Preface

An Introduction to the Subject Area

High quality software is of vital importance for the survival and success of companies where many manual tasks are now automated through software, which can provide increased speed, accuracy, assurance, reliability, robustness, and productivity. Software is often a key component of companies' strategic plans for gaining and sustaining competitive advantage. A single undetected error or omission during the software development process could have disastrous consequences during operation. Software errors and omissions can also lead to undesirable outcomes such as reduced customer satisfaction, increased maintenance costs and/ or decreased productivity and profits.

Although information technology can be considered a well-established discipline, software projects are still prone to failure. Even when a software project is not classified as a failure, the general level of software quality leaves much room for improvement. Software review or inspection is one of the important techniques for improving software quality.

In the last thirty years, software reviews have been recommended as one of the most cost effective quality assurance techniques in software process improvements and are widely used in industrial practice. The goal of software review is to improve the quality of the product by reviewing interim deliverables during design and development. It is defined as a "non-execution-based [technique] for scrutinizing software products for defects, deviations from development standards". Most researchers agree that software review is considered the most cost effective technique in cost saving, quality and productivity improvements in software engineering. More specifically, software review can 1) detect defects right through the software development life cycle from concept proposal to implementation to testing; the earlier defects are detected in development, then the easier and less costly they are to remove/correct; and 2) detect defects early in the software development life cycle that are difficult or impossible to detect in later stages; improve learning and communication in the software team, since software development is essentially a human activity.

Overall Objectives and Mission of This Book

The overall objective and mission the proposed book is to provide:

- An understanding of the critical factors affecting software review perfomance.
- Practical guidelines for software reviews.

Readers will gain a deep understanding of current software review literature and theoretical models for analysis software review performance. More specifically, this helps readers to understand the critical input and process factors that drive software review performance. Practical guidelines are drawn from the literature, theoretical models, methodologies, and the results from industry survey and cases studies.

The Scholarly Value of this Book and its Contributions to the Literature in the Information Technology Discipline:

- To increase the understanding of what inputs the typical review process uses in practice.
- To identify the key factors influencing software review performanceTheoretical models help to understand the important relationships between inputs, process, and performance perspective.
- The rigorous quantitative industry questionnaire survey and qualitative (case study: in-depth interviews) case studies are contributed to the software review literature.
- To provide useful and practical guidelines for organizing and conducting software reviews.

Abstract

Information Technology can be considered a well-established discipline, however, software development projects are still prone to failure. Even if a software project is not classified as a failure, the general level of software quality leaves room for improvement. One of the most prevalent and costly mistakes made in software projects today is deferring the activity of detecting and correcting software problems until the end of the project (Boehm & Basili, 2001). Hence, the cost of rework in the later stages of a project can be greater than 100 times that of the project costs (Fagan, 1976; Leffingwell & Widrig, 2000). About 80% of avoidable rework comes from 20% of defects (Boehm & Basili, 2001). As a result, techniques such as software review for improving software quality are important. The current software review literature lacks in empirical evidence on identifying critical inputs and process factors influencing review performance because there is little empirical manipulation of these variables. Where inputs are manipulated, the results are often conflicting and inconsistent. Hence, *what* inputs to use for effective software review in practice is still open to determination. Different input requirements directly affect how the software review is organized.

The overall objective of this book is to explore and understand the critical factors that significantly influence software review performance in practice. In other words, the aim of this book is to further empirically validate the important relationships between software review inputs, process, and performance. Thus, this study is interesting and important for both researchers and practitioners.

The main structures of the book include: literature review, review software, review tools, and technologies, understanding the relationships between inputs, process and software review performance, development of a theoretical model, development of the industry survey plan (instruments (questionnaire), design, pre-tests, sampling, data gathering, data analysis), case study (in-depth interviews of the real life cases), recommendations, and the final writing.

In this book, both quantitative and qualitative methods were employed when collecting and analysing empirical data in order to maximise the reliability and validity of the study. A questionnaire mail survey was arranged with 205 respondents from the software industry in Australia. A cross validation study using an in-depth interview with experts was conducted with five cases (companies). The rich qualitative data from the in-depth interviews and quantitative data (statistical analysis) from the questionnaire survey offers a comprehensive picture of the use of software review in practice. The final conclusion of the book is drawn from a comparative analysis of the quantitative and qualitative results. The empirical data obtained from surveys and in-depth interviews with experts is cross-examined and discussed. The main conclusion of the study is described below.

The current empirical software review studies focus heavily on the explicit inputs (e.g., supporting documents) rather than implicit inputs (reviewer characteristics). However, the survey results in this study suggest that the implicit inputs play a dominant role in software review performance. The findings suggest that the characteristics of the software artefact have no significant direct influence on software review performance and supporting documents have little direct impact on review performance. The results show that only the use of previously reviewed software documents has an effect on software review performance. Interesting results demonstrate that reading techniques and prescription documents have no impact on software review performance. It has previously been argued in the software review literature that reading techniques are considered the most effective explicit input for improving software review performance, however, the survey results show that previously reviewed software documents are more critical than reading techniques documents. Both survey and in-depth interview results suggest that current reading techniques in the software industry are not conclusively beneficial to software review performance. This suggests that reading techniques documents need to be carefully designed and used in practice.

To achieve a higher performance in the software review process, selection of reviewers becomes the most critical factor. These results confirm the theory by Sauer, Jeffery, Land, and Yetton, (2000) and in part, Laitenberger and DeBaud's model (2000). In relation to reviewer motivation, interesting results suggest that motivation, in particular, perceived contingency, is another important factor in the software review process and review performance according to the survey results. However, this variable is often ignored in the empirical software review literature. Although several researchers have recommended that reviewers' motivation should be important in software review performance, to our knowledge, no empirical study has been carried out to support this. The findings suggest that company support, encouragement and reviewer agreement for the way the company conducts software review helps to increase reviewers' motivation and effort and hence improve review performance.

Finally, teamwork is the dominant factor in the review meeting process. The survey results show that teamwork is the best indicator of a successful software review meeting. The more collaborative a review team, the higher the software review performance that can be achieved.

In summary, the key driver to software review performance is reviewers' experience, followed by previously reviewed software documents, perceived contingency (support, encouragement, and reviewer agreement with the company), and teamwork.

Structure of This Book

This book is organised into twelve chapters. Each chapter is briefly summarised as follows:

Chapter I discusses why study software review. The chapter identifies advantages of software review that include improving software quality, cost saving, and productivity. In particular, the chapter presents experts' opinions — the impact of software review on software engineering. In the final section of the chapter, the book addresses the aim of the book and the organization of the book.

Chapter II presents the software review literature including the history of software review, forms of software review structure, and informal review approaches. More specifically, in the literature review, the chapter reviews the six-step Fagan's Software Review (i.e., planning, overview, preparation, group meeting, reworks, and follow-up), form software review structure (i.e., Active Design Review, Two-Person Review, N-fold Review, Phased Review, Use of Review Meeting), IEEE standard for software review, informal review approaches (i.e., Walkthrough, Pair Programming, Peer Check, Pass-Around), and a comparison of formal and informal review approaches.

Chapter III describes tools and technologies for software review. The chapter starts with an explanation of the difference between paper-based and tool-based software reviews, as well as collaborative asynchronous vs. synchronous software review. Followed by an evaluation and comparison of software review tools' features. The chapter identifies the tools features for the group review process. The final section of the chapter reviews a framework for supporting tool-based software processes.

Chapter IV discusses software review tools and how they support the software review process. Tools including: Intelligent Code Inspection in "C" Language (CICLE), Scrutiny, Collaborate Software Inspection (CSI), InspeQ, CSRS, Requirement Traceability (RADIX), InspectA, Asynchronous Inspector of Software Artefacts (AISA), Web Inspection Prototype (WiP), HyperCode, Asynchronous/Synchronous Software Inspection Support Tool (AISSIT), Fine-Grained Software Inspection Tool, CORD, Agent-based Software Tool, Internet-based Inspection System (IBIS), and VisionQuest are discussed in the chapter.

Chapter V presents use of software review inputs, supporting process structure techniques, methods of measuring software review performance, and the limitations of the current software review literature. In particularly, the chapter reviews use of inputs (that include review task, supporting documents, reviewer characteristics), review process (team size, roles design, decision-making method during the review process, and process gain and losses), and qualitative and quantitative methods for performance measurement. The chapter also identifies limitations of the current software review literature.

Chapter VI proposes a theoretical model for analysing software review performance. The Explicit and Implicit Input-process-Output (EIIO) Model is developed for further analysis software review performance. The model includes three major components–inputs, process, and output. Inputs can be classified into explicit inputs and implicit inputs. Explicit inputs refer to software review task (artefact) characteristics and supporting documents. Supporting documents include reading techniques (e.g., checklist, scenarios readings), business reports, prescription documents, and previously reviewed software documents. Implicit inputs include reviewers' ability and their motivations. During the meeting process, the process factors can be classified into communication, teamwork, status effect, and discussion quality. Software review performance is often measured by the number of defects found. The chapter presents the important relationships between inputs, process, and performance. Five propositions between these relationships are discussed in the final section of the chapter.

Chapter VII presents the Industry survey design. In order to understand how practitioners conduct their software reviews in their development environment in software industry, an industry survey is conducted. The industry survey can also validate the theoretical EIIO model. The chapter mainly discusses the industry survey design. A survey plan (i.e., research method, survey design, questionnaire design, measurements of models and scales, sampling techniques, validation of questionnaire procedures, data collection methods, data analysis methods) is detailed described in the chapter.

Chapter VIII discusses industry survey results and findings. The overall survey results provide an understanding of software review in practice and a validation of the proposed EIIO model. This allows better understanding of the direct and indirect relationships between software review inputs, process, and performance. The survey includes four major procedures–response, preliminary analysis, exploratory analysis, and hypotheses tests. The response section discusses response rate, response characteristics, and response bias. The primary analysis focuses on descriptive and missing value analysis whereas, exploratory analysis focuses on reliability and validity of the survey results. The hypotheses tests analysis effects on software review inputs, process, and performance.

Chapter IX discusses the revised EIIO model. This presents interesting results from a comprehensive data analysis procedure. The chapter provides a simple review guide (four steps of conducting software review) after discussions of the revised EIIO model.

Chapter X presents an industry cases study. The case study provides qualitative results and rich information from industry experts' opinions. The method used in the case study is in-depth interview. The data collection procedures and the findings are discussed in the chapter. The findings include 1) issues of conducting software review, 2) common types of software review inputs, 3) discussions of inputs affect review process and performance, and 4) discussions of process affect performance (review outcome).

Chapter XI presents practical guidelines and recommendations for both practitioners and researchers. Useful recommendations of use of inputs, the need for team review meetings and selection measurement metrics (review performance) are provided in the chapter.

Chapter XII concludes contributions and future directions. Theoretical and methodological contributions are addressed. The chapter discusses limitations of the industry studies in this book and future software review directions.

References

Boehm, B. W. & Basili, B. R. (2001). Software defect reduction top 10 list. *IEEE Computer*, *34*(1), January.

Fagan, M. E. (1976). Design and code inspections to reduce errors in program development. *IBM System Journal*, *15*(3), 182-211.

Laitenberger, O. & Debaud, J. M. (2000). An encompassing life cycle centric survey of software inspection. *The Journal of Software and Systems*, *50*(1), 5-31.

Leffingwell, D. & Widrig, D. (2000). *Managing software requirements: A unified approach.* NJ: Addison Wesley.

Sauer, C., Jeffery, R., Land, L., & Yetton, P. (2000). Understanding and improving the effectiveness of software development technical reviews: A behaviourally motivated programme of research. *IEEE Transactions on Software Engineering*, *26*(1), 1-14.