Preface

It is a good thing to practice some bad habits such as smoking, eating pork, drinking over your limit or not doing any physical exercise, so that if one day you fall ill, your doctor, to make recover, will have something to ban. But if you are all virtue, you will have no further room for improvement and falling ill will take you to your death bed.

Luis Landero, in *Games of the Late Age*.

The choice of the quotation that starts this preface has not been casual. In fact, software in execution suffers many bad habits that, fortunately for software services companies, produce more and more work every year. From the point of view of software maintenance, such imperfections have their origin in the software itself, when some defects must be removed; in users, when they ask for new functionalities to be added; and in the changing technological environment, when the software must adapt to a new environment. On the other side, and mapping Lehman Laws (Lehman, 1980) with the last sentence of the quotation, non-changing software is non-used, dead software.

According to the ISO/IEC (1995) terminology, the software maintenance process is activated "when the software product undergoes modifications to code and associated documentation due to a problem or the need for improvement or adaptation." In spite of this definition, which is very similar to that of ANSI-IEEE (1990), the ignorance of maintenance activities may lead to underestimating its importance, since there is a tendency to associate software maintenance only with corrective activities. However, several authors (McKee, 1984; Frazer, 1992; Basili et al., 1996; Polo, Piattiani, & Ruiz, 2001) have shown that perfective interventions receive the most effort of maintenance.

From the seventies, software maintenance is the most costly stage of the software life cycle (see Table 1), and there are no reasons to think that the situation will change, since novel environments and technologies require great maintenance efforts to keep software products in operation. For Brereton, Budgen, and Hamilton (1999), maintenance of hypertext documents will become a serious problem that requires immediate action, since they share many characteristics (structure, development process, economical value) with classical software products. According to Lear (2000), many legacy applications written in COBOL are being adapted to be integrated with current technologies, such as e-commerce.

There are organizations that devote almost all their resources to maintenance, which impedes new development. Moreover, maintenance necessities increase as more software is produced (Hanna, 1993), and its production has al-

Table 1. Evolution of maintenance costs.

Reference	**Year**	**% Maintenance**
Lientz and Swanson (1980)	1976	60%
Pigoski, (1996)	1980-1984	55%
Schach (1990)	1987	67%
Pigoski (1996)	1985-1989	75%
Frazer (1992)	1990	80%
Pigoski (1996)	90's (prev.)	90%

ways shown a growing tendency. On the other side, big programs never are complete, but are always in evolution (Lehman, 1980). Ramil, Lehman, and Sandler (2001) confirm this old theory 21 years later.

PROBLEM CAUSES

In spite of this, software organizations still pay more attention to software development than to maintenance. In fact, most techniques, methods, and methodologies are devoted to the development of new software products, setting aside the maintenance of legacy ones. This problem is also common among programmers, for whom maintenance is "less creative" than development; in fact, many legacy systems use old and boring programming environments, file systems, etc., whereas programmers prefer working with new, powerful visual environments. However, the same software evolves and must continue to evolve along years and, to their regret, programmers devote 61% of their professional life to maintenance, and only 39% to new development (Singer, 1998).

The lack of methodologies may be due to the lack of a definition of the software maintenance process. For Basili et al. (1996), the proposal and validation of new methodologies that take into account maintenance characteristics are a must. Also, Pigoski (1996) says that there is little literature regarding maintenance organizations.

PROPOSED SOLUTIONS

It is clear that maintenance organizations require methodologies and techniques that facilitate software maintenance, decreasing costs and difficulties. There exist different types of partial solutions for software maintenance. Depending on their nature, they can be classified into:

- Technical solutions, that assist in certain moments of maintenance interventions. Reengineering, reverse-engineering or restructuration techniques are some examples.

- Management solutions, that are mainly based on quality assurance, structured management, change documentation and use of specialized human resources.

ORGANIZATION OF THE BOOK

This book collects proposals from some of the best researchers and practitioners in software maintenance, with the goal of exposing recent techniques and methods for helping in software maintenance. The chapters in this book are intended to be useful to a wide audience: project managers and programmers, IT auditors, consultants, as well as professors and students of Software Engineering, where software maintenance is a mandatory matter for study according to the most known manuals—the SWEBOK or the ACM Computer Curricula.

Chapter I, by Ned Chapin, sets a foundation for software maintenance management, analyzing the influence of business rules on maintenance of information systems. Mira Kajko-Mattsson presents in Chapter II the main problems related to the corrective maintenance, and she presents some management solutions for them, strongly supported by a solid theoretical and practical basis.

In Chapter III, Fabrizio Fioravanti analyzes the impact of the recent programming approach of XP on software maintenance, explaining the relationships of eXtreme Programming with maintenance projects. Then, in Chapter IV, Perdita Stevens introduces the idea of using design and process patterns in software maintenance; she also emphasizes the convenience of organizations writing their own set of specific patterns.

William C. Chu, Chih-Hung Chang, Chih-Wei Lu, Hongji Yang, Hewijin Christine Jiau, Yeh-Ching Chung, and Bing Qiao devote Chapter V to maintainability, the desirable property of software systems that would make maintenance work easier. They study the influence of the use and integration of standards on maintainability.

In Chapter VI, Lerina Aversano, Gerardo Canfora, and Andrea De Lucia present a strategy for migrating legacy systems to the Web. After presenting their method, the authors explain an application experience.

In Chapter VII, Norman F. Schneidewind analyzes the relationship between requirements risks and maintainability. It illustrates this study with some empirical results of a number of big spatial projects.

Harry M. Sneed gives us an excellent lesson on software maintenance cost estimation in Chapter VIII, presenting a clear, systematic method for this task.

Macario Polo, Mario Piattini, and Francisco Ruiz devote Chapter IX to present Mantema, a methodology for software maintenance with outsourcing support and explicit definition of process models for different types of maintenance. Mantema is integrated into the MANTIS environment, that is presented in Chapter X by these very same authors and Félix García.

REFERENCES

ANSI-IEEE (1990). ANSI/IEEE Standard 610: IEEE standard glossary of software engineering terminology. New York: The Institute of Electrical and Electronics Engineers, Inc.

Basili, V., Briand, L., Condon, S., Kim, Y., Melo, W. & Valett, J.D. (1996). Understanding and predicting the process of software maintenance releases. In *Proceedings of the International Conference on Software Engineering,* (pp. 464-474). Los Alamitos, CA: IEEE Computer Society.

Brereton, P., Budgen, D. & Hamilton, G. (1999). Hypertext: The next maintenance mountain. *Computer*, *31*(12), 49-55.

Frazer, A. (1992). Reverse engineering-hype, hope or here? In P.A.V. Hall (Ed.), *Software Reuse and Reverse Engineering in Practice* (pp. 209-243) Chapman & Hall.

Hanna, M. (1993, April). Maintenance burden begging for a remedy. *Datamation*, 53-63.

ISO/IEC (1995). International Standard Organization/International Electrotechnical Commission. *ISO/IEC 12207: Information Technology-Software Life Cycle Processes*. Geneve, Switzerland.

Lear, A.C. (2000). Cobol programmers could be key to new IT. *Computer, 33*(4), 19.

Lehman, M. M. (1980). *Programs, life cycles and laws of software evolution. Proceedings of the IEEE, 68*(9), 1060-1076.

Lientz, B.P. & Swanson, E.F. (1980). *Software Maintenance Management.* Reading, MA: Addison Wesley.

McKee, J.R. (1984). Maintenance as a function of design. In P*roceedings of AFIPS National Computer Conference in Las Vegas*, 187-93.

Pigoski, T. M. (1996). *Practical Software Maintenance. Best Practices for Managing Your Investment*. New York: John Wiley & Sons.

Polo, M., Piattini, M. & Ruiz, F. (2001). Using code metrics to predict maintenance of legacy programs: A case study. *Proceedings of the International Conference on Software Maintenance.* Los Alamitos, CA: IEEE Computer Society.

Ramil, J.F., Lehman, M.M. & Sandler, U. (2001). An approach to modelling long-term growth trends in large software systems. *Proceedings of the International Conference on Software Maintenance*. Los Alamitos, CA: IEEE Computer Society.

Schach, S.R. (1990). *Software Engineering*. Boston, MA: Irwin & Aksen.

Singer, J. (1998). Practices of software maintenance. In Khoshgoftaar & Bennet (Eds.), *Proceedings of the International Conference on Software Maintenance,* (pp. 139-145) Los Alamitos, CA: IEEE Computer Society.