcssg.com/cgi-bin/Smadas1.pl
Password: xxxx

There are no foreseeable risks for participating in this study. Yours and the confidentiality of your organization shall be maintained. A summary of the results of my study will be available on my Web page at: http://www.gowithcssg.com/surveys/boniface/status.htm by end of May 2003.

If you have any questions, please contact me during the day at: (301) 586-1739 or in the evening at: (410) 290-8346.

Thank you very much for your assistance.

Boniface C. Nwugwo