经 典 原 版 书 库

软件可靠性工程

（英文版）

Computing McGraw-Hill

- MORE RELIABLE SOFTWARE
- FASTER DEVELOPMENT

Software Reliability Engineering

John Musa

（美）John D. Musa 著

机械工业出版社
China Machine Press

McGraw Hill Education

经 典 原 版 书 库

软件可靠性工程

(英文版)

Software Reliability Engineering

(美) John D. Musa 著

机械工业出版社
China Machine Press

本书版权登记号：图字：01-2002-0857

图书在版编目（CIP）数据

软件可靠性工程（英文版）/（美）缪撒（Musa, J. D.）著. －北京：机械工业出版社，2003.6

（经典原版书库）

书名原文：Software Reliability Engineering

ISBN 7-111-12321-2

Ⅰ. 软…　Ⅱ. 缪…　Ⅲ. 软件可靠性－软件工程－英文　Ⅳ. TP311.5

中国版本图书馆CIP数据核字（2003）第043893号

机械工业出版社（北京市西城区百万庄大街22号　邮政编码　100037）

责任编辑：杨海玲

北京昌平奔腾印刷厂印刷・新华书店北京发行所发行

2003年6月第1版第1次印刷

787mm × 1092mm 1/16・26 印张

印数：0 001- 3 000 册

定价：49.00 元

Introduction: How to Use This Book

I designed this book with the goal of most efficiently teaching you what software reliability engineering is and how to apply it in software development and testing and software development. My object is to concretely help you deal with the conflicting and very stressful pressures that are probably impinging on you (If they aren't, where do you work and do they have any job openings?!). Software reliability engineering is a skill that can make you more competitive, whether you develop or use software-based systems or are a university student learning to become a software developer or user. The book focuses on practice, presenting methods that have been successfully used in many applications, and avoiding ideas that have not yet been sufficiently proved in actual use.

As you will see, I pay particular attention to testing, but with a very broad perspective. For example, I expect that testers will participate on the system engineering team and will directly interface with users of the software-based product. Also, I envision that many other software development personnel will be involved with testing and must therefore have a broad understanding of it.

I expect that the book will be of special value to you if you are a software tester, software developer, system engineer, system architect, quality assurance engineer, reliability engineer, or development manager of a project that contains software; or, of course, a student preparing for one or more of these roles. It is intended to help you as a text in learning the subject, a deskside companion as you start to apply software reliability engineering, and a reference in handling special situations you may encounter. Thus you will see particular emphasis on simplifying the material and organizing it for easy learning. The organization and presentation of the material evolved through seven years of experience teaching this subject to several thousand practitioners in many different organizations and a wide variety of software applications, and guiding them in learning it. I pay special attention to describing the software reliability engineering process step by step. The table of contents reinforces this hierarchy and makes it easy to find the detail for any step. Finally, I devoted special effort to preparing an index with multiple terms for recalling a topic.

The core sections (sections other than frequently asked questions, special situations, and background) of Chaps. 1 to 6 include only the material that you need to know for the common situations you will encounter in practice. The software reliability engineering process used in practice is described in Chap. 1. The chapter structure of the book reflects the process, each of Chaps. 2 through 6 covering one of the principal activities. I illustrate the process throughout by a unified simple example, Fone Follower. This example is adapted from a real project, but with proprietary data deleted and the material simplified for learning.

Each chapter includes up to three supplementary sections: special situations, frequently asked questions, and background. The special situations sections present techniques that are usually needed only for certain projects and systems. The frequently asked questions sections provide some 350 of the better questions (and answers) that have been posed to me in my experience teaching several thousand practitioners and consulting for various organizations. They represent the backgrounds, ways of learning, and perspectives of different people working on different projects. You may find that some of them correspond with yours. Thus they may help you better understand the topics. Professors should find many of these useful to assign as exercises, in addition to the problems provided at the end of many chapters. The background sections contain supplementary information that can enrich your understanding of the chapter but is not essential to exercising the practice. For example, it may present theory or explanation that justifies or supports various activities of the practice. The order of the supplementary sections is a natural one; you progress from material that occurs sometimes in practice to an elucidation of practice to material that describes the rationale behind the practice but doesn't have to be understood to perform the practice.

Both the frequently asked questions and the background sections may cover topics already treated in the core sections. However, they do so from different perspectives or greater depth. I considered the possibility of integrating this material with the corresponding material in the core sections but deliberately decided not to do so. Most practitioners told me that the core sections should be as simple and short as possible, so that they could learn the essential basics quickly, and that all supplementary material should be separated from them.

Chapter 7 discusses how to deploy software reliability engineering in your organization. The background material needed to enrich your understanding of software reliability models is quite extensive, hence it is covered in a separate chapter (Chap. 8).

Appendix A provides a step-by-step outline of the software reliability engineering process; I recommend you keep it by your desk as a guide and checkoff list the first time you use software reliability engineering. In my classes for practitioners, we learn by doing through workshops. Each workshop has teams of participants organized on project lines. After we cover each chapter, each work group discusses the material related to that chapter and tries to apply it to their project. If university students are working on a sample software engineering project as part of their total course work, the workshops can integrate very nicely with it. I have provided the template that we use to guide the workshops in App. B. This will help guide project teams that are deploying software reliability engineering, particularly if they are using a self-study approach.

Appendix C contains a glossary of terms used in this book and App. D a summary of the few formulas you may need. Appendix E lists the software reliability engineering and testing functions that can be helped with software tools.

Appendix F describes how to use CASRE, the Computer-Aided Software Reliability Estimation tool (Nikora 1994). Although not the only software reliability estimation tool, I chose it because of its convenient graphical user interface and its wide availability through the CD ROM included in the *Handbook of Software Reliability Engineering* (Lyu 1996). CASRE consists of a graphical interface and the SMERFS (Statistical Modeling and Estimation of Reliability Function for Software) software reliability estimation program (Farr and Smith 1992). CASRE developers plan a new version with an improved user interface. We can also expect a new version of SMERFS called SMERFS Cubed (SMERFS 3), with a graphical interface of its own. Both new programs will hopefully be distributed on the Internet (see Software Reliability Engineering web site below).

Some chapters include problems designed to reinforce the material presented; I have provided the answers to these problems in App. G. Appendix H is a sampling of references to papers and articles that have been published by users of software reliability engineering. These papers may be useful to you as models of applications, particularly when the application is similar to yours.

University students, researchers, and others who wish to explore the theory of software reliability engineering in depth will find Musa, Iannnino, and Okumoto (1987) an excellent reference. The *IEEE Transactions on Software Engineering* and the *IEEE Transactions on Reliability* frequently publish papers in this area.

The IEEE Technical Committee on Software Reliability Engineering, a branch of the Software Engineering Technical Council of the IEEE Computer Society, is generally considered the leading professional organization in this field (web site http://www.tcse.org). Among other activities, it publishes a newsletter and it sponsors the annual International Symposium on Software Reliability Engineering. Other relevant professional organizations include the IEEE Reliability Society, the American Society for Quality Control and The European organization ENCRESS (web site http://www.csr.ncl.ac.uk/clubs/encress.html). There is an electronic bulletin board on the Internet (subscribe at vishwa@hac2arpa.hac.com,post to sw-rel@igate1.hac com).

I personally maintain a regularly updated Software Reliability Engineering web site (http://members.aol.com/JohnDMusa/). It provides information on a course I teach based on this book. In the course, you apply software reliability engineering to your own job and receive guidance and feedback. The site also includes a general overview, a section for managers, a list of published articles by those who have applied this practice, a Question of the Month (with answer), information on software reliability estimation programs, links to other sites, and other resources.

Acknowledgments

I am grateful to the many colleagues who have read and commented on part or all of this book, including James Cusick, Willa Ehrlich, Susie Hill, Steve Meyer, Steve Peirce, and Ray Sandfoss of AT&T; Prof. Mary Helanger, Linköping University, Sweden; James Widmaier, National Security Agency; Prof. Mladen Vouk, North Carolina State University; and Prof. Carol Smidts, University of Maryland. I am indebted to Adrian Dolinsky for providing some of the questions in Chap. 7. Particular thanks go to Kathy Yale for the enormous and demanding work of word processing that has made the book a reality.

Feedback from managers and practitioners in my classes has been invaluable; they include such organizations as ATT&T, Cisco Systems, Computing Devices Canada, Eastman Kodak, Hewlett-Packard, Lucent Technologies, Microsoft, Motorola, National Security Agency, Nokia, Tandem Computers, Texas Instruments, and many more too numerous to list.

Some material from Musa, J. D., A. Iannino, and K. Okumoto, *Software Reliability: Measurement, Prediction, Application,* New York: McGraw-Hill, 1987 has been incorporated, generally in modified and updated form, with permission.

John D. Musa

ABOUT THE AUTHOR

John D. Musa is one of the creators of Software Reliability Engineering and an international leader in distilling it into a practice. He originated the concept of execution time, the distinction between faults and failures, the operational profile, and many other concepts. Currently an independent consultant, he was Technical Manager of Software Reliability Engineering at AT&T Bell Laboratories. He has published some 100 papers and delivered more than 175 major presentations in the field. In 1992, he was recognized by the International Software Testing and Analysis Conference as the individual who had contributed the most to software testing technology that year. He is a Fellow of the IEEE and one of the founders of the IEEE Technical Committee on SRE.

Contents

Chapter

1

Overview of Software Reliability Engineering

Be sure to read the Introduction to the book before starting this chapter; it explains the organization of all the chapters.

Software development is plagued with one or more high risks:

1. Unreliability of the released product
2. Missed schedules
3. Cost overruns

These situations can lead to loss of market share and/or loss of profitability. Hence the pressure we developers and testers feel is often overwhelming.

In response to this problem, much attention has been given to the mechanics of development and testing, and many tools have been built to support the process. Researchers have addressed the theory of software development and testing and the many difficult questions it entails. However, we have paid too little attention to the *engineering* of reliability in software-based products. *Engineering* software reliability means developing a product in such a way that the product reaches the "market" at the right time, at an acceptable cost, and with satisfactory reliability. You will note that *market* is in quotes; this is to convey a broad meaning beyond the world of commercial products. Even products built for the military and for governments have a market in the sense that product users have alternatives that they can and will choose if a product is too late, too costly, or too unreliable.

The traditional view of development and testing does not provide us with enough power to achieve this engineering goal. Software reliability engineering takes a much broader, more proactive view. Software reliability engineering, for example, has shown that the most efficient testing involves activities that occur throughout the product life cycle and that interface with system engineering and system design tasks. Software reliability engineering therefore empowers testers to take leadership positions in meeting user needs. It involves system engineers, system architects, potential users, managers (Musa, 1996c), and developers as collaborators (Musa, 1996a, 1996b, 1997c, 1997d, 1997g; Musa and Widmaier, 1996).

The standard definition of *reliability* for software (Musa, Iannino, and Okumoto, 1987) is the probability of execution without failure for some specified interval of natural units or time. Thus we use a definition that is compatible with that used for hardware reliability, although the mechanisms of failure may be different. The compatibility enables us to work with systems that are composed of both software and hardware components.

The product characteristics described—reliability, development time, and cost—are attributes of a more general characteristic—product quality. Product quality is the right balance among these characteristics. Getting a good balance means you must pick quantitative objectives for the three characteristics and measure the characteristics as development proceeds.

Projects, of course, have for some time been able to set objectives for delivery date and cost of products and measure progress toward these objectives. What has been lacking until recently has been the ability to do the same thing for reliability for software-based systems. Since the 1940s, we have been able to set reliability objectives and measure reliability for pure hardware systems. However, the proportion of such systems is now rapidly diminishing to near nonexistence, hence the need for and the development of software reliability engineering.

1.1 What Is Software Reliability Engineering and How Does It Help Development and Testing?

Software reliability engineering is the only standard, proven best practice that empowers testers and developers to simultaneously

1. Ensure that product reliability meets user needs
2. Speed the product to market faster
3. Reduce product cost
4. Improve customer satisfaction and reduce the risk of angry users
5. Increase their productivity

You can use software reliability engineering for any release of any software-based product, beginning at the start of any release cycle. Hence you can easily handle legacy products. We use the term *software-based* to emphasize that there are no pure software systems; therefore, hardware must always be addressed in your analysis. Before applying software reliability engineering to the testing of any product, you must first test (or verify in some other manner) and then integrate the units or modules into complete functions that you can execute.

Software reliability engineering works by applying two fundamental ideas. First, it delivers the desired functionality for the product under development much more efficiently by quantitatively characterizing the expected use of the product and using this information to

1. Precisely focus resources on the most used and/or most critical functions
2. Make testing realistically represent field conditions

Critical means having great extra value when successful or great extra impact when failing. This value or impact can be with respect to human life, cost, or system capability.

Second, software reliability engineering balances customer needs for reliability, development time, and cost precisely and hence more effectively. To do so, it sets quantitative reliability as well as schedule and cost objectives. It engineers strategies to meet these objectives. Finally, software reliability engineering tracks reliability in test and uses it as a release criterion. With software reliability engineering you deliver "just enough" reliability and avoid both the excessive costs and development time involved in "playing it safe" and the risk of angry users and product disaster resulting from an unreliable product.

Software reliability engineering is based on a solid body of theory (Musa, Iannino, and Okumoto, 1987) that includes operational profiles, random process software reliability models, statistical estimation, and sequential sampling theory. Software personnel have practiced software reliability engineering extensively over a period dating back to 1973 (Musa and Iannino, 1991b). At AT&T it has been a best current practice (BCP) since May 1991 (Donnelly, Everett, Musa, and Wilson, 1996). Selection as an AT&T BCP was significant because very high standards were imposed. First, you had to use the proposed practice on several (typically at least 8 to 10) projects and achieve significant, documented benefit to cost ratios, measured in financial terms. You then developed a detailed description of the practice and how you used it on the projects, along with a business case for adopting it. Committees of experienced third- and fourth-level managers subjected the description and the case to a probing lengthy review. Typically, the review lasted several months, with detailed examinations being delegated to first-level software managers and senior software developers. The review of the software reliability engineering BCP proposal involved more than 70 such people. Comments requiring action before final review of the proposal exceeded 100. Even then, the software reliability engineering BCP was only one of five approved from the set of some 30 proposals made in 1991.

In addition, the American Institute of Aeronautics and Astronautics approved software reliability engineering as a standard in 1993, resulting in significant impact in the aerospace industry (AIAA, 1992). A major handbook publisher issued a *Handbook of Software Reliability Engineering* in 1996, further evidence of the field's importance (Lyu, 1996). The IEEE has also been active in developing standards for software reliability engineering.

Organizations that have used software reliability engineering include Alcatel, AT&T, Bellcore, CNES (France), ENEA (Italy), Ericsson Telecom (Sweden), France Telecom, Hewlett-Packard, Hitachi (Japan) IBM, NASA's Jet Propulsion Laboratory, NASA's Space Shuttle project, Lockheed-Martin, Lucent Technologies, Microsoft, Mitre, Motorola, Nortel, North Carolina State University, Raytheon, Saab Military Aircraft (Sweden), Tandem Computers, the U.S. Air Force, and the U.S. Marine Corps, to name just a few. There is a selection of papers and articles by users of software reliability engineering, describing their experience with it, in App. H.

Tierney (1997) reported the results of a survey taken in late 1997 that showed that Microsoft has applied software reliability engineering in 50 percent of its software development groups, including projects such as Windows NT and Word. The benefits they observed were increased test coverage, improved estimates of amount of test required, useful metrics that helped them establish ship criteria, and improved specification reviews.

AT&T's Operations Technology Center in the Network and Computing Services Division has used software reliability engineering as part of its standard software development process for several years. This process is currently undergoing ISO certification. The Operations Technology Center was a primary software development organization for the AT&T business unit that won the Malcolm Baldrige National Quality Award in 1994. At that time, it had the highest percentage of projects using software reliability engineering in AT&T. Another interesting observation is that four of the first five software winners of the former AT&T Bell Laboratories President's Quality Award used software reliability engineering.

The International Definity project represents one application of software reliability engineering. They applied it along with some related software technologies. In comparison with a previous software release