原 版 风 暴 系 列

PEARSON
Addison Wesley

Effective Software Testing

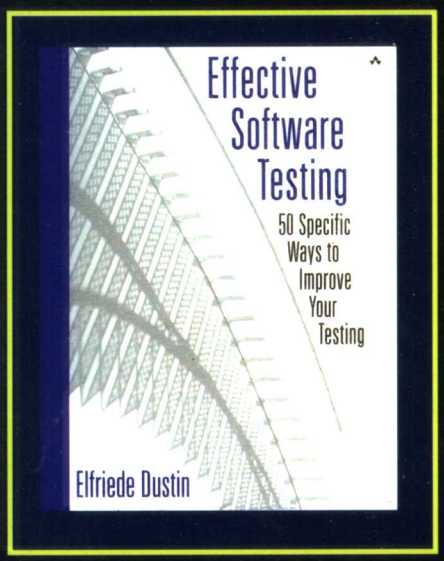
50 Specific Ways to Improve Your Testing

有效软件测试——

提高测试水平的50条建议

（影印版）

[美] Elfriede Dustin 著



- 世界一流软件质量和测试权威作品
- 覆盖软件开发生命周期所有阶段
- 大量真实具体案例有助读者理解

Pearson Education
中国电力出版社
www.infopower.com.cn

Effective Software Testing
50 Specific Ways to Improve Your Testing

有效软件测试——
提高测试水平的50条建议

（影印版）

[美] Elfriede Dustin 著

中国电力出版社

Effective Software Testing:50 Specific Ways to Improve Your Testing (ISBN 0-201-79429-2)
Elfriede Dustin
Copyright © 2003 Addison-Wesley Publishing Company, Inc.
Original English Language Edition Published by Addison Wesley Publishing Company, Inc.
All rights reserved.
Reprinting edition published by PEARSON EDUCATION ASIA LTD and CHINA ELECTRIC POWER PRESS, Copyright © 2003.

本书影印版由 Pearson Education 授权中国电力出版社在中国境内（香港、澳门特别行政区和台湾地区除外）独家出版、发行。
未经出版者书面许可，不得以任何方式复制或抄袭本书的任何部分。

本书封面贴有 Pearson Education（培生教育出版集团）激光防伪标签，无标签者不得销售。
For sale and distribution in the People's Republic of China exclusively (except Taiwan, Hong Kong SAR and Macao SAR).
仅限于中华人民共和国境内（不包括中国香港、澳门特别行政区和中国台湾地区）销售发行。
北京市版权局著作权合同登记号 图字：01-2003-8157

图书在版编目（CIP）数据
有效软件测试——提高测试水平的 50 条建议 / （美）达斯汀著. —影印本.
—北京：中国电力出版社，2003
（原版风暴系列）
ISBN 7-5083-1054-3
Ⅰ.有... Ⅱ.达... Ⅲ.软件－测试－英文 Ⅳ.TP311.5
中国版本图书馆 CIP 数据核字（2003）第 100467 号

丛 书 名：原版风暴系列
书　　名：有效软件测试——提高测试水平的 50 条建议（影印版）
编　　著：（美）Elfriede Dustin
责任编辑：陈维宁
出版发行：中国电力出版社
地址：北京市三里河路 6 号　　邮政编码：100044
电话：（010）88515918　　传　真：（010）88518169
印　　刷：汇鑫印务有限公司
开　　本：787×1092 1/16　　印　张：17
书　　号：ISBN 7-5083-1054-3
版　　次：2004 年 1 月北京第一版　　2004 年 1 月第一次印刷
定　　价：26.80 元
版权所有　翻印必究

Preface

In most software-development organizations, the testing program functions as the final "quality gate" for an application, allowing or preventing the move from the comfort of the software-engineering environment into the real world. With this role comes a large responsibility: The success of an application, and possibly of the organization, can rest on the quality of the software product.

A multitude of small tasks must be performed and managed by the testing team—so many, in fact, that it is tempting to focus purely on the mechanics of testing a software application and pay little attention to the surrounding tasks required of a testing program. Issues such as the acquisition of proper test data, testability of the application's requirements and architecture, appropriate test-procedure standards and documentation, and hardware and facilities are often addressed very late, if at all, in a project's life cycle. For projects of any significant size, test scripts and tools alone will not suffice—a fact to which most experienced software testers will attest.

Knowledge of what constitutes a successful end-to-end testing effort is typically gained through experience. The realization that a testing program could have been much more effective had certain tasks been performed earlier in the project life cycle is a valuable lesson. Of course, at that point, it's usually too late for the current project to benefit from the experience.

Effective Software Testing provides experience-based practices and key concepts that can be used by an organization to implement a successful and efficient testing program. The goal is to provide a distilled collection of techniques and discussions

that can be directly applied by software personnel to improve their products and avoid costly mistakes and oversights. This book details 50 specific software testing best practices, contained in ten parts that roughly follow the software life cycle. This structure itself illustrates a key concept in software testing: To be most effective, the testing effort must be integrated into the software-development process as a whole. Isolating the testing effort into one box in the "work flow" (at the end of the software life cycle) is a common mistake that must be avoided.

The material in the book ranges from process- and management-related topics, such as managing changing requirements and the makeup of the testing team, to technical aspects such as ways to improve the testability of the system and the integration of unit testing into the development process. Although some pseudocode is given where necessary, the content is not tied to any particular technology or application platform.

It is important to note that there are factors outside the scope of the testing program that bear heavily on the success or failure of a project. Although a complete software-development process with its attendant testing program will ensure a successful engineering effort, any project must also deal with issues relating to the business case, budgets, schedules, and the culture of the organization. In some cases, these issues will be at odds with the needs of an effective engineering environment. The recommendations in this book assume that the organization is capable of adapting, and providing the support to the testing program necessary for its success.

ORGANIZATION

This book is organized into 50 separate items covering ten important areas. The selected best practices are organized in a sequence that parallels the phases of the system development life cycle.

The reader can approach the material sequentially, item-by-item and part-by-part, or simply refer to specific items when necessary to gain information about and understanding of a particular problem. For the most part, each chapter stands on its own, although there are references to other chapters, and other books, where helpful to provide the reader with additional information.

Chapter 1 describes requirements-phase considerations for the testing effort. It is important in the requirements phase for all stakeholders, including a representative of the testing team, to be involved in and informed of all requirements and changes. In addition, basing test cases on requirements is an essential concept for any large project. The importance of having the testing team represented during this

试读结束，需要全本PDF请购买 www.ertongbook.com

phase cannot be overstated; it is in this phase that a thorough understanding of the system and its requirements can be obtained.

Chapter 2 covers test-planning activities, including ways to gain understanding of the goals of the testing effort, approaches to determining the test strategy, and considerations related to data, environments, and the software itself. Planning must take place as early as possible in the software life cycle, as lead times must be considered for implementing the test program successfully. Early planning allows for testing schedules and budgets to be estimated, approved, and incorporated into the overall software development plan. Estimates must be continually monitored and compared to actuals, so they can be revised and expectations can be managed as required.

Chapter 3 focuses on the makeup of the testing team. At the core of any successful testing program are its people. A successful testing team has a mixture of technical and domain knowledge, as well as a structured and concise division of roles and responsibilities. Continually evaluating the effectiveness of each test-team member throughout the testing process is important to ensuring success.

Chapter 4 discusses architectural considerations for the system under test. Often overlooked, these factors must be taken into account to ensure that the system itself is testable, and to enable gray-box testing and effective defect diagnosis.

Chapter 5 details the effective design and development of test procedures, including considerations for the creation and documentation of tests, and discusses the most effective testing techniques. As requirements and system design are refined over time and through system-development iterations, so must the test procedures be refined to incorporate the new or modified requirements and system functions.

Chapter 6 examines the role of developer unit testing in the overall testing strategy. Unit testing in the implementation phase can result in significant gains in software quality. If unit testing is done properly, later testing phases will be more successful. There is a difference, however, between casual, ad-hoc unit testing based on knowledge of the problem, and structured, repeatable unit testing based on the requirements of the system.

Chapter 7 explains automated testing tool issues, including the proper types of tools to use on a project, the build-versus-buy decision, and factors to consider in selecting the right tool for the organization. The numerous types of testing tools available for use throughout the phases in the development life cycle are described here. In addition, custom tool development is also covered.

Chapter 8 discusses selected best practices for automated testing. The proper use of capture/playback tools, test harnesses, and regression testing are described.

Chapter 9 provides information on testing nonfunctional aspects of a software application. Ensuring that nonfunctional requirements are met, including performance, security, usability, compatibility, and concurrency testing, adds to the overall quality of the application.

Chapter 10 provides a strategy for managing the execution of tests, including appropriate methods of tracking test-procedure execution and the defect life cycle, and gathering metrics to assess the testing process.

AUDIENCE

The target audience of this book includes Quality Assurance professionals, software testers, and test leads and managers. Much of the information presented can also be of value to project managers and software developers looking to improve the quality of a software project.

Acknowledgments

My thanks to all of the software professionals who helped support the development of this book, including students attending my tutorials on Automated Software Testing, Quality Web Systems, and Effective Test Management; my co-workers on various testing efforts at various companies; and the co-authors of my various writings. Their valuable questions, insights, feedback, and suggestions have directly and indirectly added value to the content of this book. I especially thank Douglas McDiarmid for his valuable contributions to this effort. His input has greatly added to the content, presentation, and overall quality of the material.

My thanks also to the following individuals, whose feedback was invaluable: Joe Strazzere, Gerald Harrington, Karl Wiegers, Ross Collard, Bob Binder, Wayne Pagot, Bruce Katz, Larry Fellows, Steve Paulovich, and Tim Van Tongeren.

I want to thank the executives at Addison-Wesley for their support of this project, especially Debbie Lafferty, Mike Hendrickson, John Fuller, Chris Guzikowski, and Elizabeth Ryan.

Last but not least, my thanks to Eric Brown, who designed the interesting book cover.

Elfriede Dustin

Contents

试读结束，需要全本PDF请购买 www.ertongbook.com

To Jackie, Erika, and Cedric

CHAPTER 1
Requirements Phase

The most effective testing programs start at the beginning of a project, long before any program code has been written. The requirements documentation is verified first; then, in the later stages of the project, testing can concentrate on ensuring the quality of the application code. Expensive reworking is minimized by eliminating requirements-related defects early in the project's life, prior to detailed design or coding work.

The requirements specifications for a software application or system must ultimately describe its functionality in great detail. One of the most challenging aspects of requirements development is communicating with the people who are supplying the requirements. Each requirement should be stated precisely and clearly, so it can be understood in the same way by everyone who reads it.

If there is a consistent way of documenting requirements, it is possible for the stakeholders responsible for requirements gathering to effectively participate in the requirements process. As soon as a requirement is made visible, it can be **tested** and clarified by asking the stakeholders detailed questions. A variety of **requirement tests** can be applied to ensure that each requirement is relevant, and that everyone has the same understanding of its meaning.

Item 1: Involve Testers from the Beginning

Testers need to be involved from the beginning of a project's life cycle so they can understand exactly what they are testing and can work with other stakeholders to create testable requirements.

Defect prevention is the use of techniques and processes that can help detect and avoid errors before they propagate to later development phases. Defect prevention is most effective during the requirements phase, when the impact of a change required to fix a defect is low: The only modifications will be to requirements documentation and possibly to the testing plan, also being developed during this phase. If testers (along with other stakeholders) are involved from the beginning of the development life cycle, they can help recognize omissions, discrepancies, ambiguities, and other problems that may affect the project requirements' testability, correctness, and other qualities.

A requirement can be considered **testable** if it is possible to design a procedure in which the functionality being tested can be executed, the expected output is known, and the output can be programmatically or visually verified.

Testers need a solid understanding of the product so they can devise better and more complete test plans, designs, procedures, and cases. Early test-team involvement can eliminate confusion about functional behavior later in the project life cycle. In addition, early involvement allows the test team to learn over time which aspects of the application are the most critical to the end user and which are the highest-risk elements. This knowledge enables testers to focus on the most important parts of the application first, avoiding over-testing rarely used areas and under-testing the more important ones.

Some organizations regard testers strictly as consumers of the requirements and other software development work products, requiring them to learn the application and domain as software builds are delivered to the testers, instead of involving them during the earlier phases. This may be acceptable in smaller projects, but in complex environments it is not realistic to expect testers to find all significant defects if their first exposure to the application is after it has already been through requirements, analysis, design, and some software implementation. More than just understanding the "inputs and outputs" of the software, testers need deeper knowledge that can come only from understanding the *thought process* used during the specification of product functionality. Such understanding not only increases the quality and depth of the test procedures developed, but also allows testers to provide feedback regarding the requirements.

The earlier in the life cycle a defect is discovered, the cheaper it will be to fix it. Table 1.1 outlines the relative cost to correct a defect depending on the life-cycle stage in which it is discovered.[1]

Table 1.1. Prevention is Cheaper Than Cure: Error Removal Cost Multiplies over System Development Life Cycle

Phase	Relative Cost to Correct
Definition	$1
High-Level Design	$2
Low-Level Design	$5
Code	$10
Unit Test	$15
Integration Test	$22
System Test	$50
Post-Delivery	$100+

1. B. Littlewood, ed., *Software Reliability: Achievement and Assessment* (Henley-on-Thames, England: Alfred Waller, Ltd., November 1987).

Item 2: Verify the Requirements

In his work on specifying the requirements for buildings, Christopher Alexander[1] describes setting up a **quality measure** for each requirement: "The idea is for each requirement to have a quality measure that makes it possible to divide all solutions to the requirement into two classes: those for which we agree that they fit the requirement and those for which we agree that they do not fit the requirement." In other words, if a quality measure is specified for a requirement, any solution that meets this measure will be acceptable, and any solution that does not meet the measure will not be acceptable. Quality measures are used to test the new system against the requirements.

Attempting to define the quality measure for a requirement helps to rationalize fuzzy requirements. For example, everyone would agree with a statement like "the system must provide good value," but each person may have a different interpretation of "good value." In devising the scale that must be used to measure "good value," it will become necessary to identify what that term means. Sometimes requiring the stakeholders to think about a requirement in this way will lead to defining an agreed-upon quality measure. In other cases, there may be no agreement on a quality measure. One solution would be to replace one vague requirement with several unambiguous requirements, each with its own quality measure.[2]

It is important that guidelines for requirement development and documentation be defined at the outset of the project. In all but the smallest programs, careful

1. Christopher Alexander, *Notes On the Synthesis of Form* (Cambridge, Mass.: Harvard University Press, 1964).
2. Tom Gilb has developed a notation, called Planguage (for Planning Language), to specify such quality requirements. His forthcoming book *Competitive Engineering* describes Planguage.

试读结束，需要全本PDF请购买 www.ertongbook.com