Identifying effort estimation factors for corrective maintenance in object-oriented systems

Michael J. Lee
University of Nevada, Las Vegas

Follow this and additional works at: https://digitalscholarship.unlv.edu/thesesdissertations
Part of the Software Engineering Commons, and the Theory and Algorithms Commons

Repository Citation
https://digitalscholarship.unlv.edu/thesesdissertations/1388

This Thesis is brought to you for free and open access by Digital Scholarship@UNLV. It has been accepted for inclusion in UNLV Theses, Dissertations, Professional Papers, and Capstones by an authorized administrator of Digital Scholarship@UNLV. For more information, please contact digitalscholarship@unlv.edu.
IDENTIFYING EFFORT ESTIMATION FACTORS FOR CORRECTIVE MAINTENANCE IN OBJECT-ORIENTED SYSTEMS

by

Michael J. Lee

Bachelor of Science
University of Utah
1990

A thesis submitted in partial fulfillment of the requirements for the

Master of Science in Management Information Systems

Department of Management Information Systems
College of Business
The Graduate College

University of Nevada, Las Vegas
December 2011
THE GRADUATE COLLEGE

We recommend the thesis prepared under our supervision by

**Michael J. Lee**

entitled

**Identifying Effort Estimation Factors for Corrective Maintenance in Object-Oriented Systems**

be accepted in partial fulfillment of the requirements for the degree of

**Master of Science in Management Information Systems**

*Department of Management Information Systems*

Marcus Rothenberger, Committee Chair

Kenneth Peffers, Committee Member

Andrew Hardin, Committee Member

Kendall Hartley, Graduate College Representative

Ronald Smith, Ph. D., Vice President for Research and Graduate Studies

and Dean of the Graduate College

**December 2011**
ABSTRACT

IDENTIFYING EFFORT ESTIMATION FACTORS FOR CORRECTIVE MAINTENANCE IN OBJECT-ORIENTED SYSTEMS

by

Michael Lee

Dr. Marcus Rothenberger, Examination Committee Chair
Associate Professor and Department Chair, Management Information Systems
University of Nevada, Las Vegas

This research identifies factors that impact software maintenance effort by exploring the decision-making process of expert estimators of corrective maintenance projects by using qualitative methods to identify the factors that they use in deriving estimates. We implement a technique called causal mapping, which allows us to identify the cognitive links between the information that estimators use, and the estimates that they produce based on that information. Results suggest that a total of 17 factors may be relevant for corrective maintenance effort estimation, covering constructs related to developers, code, defects, and environment. When these factors are rank-ordered, they demonstrate that some of the factors that have greater influence on corrective maintenance estimation, as expressed by expert estimators, are very specific to corrective maintenance and not generally observed in popular software estimation or maintenance estimation models. This line of research aims at addressing the limitations of existing maintenance estimation models that do not incorporate a number of soft factors, thus, achieving less accurate estimates than human experts.

Keywords: Software maintenance, effort, estimation, causal mapping
# TABLE OF CONTENTS

ABSTRACT ............................................................................................................................. iii
INTRODUCTION ...................................................................................................................... 1
BACKGROUND AND LITERATURE REVIEW ................................................................. 4
METHODOLOGY ....................................................................................................................... 9
RESULTS AND DISCUSSION .............................................................................................. 17
IMPLICATIONS ..................................................................................................................... 23
LIMITATIONS AND FUTURE RESEARCH ......................................................................... 25
APPENDIX B: FACTORS AND DEFINITIONS ...................................................................... 36
APPENDIX C: IRB EXEMPT REVIEW FORMS ................................................................. 39
REFERENCES ....................................................................................................................... 41
CURRICULUM VITAE ........................................................................................................... 45
INTRODUCTION

Software is expensive, and the majority of the cost of software over its life cycle is related to maintenance (Banker & Slaughter, 2000; Mukhopadhyay, et al., 1992). This cost can be substantial, and the predictability and control of software maintenance effort is critical to an organization's risk management strategy (Boehm & Papaccio, 1988). Maintaining software also takes time and it is difficult to estimate the effort needed. For a maintenance program to be considered successful, maintenance releases must be delivered regularly and predictably (Sneed & Brössler, 2003). Accurate effort estimations are therefore vital to accomplish these maintenance tasks in order to ensure regular delivery. Additionally, not every maintenance intervention is worth making. Some defects are not worth fixing and some adaptations are not cost effective, but one must know the costs associated with those interventions in advance to perform the necessary cost / benefit analysis needed to determine if those interventions are appropriate.

Unfortunately, success in software estimation generally, and in maintenance specifically, has been elusive, being plagued with complex models that lack relevance in practice and consistently high deviations in predicted versus actual values (Menzies, et al., 2006). For businesses to have successful maintenance programs they must be able to better estimate maintenance effort, and therefore research into identifying better estimation models is imperative for business success. The purpose of this research is to gain a deeper understanding of how experts arrive at their estimates for the purpose of ultimately improving the accuracy of those estimates. Notwithstanding the need for corrective maintenance estimation, very little research has been conducted regarding developing effort estimation models specifically for corrective maintenance, with the
DeLucia, et al. (2005) study being the most prominent devoted specifically to corrective maintenance. The DeLucia study evaluated various corrective maintenance estimation models by gathering actual maintenance data and comparing the estimates to the actual performance. Most other studies tend toward a more general approach to maintenance estimation and are not specific to corrective maintenance, including Mukhopadhyay, et al. (1992) and Smith, et al. (2001).

The ability to maintain software depends on many factors. The ease of maintenance interventions can be related to factors such as the complexity of the system (Banker & Slaughter, 2000), the component reuse strategies employed (Rothenberger, et al., 2003), or even the cognitive fit of the developer to the maintenance task (Shaft & Vessey, 2006). This wide array of factors makes it very difficult to estimate the effort involved. Complicating this further is the fact that different types of maintenance interventions exist, each of which has its own distinct tasks and requirements.

Three primary types of maintenance interventions are used to address system deficiencies. Corrective maintenance refers to the modification of a system for the purpose of ensuring that it functions according to intended specifications. Adaptive maintenance consists of modifications made to a system to alter that system to accommodate changing environments such as hardware, operating systems, or other environmental factors that can affect the functionality of the system. Finally, perfective maintenance interventions are intended to meet changing user requirements to ensure that as user needs change, the system will still meet their needs (Bandi, et al., 2003).

Research suggests that each intervention type should have its own estimation models (Fioravanti & Nesi, 2001), because each intervention type requires a significantly
different set of tasks and skills. While adaptive and perfective maintenance both involve
creating new code for an existing application to meet new or altered requirements,
corrective maintenance is very different. In corrective maintenance much of the effort is
shifted from design and coding to debugging and diagnosis. Adaptive and perfective
maintenance tasks could potentially benefit from standard software estimation models, or
at least models extended from standard models, because their lifecycle process of design
and implementation is similar to the lifecycle process of new development (De Lucia, et
al., 2005). Corrective maintenance is much different and more difficult to estimate
because the maintainer may spend substantial time identifying the cause of a defect, only
to make a one-line change to the code. As a result, metrics typically used in software
estimation, such as lines of code (LOC), or models that heavily weigh the costs of code
change, are of limited use for corrective maintenance.

In this study, we use a qualitative methodology, based on the Delphi method (Dalkey
& Helmer, 1963), that concisely captures the decision making process of expert
estimators with the goal of identifying the factors that contribute to their estimates for
corrective maintenance projects. This methodology, called the Collective Causal
Mapping Methodology (CCMM) (Scavarda et al., 2006) provides a technique for creating
an aggregate causal map from a distributed participant set. This enabled us to diversify
our participant set by many dimensions including geography, experience, industry, and
role. The result is a comprehensive understanding of the thought process of experts in the
aggregate.

Data was collected in the form of causal statements, allowing the participant to
indicate how their thought process developed in relation to effort estimation for software
maintenance activities. By aggregating the causal statements of many different participants into a single map, we were able to extract common factors that ultimately led to the participants' estimation of the effort required to complete maintenance tasks.

Some of the factors identified by this process reinforce existing understanding of software estimation generally, as well as maintenance estimation specifically. However, additional factors also emerged that are very specific to maintenance tasks. Some of these factors are not present in existing estimation models, providing a contribution to the understanding of maintenance effort estimation with direct applications to practice.

**BACKGROUND AND LITERATURE REVIEW**

A review of maintenance effort estimation models must, by necessity, begin with an overview of software estimation. Many of the concepts and metrics that provide the structure of maintenance estimation have their foundations in software estimation. An overview of software estimation research can therefore provide context to the more specific discussion of maintenance estimation.

There are numerous software estimation models available in the literature. The oldest and most established are SLIM (Putnam, 1978) and COCOMO (Boehm, 1981). Over the years, these authors have revised their models to accommodate changes in technology and methodology. For example, COCOMO II (Boehm et al., 2000) revised and enhanced Boehm's initial work. The movement to object-oriented development has also required changes to these early models to keep them relevant, and early authors are frequently revisiting their work as technology changes (Boehm & Valerdi, 2008). In an effort to leverage his research and to maintain current models, Putnam has also established a consulting firm, Quantitative Software Management, which develops a set of tools
specifically for software estimation. Most of the research in software estimation is based on this early work and much of that work has interesting augmentations that concentrate on certain aspects of software cost. As an example, In et al. (2006) proposed a quality-based estimation model called the Quality-Based Software Product Line Cost Estimation Model (qCOPLIMO) which is based on two COCOMO suite models, COPLIMO and COQUALMO. This model by In et al. considers software quality costs within the context of the existing COCOMO models, using quality as a factor that affects cost. This type of research indicates that there are techniques that can improve on the existing models.

Despite the wide availability and diversity of estimation models and studies (Jørgensen & Shepperd, 2007), the observed variances between predicted and actual values remain high (Menzies, et al., 2006), providing support for research to attempt to enhance to these models for the purpose of providing more accurate estimations. For example, Smith et al. (2001) augmented Intermediate COCOMO to include task assignment metrics, such as team size, team collaboration or concurrency, and team effort fragmentation across multiple code modules to improve estimates. Still other research attempted to determine the reasons for the estimation errors. Jørgensen & Moløkken-Østvold (2004) discovered that, when questioned about the reasons for the deviation between estimated and actual values for software estimation, respondents are biased based on their role in the organization. Other elements, such as the data collection and analysis methods were also seen to impact estimation deviation in their study. However, not all of the software estimation research is based on Boehm and Putnam. Pendharkar & Rodger (2007) posit that COCOMO and SLIM models rely too much on subjective criteria to be accurate and instead evaluated the measurable factor of team size as a
determinant of development effort. Still other research proposes completely different estimation processes, such as the Minimum Software Cost Model, which is based on economic production theory (Hu, et al., 1998). Ultimately, it is clear that there is far from consensus in the research related to software estimation.

Maintenance estimation is somewhat related to software estimation, although much of the literature focuses on software development and not on maintenance specifically. While some extrapolations can be made from estimation theory to the study of maintenance estimation, there are significant differences between development and maintenance activities. Thus, maintenance warrants its own research and models. Early research in maintenance was directed to differentiating development and maintenance tasks. Kemerer and Slaughter (1999) proposed research on maintenance processes, providing an important distinction between software maintenance and software evolution. They describe maintenance as the modifications necessary to ensure that software met its original intent, while evolution is the modifications necessary to extend the reach of a system into new areas. The research has now matured from this early work to provide an array of different maintenance estimation models and metrics. The variety of maintenance estimation literature speaks to the diversity of factors that one can use to organize and classify maintenance activities. They range from technology-based factors, such as maintenance metrics designed specifically for object-oriented systems (Fioravanti & Nesi, 2001), to models designed to meet the specific needs of different types of maintenance interventions, such as corrective maintenance (De Lucia, et al., 2005; Davis, 1989), and even application-based studies relating to factors such as application structure and complexity (Banker & Slaughter, 2000).
There is also debate as to the nature of the models themselves; whether the best results can be obtained using model-based estimation methods that perform estimations with an algorithm based on historical data and metrics, or expert-based estimation methods that rely on the expertise of humans and their knowledge of the estimated processes (Menzies, et al., 2006). Most of the models used to estimate software development and maintenance effort are algorithmic in nature, drawing on factors suggested by literature and research. Starting with the early work of Putman (1978) and Boehm (1981), there has been much research supporting the superiority of algorithmic estimation; however there is other substantial evidence in the literature suggesting that human-mediated estimation processes can be more accurate than algorithmic models (Vicinanza et al., 1991; Mukhopadhyay et al., 1992; Kitchenham et al., 2002), creating an inconsistency that cannot be ignored.

This evidence that human-mediated processes can possibly improve accuracy suggests that the algorithmic models may not be truly complete. It is possible that these models unintentionally omit factors that could improve estimation. Some research suggests that algorithmic models should include "expert" input to improve accuracy (Smith, et al., 2001).

There is also evidence that cognitive and managerial functions play a significant role in the performance of software maintainers (Jørgensen, 1995), and therefore these factors should be included in maintenance effort estimations. Cognitive factors in maintenance performance are especially critical in corrective maintenance, because the majority of the effort is spent analyzing and debugging the existing code structures. Nguyen et al. (2011) report that more time is spent in task and code comprehension activities for corrective
maintenance than for other maintenance types. While it is apparent that cognitive and behavioral issues in estimation should be researched more thoroughly, the literature is surprisingly silent in this area, leaving an opportunity for further research that explores the thought process of expert estimators and uses that information to construct effort estimation models.

An earlier study that is relevant to this line of research is the development of the Estor model (Mukhopadhyay et al., 1992). This study, used a case-based reasoning approach, simulating an expert's application of prior project knowledge to current estimation problems. Although more accurate than algorithmic models at the time, the authors admitted that one of the limitations was the lack of a deep understanding of the factors that experts use to arrive at their estimates, especially when not constrained by any existing model. The intent of this research is to fill that void, and provide a deeper understanding of an expert's analysis factors, which can improve the performance of maintenance estimation models.

It is possible that these experts may be including many currently underutilized factors in their estimates. We could therefore potentially capture the experts' causal maps that they use to arrive at their estimates, and use that information to determine which factors might truly be of interest when defining a model or promoting an environment that is optimal for maintenance tasks. These may or may not be the same factors that are currently proposed in the literature. Discovering new factors from these expert causal maps could allow us to create an estimation model that more accurately reflects the expert estimation process, with the possibility of generating more accurate estimations overall.
In summary, different types of interventions, such as corrective, adaptive, and perfective interventions, require substantially different tasks, which impacts estimation (Menzies, et al., 2006; Fioravanti & Nesi, 2001; De Lucia, et al., 2005). Corrective maintenance is fundamentally different than either adaptive or perfective maintenance in that the focus is on repairing defects rather than expanding the system's intended purpose. It also differs from other maintenance intervention types in that traditional software estimation models are less applicable because of the extensive amount of time spent on defect identification and debugging activity in corrective maintenance, which are essentially cognitive activities. For this reason, new models for corrective maintenance should be developed that consider these factors. Research also suggests that expert estimations can provide insights and factors that may be missing from current algorithmic estimations; however, little research appears to have been done on building a model that uses expert input for corrective maintenance. This presents an opportunity to fill a void related to corrective maintenance estimation, providing a deeper understanding of the factors that expert estimators consider, which could give us greater insight into how to improve the accuracy of software estimates, while exploring the cognitive and organizational aspects of corrective maintenance in more detail.

**METHODOLOGY**

The methodology followed in this paper is causal mapping, a qualitative approach used to identify the thought process of individuals related to accomplishing a goal or reaching a decision. The foundations for this approach, pioneered by Axelrod (1976), state that to comprehend the decision making process of experts, we must understand the causal links that they use to reach their decisions.
Enhancements of this technique that make it more productive for business and MIS research have led to modifications of Axelrod's original contribution. For example, Nelson et al. (2000) have developed an approach they call Revealed Causal Mapping (RCM) methodology, which they apply specifically to the identification of factors that constitute expertise in the area of software operations support (code maintenance). Their approach uses the concept of a revealed map, implying that the true causal map for any individual is strictly held within the subject's mind. All we can see and understand is the portion of that map that they choose to reveal. RCM uses traditional interview-based techniques to gather this data from participants.

Collective Causal Mapping Methodology (CCMM) (Scavarda et al., 2006), which is the methodology employed in this research, takes a more virtual approach, using web-based interactions with participants as opposed to traditional interviews. Through web-based interviews and interactions with software maintenance experts, we identify and rank order a set of factors that contribute to corrective maintenance effort. CCMM provides a complete set of guidelines defining the study progression, including how to construct the web-based interview instruments, techniques for coding the resulting unstructured data, and organizing this data into a weighted causal map. The web-based interaction paradigm of the CCMM has certain advantages over a traditional interview-based technique. It allows the researcher to work with a larger, more geographically dispersed pool of experts. The experts can remain completely anonymous, and because all communication is handled electronically, there are no interactions directly among the respondents. This eliminates the possibility of groupthink, which can negatively impact the exchange of ideas in direct group interaction.
Participants in this study were individuals recruited by the researchers from their personal contacts in the industry. One of the researchers, an industry practitioner in custom application development, had established a significant network of professionals throughout the US and Canada as a result of an extensive professional training practice. Invitations to participate were sent to professionals in this network who were known to the researchers to have expertise in software maintenance. These participants were drawn from several different geographical areas in the US and Canada, specifically, the Southwest, South and Midwest United States as well as Western Canada. They also represented diverse industries including financial services, insurance, government, non-profit, entertainment, manufacturing and gaming. The participants also represented diverse roles including quality assurance, developers, project managers, development managers and technical executives.

The selection strategy was purposeful in nature as opposed to random. We specifically selected participants that we felt could provide the most substantial contribution to our understanding of the cognitive processes involved with corrective maintenance estimation while covering the domain of knowledge. This selection strategy is not only viable, but necessary in qualitative research (Eisenhardt, 1989; Miles & Huberman, 1994; Seawright & Gerring, 2008). This approach is consistent with the CCMM, which requires non-random participant selection to ensure that the subject domain identified by the researchers is covered by the skills and abilities of the selected participants.

To identify the factors that the participant believes will impact maintenance effort, and therefore his or her estimate of the effort to complete the maintenance task, we set up
a website prompting participants to provide their insights on corrective maintenance estimation factors in a structured format that followed a pattern of "A causes B", where A and B were to be filled in by the respondents (see Appendix A). A participant was able to enter as many causal relationships as he or she found relevant. We conducted an initial pilot study to evaluate the data collection approach. Using the feedback and the results of this pilot, we adjusted the instrument to ensure that the participants would provide relevant data in the correct format. Invitations were sent to 41 potential participants, which generated a total of 27 responses. The respondent age ranged from 27 to 55 with reported maintenance experience from 6 to 31 years (4 to 16 years with regards to object oriented technology). The participants were also asked to self-report their level of proficiency in software maintenance on a seven point Likert scale; self-reported proficiency ranged from 4 to 7 on a scale where 1 represents "not proficient", 4 represents "moderately proficient", and 7 represents "extremely proficient". Thus, all participants met the inclusion criterion of having substantial practice in software maintenance of object-oriented systems.

Two of the researchers independently coded the responses into categories. As this was an exploratory study, and to be consistent with the CCMM, no categories were defined in advance, but rather we defined the categories as suggested by the data (open coding). Over 88 percent of the respondent observations were coded identically between the two researchers. The remaining 12 percent were resolved after one round of discussion, resulting in 100 percent agreement between the two researchers. All of the coding was done incrementally, with the researchers always reviewing cases in the same order. When discussion resulted in a modification to the coding categories or definitions,
we restarted the process and considered each of the cases again, in the same order, to ensure that all cases were compliant with the new categories and definitions. In a subsequent step, a third researcher audited the results by providing confirmation of the codes. This researcher independently assigned participant observations to the defined categories. This process revealed that four observations were stated ambiguously (fitting in either of two existing categories), thus the observations were excluded from the analysis without affecting the results (affected categories were supported by multiple other observations). One inconsistency led us to reword a node definition for clarity. The five remaining inconsistencies, representing only 4 percent of the observations that were entered in the analysis, were resolved in one iteration of clarification with the audit researcher who agreed with the initial coding on those observations. These final audited factors and their definitions are provided in Table B.1 (see Appendix B).

It is interesting to note that although the participants had the option to provide data in complex causal chains, very few participants chose to do this. Most of the responses indicated a direction causal relationship of a factor to maintenance effort without providing intermediary factors. Those that did provide complex chains generally indicated that one factor was causal to another known factor. Due to the direct causal nature of these responses, we were able to eliminate much of the complexity from the model by collapsing it to a set of factors that directly impacted maintenance effort. This provided the most parsimonious interpretation of the data.

One concern frequently associated with qualitative methods is the determination if sufficient data has been collected to ensure that the research has captured the maximum amount of data that is practically possible to collect. Eisenhardt (1989) refers to this point.
as "theoretical saturation." CCMM provides a method for estimating the level of saturation of causal relationships obtained from additional responses, using a non-linear least squares curve fit model that predicts the number of relationships obtained from \( n \) respondents.

\[
R(n) = \alpha(1 - e^{-\beta n})
\]  

In our research, this regression, with an \( \alpha \) estimate of 16.793 and a \( \beta \) estimate of 0.133 demonstrates a good fit to the data with an \( R^2 \) of 94.9 percent. Using this model, the addition of a 28\(^{th} \) participant into the analysis would generate an estimate of a marginal increase of .05 new factors for the next respondent, which represented a marginal percentage increase of .32%. Thus, we concluded that additional respondents would be unlikely to expand the model and that the analysis is saturated. Although we coded the data incrementally as we received it, we did not run a saturation analysis until we had gathered responses from 27 participants. Since the analysis reported an extremely low potential marginal gain from additional participants, we discontinued recruiting new participants at this point. If we had run this analysis earlier, we could have stopped gathering data at an earlier point without impacting our results.

This analysis resulted in the identification of 17 causal relationships that impact effort in corrective maintenance. These relationships represented a concise interpretation of the data both through the first two researchers' initial coding, as well as the third researcher's audit coding; thus no further clustering of the data was likely. The CCMM provides for an optional cluster step that allows for further collapsing of codes, however, because of the concise nature of the results, a further consolidation of the codes was not required. A factor was included in the results as long as there was at least one respondent that cited
the factor as causal to software maintenance effort. Table B.2 (see Appendix B) lists the factors and confirming observations from the participants for each factor.

CCMM provides a process for determining the relative strength of each relationship by using a follow-up interaction with the participants. Understanding the relative strength is critical in interpreting the results so any estimation or management models derived from the results can focus primarily on the higher rated factors. Therefore, once the factors were identified and defined, the next step was to rank-order the factors based on input from our pool of experts. We created a new survey page that presented each of the 17 factors in a different random order, along with their definitions, to the participants.

The web page was designed such that the 17 factors would be randomized for each visit, therefore even if the same participant were to return to the survey again to modify his or her responses, the factors would be presented in a different order than the one previously observed by the participant. The purpose for this randomization was to prevent any possible positional bias from presenting itself in the results. The survey asked the participants to rate each factor on a seven point Likert scale where 1 indicated that the factor had an "Extremely Weak" impact on maintenance effort, 4 indicated "Moderate" impact and 7 indicated an "Extremely Strong" impact (see Appendix A).

Invitations were sent to the original pool of 41 experts, and to an additional 9 experts in an effort to maximize the number of ranking responses received. Adding additional participants is supported and encouraged by CCMM to provide for a larger sample at this stage (Scavarda et al., 2006). These 50 invitations generated a total of 37 responses. These respondents reported maintenance experience from 1 to 30 years (1 to 16 years
with regards to object oriented technology). Their self-reported proficiency ranged from 4 to 7 on the seven point Likert scale previously described.

Defined in the CCMM is a process for scoring the relationships under study, which in this research are the factors previously identified as having an impact on maintenance effort. These scores were weighted based on the experience and reported proficiency of the respondent. The purpose of the weighting was to give higher value to the opinions of respondents with more experience.

The normalized weight of each factor \( w_{jk} \) was calculated by using an expertise factor \( e_i \) of each respondent and the rating feedback provided by each respondent for each factor \( x_{ijk} \) using the following formula:

\[
w_{jk} = \left( \sum_{i \in R_{jk}} \frac{e_i x_{ijk}}{x_{\text{max}}^i} \right) / \left( \sum_{i \in R_{jk}} e_i \right) \tag{ii}
\]

where \( R_{jk} \) is the set of respondents that rated the relationship \( (j,k) \) of the factor to maintenance effort and \( x_{\text{max}} \) is the maximum rating for any factor, which in this study is 7. The resulting weight for each factor is a standardized value between 0 and 1, where 0 corresponds to the value of 1 on the Likert scale, or "Extremely Weak", and 1 corresponds to the value of 7 on the Likert scale, or "Extremely Strong". The results of this process are described and discussed in the next section.

The expertise factor is a function of each respondent’s years of experience and his or her self-reported proficiency level. These values were combined to calculate an expertise factor using the formula

\[
e_i = \left( \frac{y_i}{y_{\text{max}}} \right)^\alpha \left( \frac{s_i}{s_{\text{max}}} \right)^\beta \tag{iii}
\]
where \( y_i \) is the years of experience reported by the respondent, \( y_{\text{max}} \) is the maximum years of experience reported by any respondent (30 in this study), \( s_i \) is the self-reported proficiency of a respondent, and \( s_{\text{max}} \) is the maximum self-reported proficiency of any respondent (7 in this study). The values of \( \alpha \) and \( \beta \) were selected based on the assumption that the number of years of experience of a respondent has diminishing returns as the number increases, while the value of self-reported proficiency increases as the number increases. \( \alpha = 0.5 \) provided diminishing returns for experience and \( \beta = 2 \) provided increasing returns for self-reported proficiency. This is the same calibration used by Scavarda et al. (2006) in their illustration of the CCMM. Based on the intent to show diminishing margins for experience and increasing margins for proficiency, we felt there was no need to alter this calibration, as it was also representative of the relationship that we were trying to measure with regard to software estimation expertise.

**RESULTS AND DISCUSSION**

The initial phase of this research produced a total of 17 factors that are reported to impact corrective maintenance effort. These factors are illustrated in Figure 1 below and their definitions are provided in Table B.1 (see Appendix B). The factors in this table are not presented in any rank order, but rather grouped into categories. To define these categories, two researchers independently arranged the factors into groups based on the general characteristics of each factor. The categories produced by the researchers were consistent with each other and therefore the categories were adopted for classification purposes. Table B.1 presents the definition of each node, as well as its relationship to maintenance effort, as reported by the experts who provided input to this study. Figure 1
is a representation of the same data, illustrating how the results can be grouped into categories.

Figure 1: Illustration of Causal Factors

While the rank-ordered list, as illustrated in Table 1 (below), is interesting with regard to the categories that emerged at both the top and the bottom of the list, it is not surprising that the weighted standardized response does not exhibit a high degree of variability since all of these factors had been previously identified by experts as causal to maintenance effort.
<table>
<thead>
<tr>
<th>Rank</th>
<th>Weighted Standardized Response</th>
<th>Developer Related Factors</th>
<th>Code Related Factors</th>
<th>Defect Related Factors</th>
<th>Environment Related Factors</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.8027</td>
<td>High code complexity</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>0.7812</td>
<td>Low maintainability of code structures</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>0.7539</td>
<td>Low developer experience in maintenance</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>0.7537</td>
<td>Low defect reproducibility</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>0.7080</td>
<td>High level of code / system dependencies</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>0.7069</td>
<td>High level of code volatility</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>0.7020</td>
<td>Low developer familiarity with product</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>0.6878</td>
<td>Low availability of required tools</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>0.6732</td>
<td>Low developer familiarity with technology</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>0.6729</td>
<td>Low clarity or availability of defect documentation</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>0.6721</td>
<td>High level of task switching</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>0.6508</td>
<td>High version / deployment complexity</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>0.6231</td>
<td>Low perception of defect criticality by management</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>0.6072</td>
<td>Low code coverage of unit tests</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>0.5985</td>
<td>High regulatory impact</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>0.5945</td>
<td>Low availability of formal design documentation and code comments</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>0.5838</td>
<td>Low level of team cohesion</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
The factors in the category labeled "Code-Related Factors" are most commonly present in standard software estimation and existing maintenance estimation models. For example, COCOMO II (Boehm et al., 2000) relies heavily on code complexity metrics, including a Lines of Code (LOC) measure for the size of the module. When evaluated in rank order, code-related factors also represent the majority of the higher-ranked factors, with four of the top six ranked factors falling into the code-related category. While "High code complexity" is commonly considered in standard software estimation models, our study revealed additional code-related factors that are specific to maintenance effort. Factors such as "Low maintainability of code structures" and "High level of code volatility" both imply that code for the system already exists and that the factors relate to structure or concurrent usage of that code.

The factors categorized as developer-related are commonly used in both standard software estimation models as well as maintenance models (Boehm et al., 2000; Putnam, 1978). Developer familiarity with the product and the technology has an obvious impact on the time required to complete a corrective maintenance task, since these factors can potentially reduce the duration of cognitive activities such as task comprehension and defect isolation. One interesting result in this study is that the definition of the "Low Developer Experience" factor, as provided by the participants, emphasized experience with defect isolation and debugging in particular, rather than general development experience.

Beyond that, the results include other factors that are specific to corrective maintenance activities and are not generally found in established software estimation models, for example the factor, "Low defect reproducibility." The steps to reproduce a
defect is one of the pieces of data that maintenance developers consider the most important to do their jobs, yet it is an artifact that is often challenging to provide (Zimmerman, et al., 2010). If a defect is consistently reproducible, it is much easier to debug and isolate the offending code. If the defect documentation does not provide these steps to reproduce the defect, then the developer must add time to the schedule to determine these steps. Typically, it is easier to debug code when the defective behavior is being exhibited. If the developer is not able to determine the steps to reproduce the defect, more complex debugging techniques must be employed. The worst possible case is that the developer must read through the code in an attempt to predict what the outcome of each action will be, and this is a very time-consuming process.

Another defect-related factor, "Low code coverage of unit tests," appeared quite low in the rank order at position 14. This is interesting due to the significant amount of practitioner literature advocating the use of unit testing. Most of the literature related to unit testing is within the context of test-driven development. While numerous articles discuss unit testing / test driven development (TDD) and its impact on software quality (Crispin, 2006; Janzen & Saiedian, 2008), very little in the literature discusses unit testing and its impact on software maintenance. The prevailing perspective in the industry is that the purpose for a unit test is solely to validate and regress granular system functionality (Runeson, 2006). Therefore, while unit tests may help developers produce better quality code, maintenance developers would use unit tests primarily for regression testing existing functionality that might be impacted by the corrective maintenance interventions needed to address the target defect. As a result, an organization's unit test program might
reduce the total number of defects in a system, but it would probably have a lesser causal impact on the effort needed to maintain the defects that are discovered in the code.

One of the more surprising results is the absence of software reuse as a factor in software maintenance effort. Numerous studies point to reduced maintenance effort as a benefit of software reuse, including Rothenberger et al. (2003) and Rombach (1991), however none of the experts that we consulted in our study indicated that software reuse had any direct or indirect impact on maintenance effort. Existing studies such as those mentioned above claim that tested and reliable reusable modules can reduce the overall number of defects and therefore reduce the overall maintenance cost. However, even though reuse can reduce the quantity of defects, it is possible reuse could contribute to an increase maintenance effort on a per-defect basis due to the higher level of dependency between application components and systems.

There are a number of possible reasons for the omission of reusability as a factor in this study. First, the experts surveyed may, in fact, see no causal relationship between code reuse and maintenance effort, however in the light of existing research, this may be an extreme interpretation of these results. Another possibility is that the experts that we surveyed do not work in an environment where code reuse is commonly implemented. Although this is possible, the fact that the participants in this study were widely distributed with regard to industry and role makes this a very unlikely interpretation.

One could argue that software reuse programs tend to be constant across all projects in an organization, and as such an expert estimator would not necessarily consider reuse as a distinguishing factor impacting maintenance effort, but rather would have already factored reuse-related issues into a base estimate before considering variations. While
that certainly could be the case in many organizations, Rothenberger (2003) demonstrated
that there are numerous project-level factors that impact the level or type of reuse, which
could then consequently impact the associated maintenance strategy.

It is also possible that experts aggregated the impact of reusability with the factor of
"High Level of Code and System Dependencies." Regardless of the reason, the fact that
reuse did not occur more prominently in the responses of our exert estimators is a
phenomenon that may warrant additional study.

**IMPLICATIONS**

This study both confirms existing research and introduces potential new lines of
research. Three of the top four factors in the rank-order are common among software
estimation models. If we use the maintenance models of COCOMO II as a benchmark,
we see many parallels. COCOMO II relies heavily on complexity metrics, with particular
emphasis on Lines of Code (LOC) size metrics to measure complexity

Even with these validated metrics, variances remain high. Boehm attributes these
variances to the fact that an organization's counting rules used to determine software size
are frequently different than those used to calibrate the models (Jørgensen & Boehm,
2009). Regardless of the reasons for the variances, it is apparent that complexity plays an
important role in estimation. There are a variety of complexity metrics available,
however, and the expert estimator need not focus solely on LOC metrics to determine
estimates. Another popular complexity metric is McCabe's Cyclomatic Complexity (CC)
(McCabe, 1976), which measures the level of cognitive difficulty in understanding a
program and its flow. This metric has been validated by the work of Midha et al. (2010)
and Kemerer & Slaughter (1997), and demonstrates the importance of the impact of
cognitive activity on effort estimation, which can only be captured implicitly in LOC metrics. We believe that confirming the critical nature of code complexity reinforces the need for continued research in complexity measures, including those that consider cognitive complexities in understanding program structure.

Code maintainability also emerges as an importance theme from this study, which has direct implications on how software should be designed and implemented. To develop an effective maintenance model, one must have measures to assess the maintainability of current code. There are a few metrics available for this purpose. Li and Henry (1993), in their work intended to provide support for the applicability of metrics to object-oriented applications, validated previously identified maintainability metrics such as the depth in inheritance tree (DIT), number of direct subclasses/children (NOC), class method cardinality measures / response for class (RFC), lack of cohesion of methods (LCOM), and weighted method complexity using McCabe's cyclomatic complexity metric (WMC). Misra (2005) also drew on existing research to investigate factors that identify maintainability. The methodology was to identify factors that correlate with the Maintainability Index (Welker & Oman, 1995). Among the metrics determined to be highly correlated to the Maintainability Index (MI) were average class size (ACLOC), average method size (AMLOC), average depths of paths (AVPATHS), control density (CDENS), coupling (COF), depth of inheritance tree (DIT), program length (N), percentage of public/protected members (PPPC), and weighted method complexity (WMC). These maintainability measures may provide a starting point for developing an effort estimation model.
The implications of this research for practitioners are twofold. First, with a better understanding of the factors that experts consider causal to maintenance effort, managers can focus on identifying and leveraging metrics on those factors to provide better estimates. This requires the organization to understand and apply the metrics discussed previously. Second, a manager can use his or her understanding of these factors to better manage the development environment to support more efficient maintenance cycles. For example, understanding that expert estimators consider code complexity and maintainability to have a strong causal relationship to maintenance effort, organizational resources can be concentrated in these areas, however organizational transformation to support a culture for maintenance effort optimization is beyond the scope of this study.

LIMITATIONS AND FUTURE RESEARCH

In qualitative research, it is imperative that the researchers collect enough information to ensure that the results are meaningful. Since this research develops its results from the cumulative knowledge obtained from multiple responses, no single response can provide a complete picture of the phenomenon being observed. With regards to this study, this means that no individual participant could possibly provide all maintenance factors to the researchers. It was therefore critical to continue adding data from multiple participants as the study progressed to ensure that we obtained the most comprehensive understanding possible of the factors that we were identifying. According to Eisenhardt (1989), this process should continue until theoretical saturation is reached, or in other words, until we can demonstrate that including additional participants in the study will not provide any additional data.
CCMM prescribes an approach to estimate the contribution of additional responses with the regression model described in the methodology of this paper. Using this approach, we have estimated that this additional contribution is negligible and that we can be confident that we have reached theoretical saturation with our participant pool. Thus, we conclude that the study contains an adequate number of participants, suggesting that with a high likelihood, the identified set of factors is complete for the participant pool selected, however a potential limitation of this study is that since we have used a purposeful sampling technique, there is a possibility that the pool of participants in this study is not diverse enough to ensure that we have captured all of the meaningful factors related to corrective maintenance. We have addressed this limitation by specifically selecting participants from a wide diversity of industry, position, and geographic classifications in an effort to ensure that all relevant factors would be revealed. This is the standard mitigation technique for this issue demanded by qualitative research (Miles & Huberman, 1994). Although these results may not be generalizable to the entire population of software maintenance professionals, we have followed best practices in case selection to ensure that these results are as generalizable as possible and we believe that we have therefore satisfied any concerns on this topic.

Finally, as with all qualitative research, the results of this study are impacted to some extent based on the researchers and their interpretations of the data provided by the participants. To address this possibility, we strictly followed the CCMM defined practice for data coding, which required two researchers to first code the data independently before meeting to reconcile and resolve any disagreements. As an extra precaution, a third researcher reviewed and audited all coding and forced another round of resolution.
This rigorous process removes virtually any possibility that the coding was impacted by the biases of any of the participating researchers.

The logical follow-up to this research would be to apply the identified factors by creating an estimation model that incorporated these findings. Several studies suggest that Multiple Linear Regression (MLR) provides the best vehicle for building estimation models similar to the one that we are proposing (Jørgensen, 1995; Fioravanti & Nesi, 2001). The most significant challenge in working toward a model would be to identify appropriate measures for each of the influence factors. While some factors, such as complexity, have measures already established by other models, many others are soft factors which are difficult to measure and have no established metrics. The ability to identify appropriate metrics is prerequisite to the development of any estimation model.

Other interesting lines of future inquiry are revealed by our results as well. For example, there is significant emphasis in the software industry on unit testing. One of the reasons frequently cited for the necessity of unit tests is to simplify code maintenance. While the presence of unit test coverage in the code base does have a normalized score in the moderate range, it is one of the lower ranked factors, coming in at 14 of 17. While this does not suggest that unit tests are not valuable maintenance tools, it does certainly indicate that expert estimators think that there are many other factors that impact maintenance effort more significantly than unit test code coverage. Additional research related to identifying the comparative value of unit testing for software activities such as requirements management, maintenance, and development tasks would certainly be valuable given that these results diverge from the conventional wisdom.
Phase 1 - Screen Shot 1: Default.aspx (Consent Form)
Phase 1 - Screen Shot 2: Decline.aspx

Phase 1 - Screen Shot 3: Authorize.aspx
Phase 1 - Screen Shot 4: Demographics.aspx
Decisions are usually the result of analyzing many factors. For example, if you were trying to estimate how long it would take for someone to drive a car to a destination, you might consider factors such as the amount of traffic on the roads or how well the driver knows the route. You might envision a causality matrix like this to determine how much time you will estimate it will take for someone drive to the destination:

IF <The driver does not know the route well> THEN <The driver will need directions more often>
IF <The driver needs directions more often> THEN <The drive-time estimate will increase>
IF <The speed limit allows faster travel> THEN <The drive-time estimate will decrease>

Graphically, it might look like this:

![Causality Diagram]

Estimating how much effort it will take for software maintenance tasks works the same way. We are interested in learning more about your thought process with regards to software maintenance estimation. Imagine that you are a manager of a software development team and you have been asked to fix some bugs in an object-oriented software component. If you needed to estimate the amount of effort that it will take for your team to fix these bugs, you could use a causality matrix like the one above. What would be the factors that you would consider? How do these factors impact your estimate?

Using this IF / Then pattern, please provide your thoughts in the boxes below. You can provide as many statements as you like, and they do not all have to be inter-related. There is no minimum or maximum number of "IF / THEN" pairs. Click the button labeled "Add New Row" to add another set of boxes to the page to enter another data pair. Your input can be as general or as specific as you like, and all information can be valuable, even if it seems trivial on the surface. When you are finished, click the "I Am Finished" button below. This will submit your information to the researchers.
Phase 1 - Screen Shot 6: Confirm.aspx

Thank you for participating in this study. Please feel free to come back to this site at any time if you have additional information that you would like to share. When the results of this stage have been compiled, you will be contacted with an invitation to view the results of this stage and to participate in the next stage of the study.

Thank you again for your time and contribution to this research.

Phase 1 - Screen Shot 7: Withdraw.aspx

Thank you for visiting the site and allowing us to inform you about the study. If you provided any data to us, it has been saved. If you would like your data removed from the study, please email the researcher at:

leem21@unlv.nevada.edu

If you change your mind and choose to participate at a later time, you are welcome to visit this site again.
Phase 1 Results

During the first phase of the study, invitations to participate were sent to 41 potential respondents. 27 chose to participate for a response rate of 65.8%. This initial phase of this research produced a total of 17 factors that impact corrective maintenance effort. These factors are illustrated in the graphic below and are grouped into categories that we defined based on the general characteristics of each factor.

On the next page, we will ask you to rate the strength of each of these relationships on a scale from 1 to 7.
Phase 2 - Screen Shot 9: Ratings.aspx (Top Portion) with sample selections

Please rate the following relationships. A rating of 1 indicates that the left node has an extremely weak impact on maintenance effort while 7 indicates that the relationship is extremely strong. For the purpose of this exercise, you can consider “effort” to be the amount of time that is spent completing the maintenance task.

You can hover your mouse over a node or arrow to read a more complete description of that node or arrow. When you are finished, click the submit button at the bottom of the page. The relationships are presented in random order.

Warning: Do not reload this page in your browser before you submit the results. Doing so will corrupt the data due to browser cache behavior.

```
<table>
<thead>
<tr>
<th>Extremely Weak</th>
<th>Very Weak</th>
<th>Weak</th>
<th>Moderate</th>
<th>Strong</th>
<th>Very Strong</th>
<th>Extremely Strong</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
</tr>
</tbody>
</table>
```

```
<table>
<thead>
<tr>
<th>Extremely Weak</th>
<th>Very Weak</th>
<th>Weak</th>
<th>Moderate</th>
<th>Strong</th>
<th>Very Strong</th>
<th>Extremely Strong</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
</tr>
</tbody>
</table>
```

```
<table>
<thead>
<tr>
<th>Extremely Weak</th>
<th>Very Weak</th>
<th>Weak</th>
<th>Moderate</th>
<th>Strong</th>
<th>Very Strong</th>
<th>Extremely Strong</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
</tr>
</tbody>
</table>
```
Phase 2 - Screen Shot 10: Ratings.aspx (Lower Portion) with sample selections

1. Low Detect Reproducibility
   - Extremely Weak
   - Very Weak
   - Weak
   - Moderate
   - Strong
   - Very Strong
   - Extremely Strong
   - 1 2 3 4 5 6 7

2. Low Availability of Required Tools
   - Extremely Weak
   - Very Weak
   - Weak
   - Moderate
   - Strong
   - Very Strong
   - Extremely Strong
   - 1 2 3 4 5 6 7

3. High Version / Deployment Complexity
   - Extremely Weak
   - Very Weak
   - Weak
   - Moderate
   - Strong
   - Very Strong
   - Extremely Strong
   - 1 2 3 4 5 6 7

4. Low Perception of Defect Criticality by Management
   - Extremely Weak
   - Very Weak
   - Weak
   - Moderate
   - Strong
   - Very Strong
   - Extremely Strong
   - 1 2 3 4 5 6 7

Click here to submit your data to the study => Submit
# APPENDIX B: FACTORS AND DEFINITIONS

<table>
<thead>
<tr>
<th>Category</th>
<th>Node Name (ID)</th>
<th>Node Definition</th>
<th>Effect On Maintenance Effort</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Developer Related Factors</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Low developer familiarity with the product (A)</td>
<td>The developer has a low level of familiarity with the code, code structure or business domain of the product.</td>
<td>As developer familiarity with the product decreases, maintenance effort increases.</td>
<td></td>
</tr>
<tr>
<td>Low developer familiarity with the technology (B)</td>
<td>The developer has a low level of familiarity with the programming language, platform, or associated technologies used in the product.</td>
<td>As developer familiarity with the technology decreases, maintenance effort increases.</td>
<td></td>
</tr>
<tr>
<td>Low developer experience in maintenance (C)</td>
<td>The developer is less skilled or experienced in designing, developing or debugging.</td>
<td>As developer experience decreases, maintenance effort increases.</td>
<td></td>
</tr>
<tr>
<td><strong>Code Related Factors</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>High code complexity (D)</td>
<td>The code being maintained is structurally complex, uses complex patterns or technologies, or is large in size.</td>
<td>As code complexity increases, maintenance effort increases.</td>
<td></td>
</tr>
<tr>
<td>Low maintainability of code structure (E)</td>
<td>The affected code has been designed or implemented in a way that limits its maintainability.</td>
<td>As code maintainability decreases, maintenance effort increases.</td>
<td></td>
</tr>
<tr>
<td>High level of code / system dependencies (F)</td>
<td>The code being maintained has substantial dependencies to other systems, components or code.</td>
<td>As the level of code dependency increases, maintenance effort increases.</td>
<td></td>
</tr>
<tr>
<td>High version / deployment complexity (G)</td>
<td>The code being maintained is present in many supported / deployed versions of the product.</td>
<td>As the level of version / deployment complexity increases, maintenance effort increases.</td>
<td></td>
</tr>
<tr>
<td>High level of code volatility (H)</td>
<td>The code being maintained is experiencing a high level of churn / change not related to the defect.</td>
<td>As code volatility increases, maintenance effort increases.</td>
<td></td>
</tr>
<tr>
<td>Low availability of formal design documentation and code comments (I)</td>
<td>There is only limited availability of design documentation including models, diagrams, use cases, etc. is not available or the code is not well-commented.</td>
<td>As the availability of design documentation or code comments decreases, maintenance effort increases.</td>
<td></td>
</tr>
<tr>
<td><strong>Defect Related Factors</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Low clarity or availability of defect documentation (J)</td>
<td>Documentation of the defect behavior is low; availability of logs and/or access to stakeholders for clarification is low.</td>
<td>As the clarity or availability of defect documentation decreases, maintenance effort increases.</td>
<td></td>
</tr>
<tr>
<td>Low defect reproducibility (K)</td>
<td>The defect is not easily reproducible in a maintenance environment.</td>
<td>As the reproducibility of the defect decreases, maintenance effort increases.</td>
<td></td>
</tr>
</tbody>
</table>

---

36
<table>
<thead>
<tr>
<th>Category</th>
<th>Node Name</th>
<th>Node Definition</th>
<th>Effect On Maintenance Effort</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Defect Related Factors</strong></td>
<td>Low code coverage of unit tests (L)</td>
<td>At the beginning of the maintenance project, few unit tests are available to test, validate, or regress behavior.</td>
<td>As the code coverage of unit tests decreases, maintenance effort increases.</td>
</tr>
<tr>
<td></td>
<td>High regulatory impact (M)</td>
<td>The code being maintained covers a feature or functionality that has high legal or regulatory impact on the business.</td>
<td>As the regulatory impact of the maintained code increases, maintenance effort increases.</td>
</tr>
<tr>
<td></td>
<td>Low perception of defect criticality by management (N)</td>
<td>Management views the defect’s correction to be of low criticality or low priority.</td>
<td>As the perceived criticality of the defect by management decreases, maintenance effort increases.</td>
</tr>
<tr>
<td></td>
<td>High level of task switching (O)</td>
<td>The developer or team has responsibilities not related to fixing the defect and must frequently switch between assignments.</td>
<td>As the level of task switching increases, maintenance effort increases.</td>
</tr>
<tr>
<td></td>
<td>Low level of team cohesion (P)</td>
<td>The team does not collaborate or coordinate their efforts well.</td>
<td>As the level of team cohesion decreases, maintenance effort increases.</td>
</tr>
<tr>
<td></td>
<td>Low availability of required tools (Q)</td>
<td>There is little access to tools such as debuggers, libraries, compilers, etc.</td>
<td>As the availability of tools decreases, maintenance effort increases.</td>
</tr>
</tbody>
</table>
### Table B.2
Factors and Participant Responses

| Participant Number | A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q |
|--------------------|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1                  |   |   |   |   |   | X |   |   |   |   |   |   |   |   |   |   |   |   |
| 2                  |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| 3                  | X | X | X |   | X |   |   |   |   |   |   |   |   |   |   |   |   |   |
| 4                  | X | X | X | X | X |   |   |   |   |   |   |   |   |   |   |   |   |   |
| 5                  |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| 6                  |   | X |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| 7                  | X |   | X |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| 8                  | X | X |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| 9                  | X | X |   |   |   |   |   | X |   |   |   |   |   |   |   |   |   |   |
| 10                 | X |   | X | X |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| 11                 | X | X | X |   | X |   |   |   |   |   |   |   |   |   |   |   |   |   |
| 12                 | X |   | X |   |   |   |   |   |   |   |   |   |   |   |   | X | X |   |
| 13                 |   |   | X |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| 14                 |   | X | X |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| 15                 |   |   | X | X |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| 16                 | X | X |   | X |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| 17                 | X | X | X |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| 18                 | X |   | X |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| 19                 |   |   | X |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| 20                 |   | X | X |   | X |   |   |   |   |   |   |   |   |   |   |   |   |   |
| 21                 | X |   | X |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| 22                 | X | X |   | X |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| 23                 |   | X |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| 24                 | X |   | X |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| 25                 | X | X |   | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X |
| 26                 | X | X |   | X | X |   |   | X |   |   |   |   |   |   |   |   |   |   |
| 27                 | X | X | X |   |   |   |   |   | X | X | X | X | X | X |   |   |   |   |

** Factor ID values in column headers are provided in Table B.1 next to the Node Name
Social/Behavioral IRB – Exempt Review
Deemed Exempt

DATE: July 13, 2010
TO: Dr. Marcus Rothenberger, Management Information Systems
FROM: Office of Research Integrity – Human Subjects
RE: Notification of IRB Action by Dr. Ramona Denby Brinson, Chair

Protocol Title: Developing an Effort Estimation Model for Corrective Maintenance in Object Oriented Systems
Protocol # 1007-3507

This memorandum is notification that the project referenced above has been reviewed by the UNLV Social/Behavioral Institutional Review Board (IRB) as indicated in Federal regulatory statutes 45CFR46.

PLEASE NOTE:
Attached to this approval notice is the official Informed Consent/Assent (IC/A) Form for this study. The IC/A contains an official approval stamp. Only copies of this official IC/A form may be used when obtaining consent. Please keep the original for your records.

The protocol has been reviewed and deemed exempt from IRB review. It is not in need of further review or approval by the IRB.

Any changes to the exempt protocol may cause this project to require a different level of IRB review. Should any changes need to be made, please submit a Modification Form.

If you have questions or require any assistance, please contact the Office of Research Integrity - Human Subjects at IRB@unlv.edu or call 895-2794.
Social/Behavioral IRB – Exempt Review
Continuing Review Approved

NOTICE TO ALL RESEARCHERS:
Please be aware that a protocol violation (e.g., failure to submit a modification for any change of an IRB approved protocol may result in mandatory remedial education, additional audits, re-consenting subjects, researcher probation, suspension of any research protocol at issue, suspension of additional existing research protocols, invalidation of all research conducted under the research protocol at issue, and further appropriate consequences as determined by the IRB and the Institutional Officer.

DATE: August 3, 2011
TO: Dr. Marcus Rothenberger, Management Information Systems
FROM: Office of Research Integrity – Human Subjects
RE: Notification of IRB Action by /Charles Rasmussen/ Dr. Charles Rasmussen, Co-Chair Protocol Title Developing an Effort Estimation Model for Corrective Maintenance in Object Oriented Systems Protocol #: 1007-3507

Continuing review of the protocol named above has been reviewed and approved.

PLEASE NOTE:
Upon approval, the research team is responsible for conducting the research as stated in the protocol most recently reviewed and approved by the IRB, which shall include using the most recently submitted Informed Consent/Assent forms and recruitment materials. The official versions of these forms are indicated by footer which contains current approval and expiration dates.

Should there be any change to the protocol, it will be necessary to submit a Modification Form through ORI - Human Subjects. No changes may be made to the existing protocol until modifications have been approved by the IRB. Modified versions of protocol materials must be used upon review and approval. Unanticipated problems, deviations to protocols, and adverse events must be reported to the ORI – HS within 10 days of occurrence.

If you have questions or require any assistance, please contact the Office of Research Integrity – Human Subjects at IRB@unlv.edu or call 895-2794.
REFERENCES


CURRICULUM VITAE

Michael J. Lee

Degrees:

Bachelor of Science, Finance, 1990
University of Utah

Thesis Title:

Identifying Effort Estimation Factors for Corrective Maintenance in Object-Oriented Systems

Books:


Conference Presentations:


Professional Experience:

Strategic Data Insights - Las Vegas, NV: 2011 to present
   Owner / Researcher / Data Specialist

Rocket Gaming – Las Vegas, NV: 2009 to 2011
   Software Development Manager

   Court Technology Officer (Director of IT)

   Owner / Director

Classes Taught:

IS 210 - Introduction to Programming Methodology
   University of Nevada, Las Vegas; Fall 2011