Foreword

The topic of software evolution and feedback as it relates to software development is often misunderstood. To make matters worse, the term ‘evolution’ was not used until the 1970s. Thus, the topic is only recently getting its just due within the software engineering community.

Change is endemic to software systems. Requirements change, environments change – and so must the systems that are built. The problem is that often researchers and practitioners do not have sufficient information and understanding about why and how systems progressively change over time. Research has shown that more than 80% of life-cycle costs are incurred after a system is delivered. Therefore, there are significant economic reasons to possess a better understanding of why and how systems evolve.

A pioneer in the field of software evolution, Meir ‘Manny’ Lehman has spent a lifetime conducting research into this difficult topic, and continues to do so. He has also been successful in getting others involved, and the fruits of their collective labor are found within these pages. This book brings together extensive works from significant evolution researchers.

With the publication of this book, readers will now be able to go to one authoritative source to obtain critical information about evolution, including patterns of change, feedback issues and feedback mechanisms. The time is right for this book, as there are no books in print that specifically address software evolution.

This book is a must-read for anyone in the software engineering community. Every software maintenance researcher, practitioner, graduate student and instructor needs to clear space on their bookshelf for a copy, as it will be an invaluable resource you will come back to again and again.

Thomas M. Pigoski
Pensacola, Florida
Preface

‘All is flux, nothing stays still’
Heraclitus (540–480 B.C.)

From Diogenes Laertius, Lives of Eminent Philosophers.

About the field of software evolution and feedback

Since the advent of electronic computers, the growing software development community is increasingly faced with not only how to create a new software application (or system) of the desired quality attributes (e.g., reliability, performance, security, interoperability, portability, etc.) within shrinking timeframes and budgets, but also, following initial development, how to upgrade it so that the users find it satisfactory for their continued and changing needs. In this respect, software is arguably different from most other kinds of products of human endeavor in that a software system needs to be upgraded, time after time, to keep the users satisfied with it in their continually changing operational contexts. In contrast, most other forms of human-made products (though not necessarily their blueprints) do not have to be so modified to keep the users satisfied. Instead, these products are typically repaired to obtain more use from them, or discarded or replaced. From this, a reality with respect to software is that if it is not changed adequately with time then it is probably ‘dying’ if not already dead.

In ‘Software Maintenance and Evolution: a Roadmap’¹, Bennett and Rajlich lucidly depict a staged life-cycle model of a software system. According to this model, following the initial development of a system, it sequentially goes through the stages of evolution, servicing, phase-out and closedown. Though all the stages are critical from the business point of view, the evolution stage is where most of the challenges are in research and practice.

This book is concerned mainly with the evolution of a system following its initial development. It encompasses concern for the manner in which a given system (or a class of systems) progressively changes over time, for example, from the point of view of the system’s size, features, structure and design entropy, documentation size, agent-roles and technological support needed to keep it current, average time or budget needed to add a new unit of software measure (e.g., function point or line of code) into the system and so forth. It also encompasses concern for the technology needed to change the system over time, for example, methods, techniques, tools and processes that not only facilitate the

change at hand but also are sensitive to the long-term ‘health’ of the software system. The term software *evolution* was coined by Lehman and colleagues in the 1970s, whose work has started and, has since, significantly fueled this field.

In software engineering, any kind of change following the initial development or delivery of a software system is widely referred to as software *maintenance* (see, for example, the standards: IEEE 1219 and ISO/IEC 14764). Typically, this includes corrective, adaptive, perfective and preventive changes. Though for practical reasons it is at times difficult to classify a given change among these cleanly (e.g., a corrective change might involve substantial adaptation\(^2\)), it is generally the *adaptive* aspect (to maintain user’s satisfaction with a system in the changing environment) that is of most concern in this book.

From an economic point of view, system adaptation soaks up most of the 80%, or so, of the post-delivery, noncorrective software costs in the life of a software system. There is thus a significant economic reason to improve the science of system adaptation (both managerial and technical aspects) and of adaptive technologies so that, ultimately, the running systems would be healthier, last longer, and would result in increased benefits to the end-users, customers and other stakeholders.

In his eloquent description on Software Maintenance, as part of the ‘Software Engineering Body of Knowledge (SWEBOK)’ sponsored by the IEEE Computer Society, Pigoski identifies maintenance techniques such as program comprehension, reverse engineering, re-engineering and impact analysis. Also, in the cited ‘Software Maintenance and Evolution: a Roadmap’, Bennett and Rajlich identify yet other areas of importance such as linking software attributes to maintainability, change and system validation. While all such topics are considered to help in the analysis and implementation of the change at hand, they do not generally consider the dimension of *time* spanning multiple releases of a software system and are thus *not* represented in this book. Do note, however, that there is an emergence of interest in linking topics such as code exploration and visualization and aspect-oriented software development to software evolution as visualized over multiple releases. The field is indeed not staying still!

Nonetheless, as the technologies and development paradigms have changed over time, for example, from main-frames to workstations to web and pervasiveness; or from solely closed-source (or proprietary) development to increasingly open-source (or community-based) development; or from fine-grained (or lines-of-code based) development to coarse-grained (or component based) development to loosely coupled system integration; or from purely function orientation to the emergence of service orientation; or from traditional life-cycle models to the emergence of agile methods and no doubt other such transitions and advances, so has the intricacy of what constitutes the field of software evolution. In turn, this has presented an open-ended opportunity to researchers and practitioners alike to grasp new challenges in software evolution with innovations in system design, development technologies and management.

Software evolution would arguably be nonexistent without the impetus for change, which results from a wide variety of reasons, such as technological changes; new needs of the stakeholders; changes in laws, rules and regulations and others. Thus, an important driver here is *feedback* concerning the relevance of an existing system in its operationally

changing contexts. Gathered typically from the field of use, such feedback consists of, amongst other things, where the system is failing to meet the expectations and what new functional, nonfunctional and behavioral features are needed to keep the system current.

As can be easily imagined, feedback involves not only the ‘what’ (as in the type of information fed back from the field to development) but also other dimensions described above: the processes needed to effectively feed back, monitor and control information, the organizational structures and technologies needed to support the processes, the roles played by people in the feedback processes, the cost and benefits of feedback, and so forth. Opportunities for innovation in research and practice abound here as well, as new paradigms and technologies are embraced by society. Let us end this description on evolution and feedback with Lehman’s Eighth law, which says that evolution processes are multi-level, multi-loop, multi-agent feedback systems.

About this book

This book provides a depth of material in the field of software evolution and feedback. It focuses on the concepts, theory and practice underlying the numerous changes a system typically undergoes to keep it in productive use over its lifetime.

In particular, this book describes the phenomenological underpinnings; concepts in the software products and software organizations that encircle evolutionary changes; patterns of change discovered through statistical analysis of object-oriented systems; how requirements change over time due to external factors; characteristics of open-source software evolution; the role of ripple effects; the relationship between software architectures and software evolution; the evolution of object-oriented frameworks; formal aspects of software evolution; feedback issues in the software process; use of policies to guide software evolution; feedback in requirements elicitation; the role of metrics that characterize the risk of making requirements changes; how process feedback can be combined with discrete event simulation models to support software project management; feedback and feed-forward capabilities to aid software estimation and learning; self-adaptive software through internalized feedback, and rules and tools for software system process planning and management.

While, clearly, not every conceivable topic in the field of evolution and feedback could possibly be treated in any book of this nature, the coverage provided by the book’s 27 chapters is significant. The scope, however, is constrained not only by the lack of body of knowledge in many areas of evolution and feedback but also by other considerations such as time and effort required to create the content, and the size and cost of the book. Still, in what appears in between the covers, many chapters are original, written specifically for this book, while others are significant revisions of earlier publications.

In fact, the book has several goals:

- To capture and disseminate a substantial body of knowledge in the area of software evolution and feedback that represents a cross-section of modern research.
- To promote the book’s subject in a learning environment.
- To promote critical thinking on different topics in software evolution and feedback.
- To precipitate ways to improve the practice of software development and evolution.
- To precipitate further research in this field.
These goals are pursued through different means. For example, the different perspectives of the field of software evolution and feedback are brought together in this book through the contributions made by numerous researchers of international standing. There are presentation slides that ease the dissemination of the book’s content in a learning or group environment, and suggested questions, model responses and discussion points to promote critical thinking and dialogue within a group. Also, improved understanding of the life of a software system after its initial development can help improve practice through adoption of new concepts, technical and managerial processes, technologies and organizational structures in the workplace. Lastly, researchers now have a significant source of knowledge in the area of software evolution and feedback, all in one book, to support their investigation.

Level and readership

The book is intended for researchers in software engineering; senior practitioners and consultants in the software industry who face software evolution challenges; graduate students and junior practitioners enrolled in software engineering, computer science, IT and related courses; Masters and Doctoral thesis students in software engineering and computer science; advanced undergraduate students undertaking enrichment studies and final-year projects in software engineering and instructors of software engineering courses.

Reading guide

It is the described duality of evolution and feedback that underlies the structure of this book. Chapters 1 to 16 are ‘evolution’ centered; whereas chapters 17 to 27 are ‘feedback’ centered though both these topics are often discussed in the same chapter. Within these partitions, the chapters are organized from more conceptual to more concrete content. Do note that the book has not been structured to follow any particular route map. It is conceivable that one could read from beginning to end though this may not be a typical reading pattern. Other suggestions to consider are to read the abstracts to select desired chapters, to read chapters involving the same author or the same topic together or to use the index terms to read the related chapters.

Note to the students and instructors, and support material

Though the book is not designed to steer any particular course as laid out by a software engineering curriculum (see, e.g., the Unit ‘EVO – Evolution’ in ‘Computing Curriculum – Software Engineering, May 2004’, The Joint Task Force on Computing Curricula, IEEE Computer Society and Association for Computing Machinery), it is a valuable reference for instructors and students for selecting specific topics on evolution and feedback and fitting them into their own frameworks for learning. The support material of presentation slides and discussion questions and responses are particularly useful for enhancing the learning experience. For access, please see the website: www.wiley.com/go/softwareevolution.

Likewise, senior undergraduate and graduate students are particularly encouraged to undertake ‘enrichment studies’ where possible, suitably guided by supervising instructors.
and involving specific topics of interest covered in the book. Of course, additional sources of information (see below) would add to the richness of such studies. In supervising such studies over many years, our experience suggests that students obtain a deep knowledge of the subject area and tend to have competitive advantage for employment and entrance in advanced degree programs over others who have not conducted such studies.

Online and related resources

Here, we list, in no particular order, additional resources that may enhance the learning experience:

- IEEE International Conference on Software Maintenance (ICSM).
- IEEE European Conference on Software Maintenance and Reengineering (CSMR).
- International Workshop on Principles of Software Evolution (IWPSE).
- International Workshop on Evolution of Large-scale Industrial Software Applications (ELISA).
- IEEE International Workshop on Web Site Evolution (WSE).
- International Workshop on Evaluation & Evolution of Component Composition (EECC)
- IEEE International Workshop on Program Comprehension (IWPC).
- Research Links to Explore and Advance Software Evolution (RELEASE).
- Consortium for Software Engineering (CSER), Canada.
- IEEE Transactions on Software Engineering (TSE).
- ACM Transactions on Software Engineering and Methodology (TOSEM)
- International Conference on Software Engineering (ICSE)
- IEEE International Requirements Engineering Conference (RE).
- ACM Sigsoft Foundations of Software Engineering (FSE)
- Asia-Pacific Software Engineering Conference (APSEC).
- ACM Sigsoft Software Engineering Notes (SEN)

Why this book?

Studies of software evolution and feedback should be central to our knowledge, understanding and practice of software development. Yet, ironically, it has received relatively little attention in the field of software engineering, which tends to focus more on initial development. While a small number of other texts on the subject of software evolution or software maintenance are known, they are either no longer available in print form (e.g., M.M. Lehman and L.A. Belady, Academic Press, 1985) or address the general topics in software maintenance (e.g., J. Martin and C. McClure, Prentice-Hall, 1983; G. Parikh, John Wiley & Sons, 1986; L.J. Arthur, John Wiley & Sons, 1988; T.M. Pigoski, John
Wiley & Sons, 1996; A.A. Takang and P.A. Grubb, Int. Thomson Computer Press, 1996; and general software engineering texts). There is thus a huge gap in the area of software evolution and feedback that this book in part attempts to fill.

The story of the book

This book has humble beginnings. One day, following the fourth ‘Feedback, Evolution And Software Technology’ (FEAST 2000) workshop, which was held in June 2000, Lehman (who was the initiator and chair of the series of four FEAST workshops between 1994 and 2000) invited me to produce the proceedings of the last FEAST workshop based on the accepted position and full papers. Upon analysis of these pre-prints (which were edited by Fernández-Ramil), it seemed to me that there was little new to be gained by producing a hardcopy proceedings because the pre-prints were already available on FEAST’s website. Instead, I proposed a completely new effort in creating an edited book of chapters, involving authors active in the field of software evolution and not necessarily restricted to only those involved with FEAST.

Seeing this as a possible follow-up book to the Lehman–Belady’s book of 1985, Lehman supported the proposed idea but not without cautionary advise on the complexity and volume of the task that would necessarily follow this decision. Despite my guarded optimism, I still underestimated the effort it would take to bring together the works of diverse researchers, many of whom had to start from scratch to submit their chapter. I have no doubt that this effort would have been even more arduous had it not been for the support by the co-editors (Fernández-Ramil and Perry) and Lehman throughout this project. Working with the contributors over numerous iterations, the book has taken nearly five years of elapsed time to complete. It is worth mentioning that a significant contributor to this span of time was that several of us changed institutions and, in one case, changed country as well. While the task has been monumental, it has been a worthwhile experience, the results of which, I hope, will serve the researchers, practitioners and learners in the field.


Nazim H. Madhavji