8-6-2011

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

Michael Lee
University of Nevada Las Vegas, mike@strategicdatainsights.com

Marcus A. Rothenberger
University of Nevada Las Vegas, marcus.rothenberger@unlv.edu

Ken Peffers
University of Nevada Las Vegas, ken.peffers@unlv.edu

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

Michael Lee
University of Nevada Las Vegas
mike@strategicdatainsights.com

Marcus Rothenberger
University of Nevada Las Vegas
marcus.rothenberger@unlv.edu

Ken Peffers
University of Nevada Las Vegas
ken.peffers@unlv.edu

ABSTRACT

This research explores 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. 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

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 proper planning and control of software maintenance effort is critical to an organization's overall financial health. 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 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 will focus on identifying factors that impact corrective maintenance effort, and therefore its estimation. 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. Most other studies tend to a more general approach to maintenance estimation including Mukhopadhyay, et al. (1992) and Smith, et al. (2001). It is also important to note that most systems being actively developed today are object-oriented systems. To keep this research relevant, we will focus on object-oriented systems.

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. In's model 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.

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 appliction 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 begs the question of whether or not the algorithmic models are truly complete, or if they unintentionally omit factors that could improve estimation. Some research suggests that algorithmic models should include "expert" input to improve accuracy (Smith, et al., 2001) and that cognitive and managerial functions play a significant role in the performance of software maintainers (Jørgensen, 1995.) 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. 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. A notable exception is the development of the Estor model (Mukhopadhyay et al., 1992). This approach, however, 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 a deep understanding of the factors that experts use to arrive at their estimates, especially when not constrained by any existing model. Thus, we are extending the earlier argument that expert input can improve the performance of maintenance estimation models. It is possible that these experts may be including many of these cognitive factors in their estimations. 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. 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, which are essentially cognitive activities. For this reason, new models for corrective maintenance should be developed. 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, while exploring the cognitive and organizational aspects of corrective maintenance in more detail.

METHODOLOGY

The goal of this research is to identify factors that impact effort in corrective software maintenance, specifically for object-oriented systems. We use qualitative data provided by experts to identify the factors that, in their opinions, impact effort for corrective maintenance tasks in object-oriented systems. Through web-based interviews and interactions with software maintenance experts, based on the Collective Causal Mapping Methodology (CCMM) (Scavarda et al., 2006), we identify a set of factors that contribute to corrective maintenance effort. 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 modification of Axelrod's original contribution. 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. However, unlike RCM which uses an interview-based data collection approach, the web-based data collection of the CCMM (Scavarda, et al. 2006) has certain advantages. 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 among the respondents. That eliminates the possibility of *groupthink*, which can negatively impact the exchange of ideas in direct group interaction.

Participants in this qualitative study were individuals known to the researchers to have expertise in software maintenance. These participants were drawn from several different geographical areas in the US, specifically, the Southwest, the South, and the Midwest. They also came from diverse industries and professional backgrounds. 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. In our study, the subject domain included developers, project managers, development managers and technical executives. 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 respondent selection to ensure that the subject domain identified by the researchers is covered by the respondent set.

To identify the factors that the subject 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. Participants were able to enter as many causal relationships as they found relevant.. We conducted an initial pilot study to evaluate our data collection approach. Using the feedback and the results of this pilot, we adjusted our instrument to ensure that the respondents 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 a reported maintenance experience from 6 to 31 years (4 to 16 with regards to object oriented technology). The respondents 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. Thus, all respondents met our inclusion criterion of having substantial practice in software maintenance of object-oriented systems.

The causal relationships were first coded independently by the two researchers. 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. In a subsequent step, an audit coder provided confirmation of the codes by independently assigning respondent observations to the 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 model (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 coder who agreed with the initial coding on those observations. These final audited factors and their definitions are provided in Table 1, located in the Results section of this paper.

CCMM also prescribes a method for estimating the level of saturation of causal relationships obtained from additional responses, using a non-linear least squares curve fit model: $R(n) = \alpha(1-e^{-\beta n})$ where $R(n)$ is the predicted number of relationships obtained from n respondents. The model demonstrates a good fit to the data with an $R^2 = 94.9$ percent. After the inclusion of all respondents into the analysis, the model estimates a marginal increase of .o6 new factors for the next respondent, which represents a marginal percentage increase of .37%. Thus, we conclude that additional respondents would be unlikely to expand the model and conclude that the analysis is saturated.

Our 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 our coding, as well as the audit coder. 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.

RESULTS

The initial qualitative phase of this research produced a total of 17 factors that impact corrective maintenance effort. These factors and their definitions are provided in Table 1. The factors in this table are not presented in any rank order, but rather grouped into categories that we defined based on the general characteristics of each factor. The table presents both the definition of each node, as well as its relationship to maintenance effort, as reported by the experts who provided input to this study.

The factors show that experts consider many issues beyond those that are code-related. If we compare these factors to COCOMO II (Boehm et al., 2000), we see that while many of the factors are consistent with existing models, corrective maintenance activities require consideration of factors that go beyond those identified by traditional software estimation models. Some of these additional factors, such as the level of task switching and the management perception of the criticality of the defect, are indicative of the nature of the maintenance role in an organization and the fact that an estimator must balance the maintenance task with other development efforts, or perhaps the constant shifting of priorities of the defect backlog. Other factors, such as the level of code volatility or unit test coverage, speak to the special demands of maintenance as an activity that is often concurrent with other development activities in the same code base.

Category	Node Name	Node Definition	Effect On Maintenance Effort
Developer Related Factors	Low developer familiarity with the product	The developer has a low level of familiarity with the code, code structure or business domain of the product.	As developer familiarity with the product decreases, maintenance effort increases.
	Low developer familiarity with the technology	The developer has a low level of familiarity with the programming language, platform, or associated technologies used in the product.	As developer familiarity with the technology decreases, maintenance effort increases.
	Low developer experience	The developer is less skilled or experienced in designing, developing or debugging	As developer experience decreases, maintenance effort increases.
Code Related Factors	High code complexity	The code being maintained is structurally complex, uses complex patterns or technologies, or is large in size.	As code complexity increases, maintenance effort increases.
	Low maintainability of code structure	The code being maintained is not designed or implemented in a maintainable way.	As code maintainability decreases, maintenance effort increases.
	High level of code / system dependencies	The code being maintained has significant dependencies to other systems, components or code.	As the level of code dependency increases, maintenance effort increases.
	High version / deployment complexity	The code being maintained is present in many supported / deployed versions of the product.	As the level of version / deployment complexity increases, maintenance effort increases.
	High level of code volatility	The code being maintained is experiencing a high level of churn / change not related to the defect.	As code volatility increases, maintenance effort increases.
	Low availability of formal design documentation and code comments	Design documentation including models, diagrams, use cases, etc. is not available or the code is not well-commented.	As the availability of design documentation or code comments decreases, maintenance effort increases.
Defect Related Factors	Low clarity or availability of defect documentation	The details of the behavior of the defect are not clearly documented. Logs are not available and/or stakeholders are not accessible to clarify behavior.	As the clarity or availability of defect documentation decreases, maintenance effort increases.
	Low defect reproducibility	The defect is not easily reproducible in a maintenance environment.	As the reproducibility of the defect decreases, maintenance effort increases.
	Low code coverage of unit tests	Unit Tests are not generally available to test, validate, or regress behavior.	As the code coverage of unit tests decreases, maintenance effort increases.
Environment Related Factors	High regulatory impact	The code being maintained covers a feature or functionality that has high legal or regulatory impact on the business.	As the regulatory impact of the maintained code increases, maintenance effort increases.
	Low perception of defect criticality by management	Management does not view the defect as critical or high priority.	As the perceived criticality of the defect by management decreases, maintenance effort increases.
	High level of task switching	The developer or team has responsibilities not related to fixing the defect and must frequently switch between assignments.	As the level of task switching increases, maintenance effort increases.
	Low level of team cohesion	The team does not collaborate or coordinate their efforts well.	As the level of team cohesion decreases, maintenance effort increases.
	Low availability of required tools	There is insufficient access to tools such as debuggers, libraries, compilers, etc.	As the availability of tools decreases, maintenance effort increases.

Table 1: Effort Estimation Factors

CONCLUSION AND FUTURE RESEARCH

This research provides new insights into estimation specific to corrective maintenance. By peering into the decision-making criteria of expert estimators, we have identified a collection of factors that influence estimation. These factors provide an important step on the path to understanding this estimation process. Knowing and having documented these factors may help corrective maintenance professionals to improve their estimates. Nevertheless, such estimates still would be based on a holistic approach; this is where further research can make additional contributions. Subsequent research can use this information to simplify the estimation process by developing a formal estimation model that incorporates the factors identified in this study. The identification of new factors suggests that new models should be created for corrective maintenance estimation. Several studies suggest that Multiple Linear Regression (MLR) provides the best vehicle for building estimation models similar to the one that we would propose (Jørgensen, 1995; Fioravanti & Nesi, 2001.) There are two primary challenges in this process. First, relevant measures must be developed for each of these factors, and second, actual maintenance data must be gathered to build and calibrate a model. Our current work is aimed in this direction. In addition to what was described in this paper, the CCMM suggests eliciting feedback from the experts regarding the strength of the proposed causal relationships in a subsequent step. We will solicit this information and use it to create a final cluster structure that provides the most parsimonious implementation of the causal map possible with a rank order of the factors. This will allow us to enter the factors with the strongest relationships into a model that will be calibrated using additional data on completed maintenance projects.

REFERENCES

1. Axelrod, R. (1976) Structure of decision: The cognitive maps of political elites, Princeton University Press, Princeton, NJ.
2. Bandi, R. K., Vaishnavi, V. K. and Turk, D. E. (2003) Predicting maintenance performance using object-oriented design complexity metrics, *IEEE Transactions on Software Engineering 29, 1* , 77-87.
3. Banker, R. D. and Slaughter, S. A. (2000) The moderating effects of structure on volatility and complexity in software enhancement, *Information Systems Research* 11, 3 , 219-240.
4. Boehm, B. W. (1981) Software engineering economics, Prentice-Hall, Englewood Cliffs, NJ.
5. Boehm, B. W., Abts, C., Brown, A. W., Chulani, S., Clark, B. K., Horowitz, E., Madachy, R, D., Reifer, D. J. and Steece, B. (2000) Software cost estimation with COCOMO II, Prentice-Hall, Englewood Cliffs, NJ.
6. Boehm, B. W. and Valerdi, R. (2008) Achievements and challenges in COCOMO-based software resource estimation, *IEEE Software,* Sep / Oct , 74-83.
7. Davis, J. S. (1989) Investigation of predictors of failures and debugging effort for large MIS, *Information and Software Technology,* 31, 4, 170-174.
8. De Lucia, A., Pompella, E. and Stefanucci, S. (2005) Assessing effort estimation models for corrective maintenance through empirical studies, *Information and Software Technology,* 47 , 3-15.
9. Eisenhardt, K. (1989) Building theories from case study research, *Academy of Management Review,* 14, 4, 532-550.
10. Fioravanti, F. and Nesi, P. (2001) Estimation and prediction metrics for adaptive maintenance effort of object-oriented systems, *IEEE Transactions on Software Engineering,* 27, 12, 1062-1084.
11. Hu, Q., Plant, R. T. and Hertz, D. B. (1998) Software cost estimation using economic production models, *Journal of Management Information Systems,* 15, 1, 143-163.
12. In, H. P., Baik, J., Kim, S., Yang, Y. and Boehm, B. (2006) A quality-based cost estimation model for the product line life cycle, *Communications of the ACM,* 49, 12, 85-88.
13. Jørgensen, M. (1995) Experience with the accuracy of software maintenance task effort prediction models, *IEEE Transactions on Software Engineering,* 21, 8, 674-681.
14. Jørgensen, M. and Moløkken-Østvold, K. (2004) Reasons for software effort estimation error: Impact of respondent role, information collection approach, and data analysis method, *IEEE Transactions on Software Engineering,* 32, 12, 993-1007.

15. Kemerer, C. F. and Slaughter, S. (1999) An empirical approach to studying software evolution, *IEEE Transactions on Software Engineering,* 25, 4, 493-509.

16. Kitchenham, B., Pfleeger, H. L., McColl, B. and Eagan, S. (2002) An empirical study of maintenance and development estimation accuracy, *Journal of Systems and Software,* 64, 57-77.

17. Menzies, T., Chen, Z., Hihn, J. and Lum, K. (2006) Selecting best practices for effort estimation, *IEEE Transactions on Software Engineering,* 32, 11, 883-895.

18. Miles, M. B. and Huberman, M. (1994) Qualitative Data Analysis, Sage, London.

19. Mukhopadhyay, T., Vicinanza, S. S. and Prietula, M. J. (1992) Examining the feasibility of a case-based reasoning model for software effort estimation, *MIS Quarterly,* 16, 2, 155-171.

20. Nelson, K. M., Nadkarmi, S., Narayanan, V. K. and Ghods, M. (2000) Understanding software operations support expertise: A revealed causal mapping approach, *MIS Quartlerly,* 24, 4, 475-507.

21. Pendharkar, P. C. and Rodger, J. A. (2007) An empirical study of the impact of team size on software development effort, *Information Technology Management,* 8, 253-262.

22. Putnam, L. H. (1978) A general empirical solution to the macro software sizing and estimating problem, *IEEE Transactions on Software Engineering,* 4, 4, 345-361.

23. Rothenberger, M. A., Dooley, K. J., Kulkarni, U. R. and Nada, N. (2003) Strategies for software reuse: A principal component analysis of reuse practices, *IEEE Transactions on Software Engineering,* 29, 9, 825-837.

24. Scavarda, A. J., Bouzdine-Chameeva, T., Goldstein, S. M., Hays, J. M. and Hill, A. V. (2006) A methodolology for constructing collective causal maps, *Decision Sciences* 37, 2, 263-283.

25. Seawright, J. and Gerring, J. (2008) Case selection techniques in case study research, *Political Research Quartley,* 61, 2 , 294-308.

26. Shaft, T. M. and Vessey, I. (2006) The role of cognitive fit in the relationship between software comprehension and modification, *MIS Quarterly,* 30, 1, 29-55.

27. Smith, R. K., Hale, J. E. and Parrish, A. S. (2001) An empirical study using task assignment patterns to improve the accuracy of software effort estimation, *IEEE Transactions on Software Engineering,* 27, 3, 264-271.

28. Sneed, H. M. and Brössler, P. (2003) Critical success factors in software maintenance, in *Proceedings of the International Conference on Software Maintenance*, September 22-26, Vienna, Austria: IEEE Computer Society, 190-198.

29. Vicinanza, S. S., Mukhopadhyay, T. and Prietula, M. J. (1991) Software-effort estimation: An exploratory study of expert performance, *Information Systems Research,* 2, 4, 243-262.