

DISS 725 – System Development: Research Paper 3
Software Maintenance

by

Ronald G. Wolak
wolakron@nova.edu

A paper submitted in fulfillment of the requirements
for DISS 725 Spring 2001 – System Development: Research Paper 3

School of Computer and Information Sciences
Nova Southeastern University

June 2001

An Abstract of a Paper Submitted to Nova Southeastern University in Fulfillment of the Requirements for DISS 725 Spring 2001 – System Development: Research Paper 3

DISS 725 – System Development: Research Paper 3
Software Maintenance

by
Ronald G. Wolak

June 2001

The paper that follows was submitted to satisfy the requirements of DISS 725 Spring 2001 – System Development: Research Paper 3. Software maintenance is recognized as an important part of the software development life cycle. Maintenance systems are integral in achieving control of the total software process and when properly applied lead to higher quality. Software maintenance activities currently account for more than half of the typical software budget. In addition, more than 50 percent of global software developers are engaged in modifying existing applications. In the following pages, the paper began with an introduction to the software maintenance process. In this discussion, the four major types of software maintenance were explored: corrective, adaptive, perfective, and preventive. In addition, the use of software maintenance tools was explored. This was followed by a discussion of the problems encountered during the software maintenance process and a look into positive developments in the maintenance industry. The paper concluded with a summary and recommendations.

Chapter 1 Introduction

Software maintenance is recognized as an important part of the software development life cycle (SDLC). Maintenance systems are integral in achieving control of the total software process and when properly applied lead to higher quality. Software maintenance activities currently account for more than half of the typical software budget (Glass, 1989). In addition, more than 50 percent of global software developers are engaged in modifying existing applications (Jones, 2000). The following introductory sections describe the problem to be investigated and the goal to be achieved. In addition, the introduction provides an analysis of the relevance of the research and discusses the paper's five-chapter format.

Problem Statement and Goal

Maintaining software is expensive (Smith, 1999). The maintenance phase of the software development life cycle is often the longest and the most expensive. Software applications that take months or years to develop are sometimes in service for decades, and maintenance often accounts for 65 to 75 percent of total SDLC costs. In response to this problem, the software industry is exhibiting an increased interest in the benefits yielded by improved software maintenance processes. The professional community now recognizes the importance of timely and accurate software maintenance.

The goal of this paper is to provide an overview of the four major types of software maintenance, to discuss the common problems encountered during the software maintenance process, and to investigate positive developments in the field.

Relevance

This research paper is relevant to the topic of software maintenance and support. The paper begins with an introduction to the software maintenance process. In this discussion the four major types of software maintenance are explored: corrective, adaptive, perfective, and preventive. This is followed by a discussion of the problems encountered during software maintenance.

Format

This research paper is a descriptive study formatted in five chapters. The first chapter covers the paper's problem statement and goal, relevance, and format. This is followed in the second chapter by a review of the literature relevant to the problem. In the third chapter, the research methods and online tools and resources employed during the completion of the paper are described. The fourth chapter presents the results of the research and provides an analysis of current software maintenance problems and positive industry developments. The fifth chapter concludes the paper with a summary and recommendations.

Summary

Software maintenance is an integral part of the software development life cycle. Many problems are encountered during software maintenance (Dart, Christie, & Brown, 1993). These include the need for more effective software maintenance tools, a lack of software documentation, the low status of maintainers, and the lack of a design-for-maintenance viewpoint in the software development phase. In the following pages, this paper provides a review of literature relevant to software maintenance, a description of research methods employed, results of the research, recommendations, and an overall summary.

Chapter 2 Review of Literature

The literature review that follows is organized by subject heading. Those subjects include the four major types of software maintenance: corrective, adaptive, perfective, and preventive. Also included is a section on maintenance tools.

The software maintenance phase of the software development life cycle begins after delivery of the software product (Pigoski, 1996). Maintenance covers the life a software system from the time of installation until the software is no longer used. In addition, the ISO 11207 standard places emphasis on predelivery aspects of software maintenance (Pigoski, 2001). These aspects include planning for postdelivery operations, supportability, and logistics determination. Postdelivery activities include software training, modification, and help desk operations.

Maintenance Types

E.B. Swanson was one of the first to categorize software maintenance (Pigoski, 1996). He defined three different categories of software maintenance: corrective, adaptive, and perfective. The 1993 IEEE Standard on Software Maintenance further defined the categories and added a fourth preventive maintenance category.

Corrective

Corrective maintenance involves changing a software application to remove errors (Chapin, 2000b). The three main causes of corrective maintenance are design errors, logic errors, and coding errors. Examples include the correction of problems such as misspelled words in a user interface or incorrect algorithms that damage, corrupt, or destroy data. However, new faults are often introduced during the maintenance process (Kammer, 2000). For example, the more difficult it is to maintain a software application, the more likely it is that software updates will not be correctly installed by users. This often leads to system faults that further impact operation, functionality, and security. Corrective maintenance accounts for approximately 20 percent of all software maintenance.

A study conducted by Mockus and Votta (2000) used keyword classification rules to identify the corrective maintenance activities of large-scale software systems. Keywords included fix, bug, error, fixup, and fail. Results showed that when compared to the other types, corrective changes tended to be the most difficult. In addition, the interval for corrective changes was the smallest. The study also showed a strong relationship between the type and size of a change and the time required to carry it out.

A related article by Mayrhauser and Vans (1997) reported on a field study conducted to understand the corrective maintenance of large-scale software by professional software maintenance engineers. The study found that corrective maintenance was not a popular activity. Maintenance engineers were required to have knowledge about the specific application in addition to language and domain skills. In many cases, the task was assigned to novices with some language skills or to an expert programmer that lacked experience in the new language. In both cases, corrective maintenance activities involved “learning on the job.”

Mattsson (1998) proposed a conceptual model of the basic concepts utilized within corrective software maintenance. The model was a synthesis of six different models: a framework suggested by the Software Engineering Institute, the IEEE standards definitions, and the industrial maintenance systems utilized by Ericsson, Ellemtel, AAB Automation Products, and ABB Robotics. The model clarified the most fundamental concepts required for the management of software problems and defects. In addition, it provided a common forum for communication during corrective software maintenance activities.

Adaptive

Adaptive software maintenance is any effort that is the result of changes in a software application's operating environment (Burrows, 1984). These environmental modifications consist mainly of changes to the following:

- Rules, laws, and regulations that affect the application
- Hardware configurations (e.g. new printers)
- File structures and data formats
- System software (e.g. operating system or utilities)

Adaptive maintenance accounts for approximately 20 percent of software maintenance activities. The end user does not see a change in the operation of the software, but the software maintainer must expend resources to make the necessary changes.

Fioravanti, Nesi, and Stortoni (1999) presented a model for the estimation and prediction of the adaptive maintenance required for object oriented software systems. Object oriented modeling has been adopted by the industry recently and older systems are in need of adaptive maintenance to better meet current user requirements. The paper validated that certain effort estimation/prediction metrics are also useful for the estimation/prediction of software maintenance effort. In addition, the paper concluded that during adaptive maintenance considerable attention should be paid to methods definition and the methods interface. The complexity of these interfaces for locally defined methods is the most important factor when estimating adaptive software maintenance effort.

In a related paper, Rayside, Kerr, and Kontogiannis (1998) discussed techniques to automatically detect and adapt to changes in a Java application's library. The rapid evolution of Java libraries, together with Java's run-time linking, may produce incompatibilities between an application and the library it relies on. In the paper, the authors developed and tested a prototype tool that was useful in detecting adaptive maintenance in Java applications developed on new versions of the Java Development Kit (JDK) but deployed on older versions.

Self-adaptive software takes this concept one step further by modifying its behavior in response to changes in its operating environment (Oreizy, Gorlick, Taylor, Heimbigner, Johnson, Medvidovic, Quilici, Rosenblum, & Wolf, 1999). Operating environment includes anything observable by the software system (e.g. end-user input, external hardware devices and sensors, and program instrumentation). Self-adaptive software systems would be particularly effective when applied to military systems in which battlefield conditions (i.e. operating environment) are subject to change without warning. One civilian application might be the use of unmanned air vehicles (UAV)

deployed for land-use management, freeway-traffic management, and airborne cellular-telephone relay stations.

Perfective

Perfective maintenance is software maintenance implemented to improve the maintainability, performance, or other attributes of a computer application (Burrows, 1984). Furthermore, perfective maintenance includes all changes, insertions, deletions, modifications, extensions, and enhancements made to a system to meet evolving and/or expanding user needs (Pigoski, 1996). For example, the addition of features that were not in the original specification that a user wishes added is perfective maintenance. Perfective maintenance comprises approximately 60 percent of all software maintenance.

In a related paper, Domsch and Schach (1999) reported on a case study of the maintenance of an object-oriented application in which a text-based user interface was replaced with a graphical user interface (GUI). The study found that 94.8 percent of the maintenance effort was perfective (i.e. GUI development). The paper concluded that adding a GUI to an existing software application was difficult and time-consuming unless the maintainer had extensive GUI design experience. The quality of the existing software product and its current user interface had little impact on this conclusion.

Preventive

In 1993, the IEEE added a fourth category of software maintenance – preventive maintenance (Pigoski, 1996). Some software maintainers classify preventive maintenance under the corrective category. Preventive maintenance is defined as maintenance performed for the purpose of preventing problems before they happen. This category is particularly important in safety-critical systems such as in aircraft and the space shuttle. Another definition offered by Vehvilainen (2000) is that preventive maintenance refers to all software maintenance activities that are prepared and decided upon regularly. These activities are based upon the analyses of present conditions and the forecasted needs of the software.

In a related article, Chapin (2000) provided a review of the history of software maintenance and the role of preventive maintenance. The link between scheduled and preventive maintenance was also discussed. Preventive maintenance was interpreted to mean improving a software application's future maintainability. Forecasting what would improve maintainability was shown to be difficult to accomplish. The author also pointed out that most successful forecasting was done when software maintenance was performed on a scheduled basis (i.e. preventive maintenance should be incorporated into scheduled maintenance).

The effectiveness of preventive maintenance in enhancing the software dependability of operational software was explored in another article (Garg, Puliafito, Telek, & Trivedi, 1998). The paper presented a model for a transaction based software application that employed preventive maintenance to increase availability, decrease response time, and minimize the probability of loss. Numerical examples were presented to illustrate the applicability of the model. The main strength of the model was its ability to capture the dependence of crash/hang failures and performance degradation on time and instantaneous load.

Maintenance Tools

Software maintenance tools can be defined as anything functional that can assist the software maintainer to address maintenance problems (Lethbridge & Singer, 1997). Software maintenance tools are designed to satisfy requirements in the following five areas:

- Analysis and design
- Testing
- Software configuration management
- Reverse engineering
- Documentation management

Computer-Aided Software Engineering (CASE) tools currently exist to aid software engineers perform a variety of tasks that include requirements analysis, software design, code production, testing, document generation, and project management. While CASE tools have focused on other areas of SDLC, the use of CASE tools for software maintenance has the potential to significantly improve productivity and reduce cost.

A study by CASE Associates showed that between 50 to 70 percent of a developer's time was spent making changes to software (Sharon, 1996). Therefore, the tool with the greatest potential impact on software development is not the development tool. It is the software maintenance tool.

A related paper by Dumke and Winkler (1997) discussed the use of Computer Assisted Software Measurement and Evaluation (CAME) tools. CAME tools are designed for code analysis and measurement. They are applied during the implementation and maintenance development phases. The CAME tool classification includes tools for model-based software component analysis, metrics application, measurement result presentation, and statistical analysis and evaluation.

Glass (1989) identified software maintenance documentation as a key factor in the software maintenance process. Understanding a software application consumes more time than any other maintenance task. Reducing this time with up-to-date documentation is the goal of software maintenance tools. In a related paper, Cioch and Palazzolo (1996) discussed a documentation approach and tool developed to accelerate the efficiency of software maintainers. The approach identified the four learning stages of a software maintainer: newcomer, student, intern, and expert. The information and form of the documentation presented by the documentation tool differed according to the ability of the maintainer. The approach was successfully used by the U.S. Army TARDEC to maintain the ground vehicle simulation software used by the Vetronics Simulation Facility.

Examples of maintenance tools include VIFOR 2, xVue, and RETIRE. VIFOR 2 focuses on the deterioration in structure and documentation that typically occurs during the maintenance of legacy systems (Rajlich & Adnapally, 1996). The tool employs browsing and hypertext documentation technologies to provide rapid code navigation along with the incremental recording and retrieval of documentation.

xVue, a software maintenance tool by Telcordia Technologies, allows maintainers to quickly locate code associated with a particular system feature by highlighting the source code related to that feature (Telcordia, 2001). xVue also facilitates the mapping of system features to program components. Another maintenance tool, RETIRE, was

developed by the Source Recovery Company (SRC, 1999). RETIRE automatically converts outdated 4GL (fourth-generation language) code to COBOL. The tool replaces the manual rewrite process for 25 to 5 percent of the cost and completes tasks in 10 to 20 percent of the time.

Summary

The literature review presented above was organized by subject heading. The subjects included the four major types of software maintenance: corrective, adaptive, perfective, and preventive. Also included was a section on software maintenance tools.

Chapter 3 Methodology

Research Type

This paper was a research-based descriptive study. The key outcome of the investigation was the exploration of the four major types of software maintenance along with an in-depth look at problems encountered during the maintenance process.

Research Methods Employed

The primary research method employed throughout the course of writing this paper was browser-based Internet searches. The literature reviewed included textbooks, journal articles, and magazine articles referenced by a select set of online resources. Relevant texts were located, ordered, and delivered using the Fatbrain.com Internet site. The full text articles from journals and magazines were located and subsequently downloaded.

Online Tools and Resources

A variety of online resources were used to locate and download literature relevant to the goal of the paper. These resources included ACM Search (www.acm.org/dl/search.html), IEEE Digital Library (<http://computer.org/search.htm>), and ProQuest Direct (<http://proquest.umi.com/>). Perhaps the most powerful search tools to be employed were the intelligent search agents Copernic 2001 and LexiBot.

Copernic 2001 is a well-documented freeware search agent (Copernic, 2001). It uses predefined channel sets, which allows researchers to target inquiries to all major Web search engines and also search for relevant text in newsgroups. Copernic conducts fast, multithreaded, full Boolean searches with progress displays and customizable search depth. Once results are compiled, Copernic displays returns (including name, location, and introductory text) in a right-click-enhanced list box sorted by relevance.

Another search technology utilized to gather literature was LexiBot from BrightPlanet (LexiBot, 2001). The LexiBot desktop search client acts as a universal translator for all dialects of search engines and searchable databases. LexiBot is able to search 150 services at one time using a standard query format. In addition, LexiBot's search technology is capable of identifying, retrieving, qualifying, and organizing "deep" and "surface" content from the Internet.

Summary

In summary, this paper was a research-based descriptive study. Browser-based Internet searches were the primary research method employed. These searches queried databases that included the ACM, IEEE, and ProQuest Direct. Specialized client-based search technologies (i.e. Copernic 2001 and LexiBot) also aided in locating relevant literature.

Chapter 4 Results

Despite years of progress in managing the development life cycle of software applications, software maintenance organizations continue to experience difficulty in performing the four types of maintenance discussed in previous chapters (Sawyer, Sommerville, & Viller, 1999). The following sections begin with an investigation of the problems encountered. This is followed by a discussion of positive developments in the field of software maintenance.

Software Maintenance Problems

A study by the Software Engineering Institute (SEI) revealed problems believed to be typical of many software maintenance organizations regardless of the type of maintenance performed (Dart, Christie, & Brown, 1993). Software maintenance practices are frequently associated with high cost and low productivity. Problems contributing to these factors include the need for more effective software maintenance tools, a lack of software documentation, the low status of maintainers, and the lack of a design-for-maintenance viewpoint in software development processes.

Effective Software Maintenance Tools

More effective software tools are needed to facilitate maintenance activities such as coding and testing in addition to related management activities (e.g. budgeting and resource allocation) (Dart, Christie, & Brown, 1993). Maintenance tool-related issues typically center on the following:

- Availability, quality, and integration of CASE tools
- Adequate documentation tools
- Configuration management support tools
- Lack of reverse engineering tools
- Different tooling for lifecycle maintenance versus new software development
- Better testing tools and procedures

Effective tools improve maintenance productivity by helping the maintainer understand the current system and make changes and repairs more efficiently (Sharon, 1996).

Documentation

Documentation is another area in which problems exist (Glass, 1989). There are typically two extremes in software documentation. The first is a total lack of documentation because of schedule and budget factors during product development. The second is too much documentation. As a software application evolves this bulky documentation becomes out of date and essentially useless. The result in both cases is a lack of quality documentation for the maintainer to use.

Maintainer Status

Maintaining software applications is not highly regarded in the industry (Smith, 1999). Maintainers often have little esteem and are viewed to be at bottom of the

programming hierarchy. In the past, this viewpoint was reinforced by the practice of relegating maintenance tasks to the less capable while talented developers were given the task of developing new systems (Sharon, 1996).

Design-for-Maintenance

Software applications designed without considering maintainability is a major maintenance issue (Dart, Christie, & Brown, 1993). Tight schedules often result in software applications that are moved into the maintenance phase before all deliverables have been finished. Consequently, significant maintenance time is devoted to problems related to the support of poorly designed, coded, tested, and documented software.

Designing maintainable software is more than writing understandable code. It involves looking at programs from the viewpoint of a person required to change or repair the code (Smith, 1999). For example, programming languages with large and varied instruction sets allow developers to code an algorithm in a variety of ways. However, they are also more difficult for a maintainer to understand. The best choice is sometimes a language that allows a process to be written in a limited number of ways.

Positive Developments

In response to the problems discussed above, the software industry has made changes in several areas of software maintenance. These include developments in the areas of cost, standards, tools, process improvement, and outsourcing (Schneidewind, Chapin, Keller, Pigoski, & Zvegintzov, 1996).

Cost

Although there is no agreement on the actual cost of software maintenance, data exists to support the fact that maintenance costs account for a large portion of overall software development life cycle costs (Pigoski, 1996). For example, the U.S. Department of Defense (DOD) spends approximately \$30 billion per year on software applications and estimates two thirds of that amount is for maintenance (Wolfinger, Taub, Stark, Holtzblatt, Dhama, & Cho, 1996). In the early 1980s, the software industry did not understand the costs involved in software maintenance. Today, the journal articles regularly appear that detail maintenance cost factors and how to reduce them.

Standards

While standards for software development have existed for many years, it was not until 1993 that the IEEE Standard on Software Maintenance (IEEE 1219) was published (Wolfinger et al., 1996). This standard provided maintainers with a common framework and process for software maintenance. IEEE 1219 was followed in 1995 by the publication of ISO/IEC 12207 and its maintenance process. Maintenance standards are now an active area, and proposed changes to enhance these standard maintenance methodologies are frequent (Polo, Piattini, Ruiz, & Calero, 1999).

Tools

Industry leaders heralded the use of CASE tools to facilitate software maintenance and reduce increasing maintenance costs (Schneidewind et al., 1996). However, in many cases organizations failed to adopt the technology and the tools have

not lived up to expectations. The development and use of integrated CASE (ICASE) tools is another attempt by the industry to address the problem of increasing costs (Pigoski, 1996). ICASE tools are a combination of two or more CASE tools to form an integrated whole.

Process improvement

Process improvement efforts are beginning to impact the field of software maintenance (Schneidewind et al., 1996). New software development approaches are tailored for use in maintenance, and maintainers are familiar with process improvement efforts. For example, reduced maintenance effort that is the result of development processes utilizing structured programming, report generators, and packaged software. In contrast, development processes using software code generators yield increased maintenance effort (Slaughter & Banker, 1996).

Outsourcing

In the past, software developers performed their own maintenance (Pigoski, 1996). However, outsourcing the maintenance function (primarily corrective and adaptive) has yielded cost savings for organizations. A current industry best practice is the utilization of full-time, trained maintenance specialists instead of untrained generalists (Jones, 2000). The positive impact of this practice is one of the reasons why maintenance outsourcing is growing rapidly.

Summary

The results chapter presented above began with an investigation into the problems encountered during software maintenance. These included the need for more effective software maintenance tools, a lack of up-to-date software documentation, the low status of maintainers, and the lack of a design-for-maintenance viewpoint in the software development phase. This was followed by a discussion of positive developments in the areas of cost, standards, tools, process improvement, and outsourcing.

Chapter 5 Conclusion

The maintenance of software applications is one of the major problems in the software development life cycle. The maintenance process is costly, conflictive, and extremely resource intensive (Polo et al., 1999). The practice of software maintenance has improved significantly in past 15 years (Bennett, Griffiths, Brereton, Munro, & Layzell, 1996). However, software applications are becoming larger and more complex. In addition, user requirements dictate the flexibility to meet changing business needs. As more and more of these systems go online, the pressure to continuously improve and adapt them in response to user requests will continue to burden software maintenance organizations.

In previous chapters, the four maintenance categories were discussed; the use of maintenance tools to reduce cost and improve efficiency was explored; software maintenance problems were investigated; and positive industry developments were explained. In the future, the software industry must continue to focus on improving software maintenance processes. This can be accomplished through the implementation of industry best practices.

Recommended best practices include software that is designed with maintainability in mind. This requires looking at programs and programming from the perspective of a programmer about to alter the code. In addition, as systems become more complex and the technology they use becomes more obscure, software organizations must elevate the position of software maintainer and encourage the best programmers to become maintainers. Finally, as the best programmers become dedicated to software maintenance it is only reasonable to equip them with the best maintenance tools and procedures available.

Summary

The research paper presented above was a descriptive study formatted in five chapters. The first chapter covered the paper's problem statement and goal, relevance, and format. This was followed in the second chapter by a review of the literature relevant to the problem. In the third chapter, the research methods and online tools and resources employed during the completion of the paper were described. The fourth chapter presented the results of the research and provided an analysis of software maintenance problems and positive developments in the field. Finally, the last chapter provided a summary of software maintenance along with recommendations for improvement.

References

- Bennett, K., Griffiths, D., Brereton, P., Munro, M., & Layzell, P. (1996). *Software maintenance for 2005*. Paper presented at the 1996 International Conference on Software, Monterey, CA.
- Burrows, J. (1984). Guideline on Software Maintenance. *U.S. Department of Commerce*. Retrieved May 28, 2001, from the World Wide Web: <http://www.jcte.jcs.mil/htdocs/teinfo/directives/pub/pub92f.html>.
- Chapin, N. (2000a). *Do we know what preventative maintenance is?* Paper presented at the International Conference on Software Maintenance (ICSM'00), San Jose, California.
- Chapin, N. (2000b). *Software Maintenance Types - A Fresh View*. Paper presented at the International Conference on Software Maintenance (ICSM'00), San Jose, California.
- Cioch, F., & Palazzolo. (1997). *A documentation suite for maintenance programmers*. Paper presented at the 1996 International Conference on Software, Monterey, California.
- Copernic (2001). Copernic 2001. Retrieved June 6, 2001, from the World Wide Web: <http://www.copernic.com>.
- Dart, S., Christie, A., & Brown, A. (1993). A Study in Software Maintenance. *Carnegie Mellon Software Engineering Institute*. Retrieved May 28, 2001, from the World Wide Web: <http://www.sei.cmu.edu/publications/documents/93.reports/93.tr.008.html>.
- Domsch, M., & Schach, S. (1999). *A case study in object-oriented maintenance*. Paper presented at the IEEE International Conference on Software Maintenance, Oxford, England.
- Dumke, R., & Winkler, A. (1997). *CAME tools for an efficient software maintenance*. Paper presented at the 1st Euromicro Working Conference on Software Maintenance and Reengineering, Berlin, Germany.
- Fioravanti, F., Nesi, P., & Stortoni, F. (1999). *Metrics for controlling effort during adaptive maintenance of object oriented systems*. Paper presented at the IEEE International Conference on Software Maintenance, Oxford, England.
- Garg, S., Puliafito, A., Telek, M., & Trivedi, K. (1998). Analysis of preventive maintenance in transactions based software systems. *IEEE Transactions on Computers*, 47(1), 96-107.

- Glass, R. (1989). *Software maintenance documentation*. Paper presented at the Annual ACM Conference on Systems Documentation, Pittsburgh, PA.
- Jones, C. (2000). The Economics of Software Maintenance in Twenty First Century. *Object-Z Systems*. Retrieved May 28, 2001, from the World Wide Web: <http://www.objectz.com/columnists/capers/>.
- Kammer, R. (2000). *Software maintenance in the new millennium: Issues and challenges*. Paper presented at the International Conference on Software Maintenance (ICSM'00), San Jose, California.
- Lethbridge, T., & Singer, J. (1997). *Understanding software maintenance tools: Some empirical research*. Paper presented at the IEEE Workshop on Empirical Studies of Software Maintenance (WESS'97), Bari, Italy.
- LexiBot (2001). Our Technology - Results: The LexiBot Expression. *BrightPlanet*. Retrieved June 6, 2001, from the World Wide Web: <http://www.brightplanet.com/technology/results2.asp>.
- Mattsson, M. (1998). *A conceptual model of software maintenance*. Paper presented at the 1998 International Conference on Software Engineering, Kyoto Japan.
- Mayrhauser, A., & Vans, A. (1997). *Program understanding needs during corrective maintenance of large scale software*. Paper presented at the 21st International Computer Software and Applications, Washington, DC.
- Mockus, A., & Votta, L. (2000). *Identifying reasons for software changes using historic databases*. Paper presented at the International Conference on Software Maintenance (ICSM'00), San Jose, California.
- Oreizy, P., Gorlick, M., Taylor, R., Heimbigner, D., Johnson, G., Medvidovic, N., Quilici, A., Rosenblum, D., & Wolf, A. (1999). An architecture-based approach to self-adaptive software. *IEEE Intelligent Systems*, 14(3), 54-62.
- Pigoski, T. (1996). *Practical software maintenance: Best practices for managing your software investment*. New York: Wiley Computer Publishing.
- Pigoski, T. (2001). Software Maintenance. In A. Abran & J. Moore (Eds.), *Guide to the Software Engineering Body of Knowledge*. New York: IEEE.
- Polo, M., Piattini, M., Ruiz, F., & Calero, C. (1999). *MANTEMA: A software maintenance methodology based on the ISO/IEC 12207 standard*. Paper presented at the Fourth IEEE International Symposium and Forum on Software Engineering Standards, Curitiba, Brazil.

- Rajlich, V., & Adnapally, S. (1996). *VIFOR 2: a tool for browsing and documentation*. Paper presented at the 1996 International Conference on Software, Monterey, CA.
- Rayside, D., Kerr, S., & Kontogiannis, K. (1998). *Change and adaptive maintenance detection in Java software*. Paper presented at the Fifth Working Conference on Reverse Engineering, Honolulu, Hawaii.
- Sawyer, P., Sommerville, I., & Viller, S. (1999). Capturing the benefits of requirements engineering. *IEEE Software*, 16(2), 78-85.
- Schneidewind, N., Chapin, N., Keller, T., Pigoski, T., & Zvegintzov, N. (1996). *How much has software maintenance changed since 1983?* Paper presented at the 1996 International Conference on Software, Monterey, CA.
- Sharon, D. (1996). Meeting the challenge of software maintenance. *IEEE Software*, 13(1), 122-126.
- Slaughter, S., & Banker, R. (1996). *A study of the effects of software development practices on software maintenance effort*. Paper presented at the 1996 International Conference on Software Maintenance (ICSM'96), Monterey, California.
- Smith, D. (1999). *Designing maintainable software*. New York: Springer.
- SRC. (1999). *RETIRE: Automatically Converts Old 4GL Code to COBOL*. Retrieved May 28, 2001, from the World Wide Web: <http://www.source-recovery.com/news/1999-pressrel-retire1.htm>.
- Telcordia. (2001). *xVue: A Tool for Effective Software Maintenance*. Retrieved May 28, 2001, from the World Wide Web: <http://xsuds.argreenhouse.com/flyer/xvuefly.html>.
- Vehvilainen, R. (2000). *What is preventative maintenance?* Paper presented at the International Conference on Software Maintenance (ICSM'00), San Jose, California.
- Wolfinger, B., Taub, A., Stark, G., Holtzblatt, L., Dhama, H., & Cho, C. (1996). Controlling the Post-Deployment Cost of Software. *Mitre*. Retrieved May 28, 2001, from the World Wide Web: <http://www.mitre.org/resources/centers/costs/>.