TURING 图灵程序设计丛书 微软技术系列

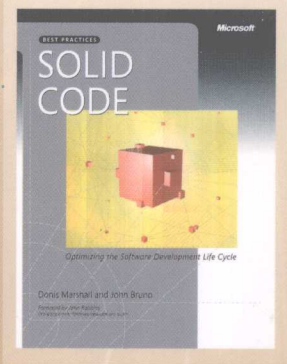
Microsoft

Solid Code

我们在微软怎样开发软件

（英文版）

[美] Donis Marshall
John Bruno 著



- 第一次全面揭示世界软件巨人微软致胜的技术奥秘
- 深入剖析成就高质量代码的四大关键原则
- 软件开发人员的必读秘籍

人民邮电出版社
POSTS & TELECOM PRESS

TURING 图灵程序设计丛书 微软技术系列

Solid Code
我们在微软怎样开发软件
（英文版）

[美] Donis Marshall
John Bruno 著

人民邮电出版社
北 京

图书在版编目（CIP）数据

我们在微软怎样开发软件—Solid Code：英文 /（美）马歇尔（Marshall, D.），（美）布鲁诺（Bruno, J.）著．—北京：人民邮电出版社，2009.6
（图灵程序设计丛书）
ISBN 978-7-115-20679-4

Ⅰ.我… Ⅱ.①马…②布… Ⅲ.软件开发－英文 Ⅳ.TP311.52

中国版本图书馆CIP数据核字（2009）第050617号

内 容 提 要

本书探讨了编写高质量代码的最佳实践，涉及软件开发的各个方面。书中的实用建议来自经验丰富的工程开发人员，这些建议可以应用于设计、原型化、实现、调试和测试等产品开发生命周期的各阶段。同时，本书也提供了微软公司 Windows Live Hotmail 和 Live Search 等团队的真实开发案例。

本书适合各层次软件开发人员阅读。

图灵程序设计丛书

我们在微软怎样开发软件（英文版）

◆ 著 ［美］Donis Marshall John Bruno 著
责任编辑 傅志红

◆ 人民邮电出版社出版发行 北京市崇文区夕照寺街14号
邮编 100061 电子函件 315@ptpress.com.cn
网址 http://www.ptpress.com.cn
北京隆昌伟业印刷有限公司印刷

◆ 开本：800×1000 1/16
印张：21
字数：403千字 2009年6月第1版
印数：1－3 000册 2009年6月北京第1次印刷

著作权合同登记号 图字：01-2009-1525号

ISBN 978-7-115-20679-4/TP

定价：69.00元

读者服务热线：(010)51095186 印装质量热线：(010)67129223
反盗版热线：(010)67171154

版 权 声 明

对本书的赞誉

“这本书很好地兼顾了管理和技术两个方面，内容涉及软件建模、安全设计、防御性编程等。应用书中提供的最佳实践，可以令开发人员的软件开发水平更上一层楼。”

——John Robbins，微软技术大师，Wintellect 创始人之一

“本书超越了编程本身，提供了成功完成高质量软件的大量实用知识。书中将简单易行的最佳实践与来自真实项目的案例研究、经验教训结合起来，引导读者经历从设计到开发、发布和维护的全过程。”

——Janson Blankman，微软公司软件工程师

“作为一名有 20 年软件开发经验的老兵，我仍然有几本书每过上几年就要重读一次。我相信本书将成为其中一员，你不断重读，而且每次都会发现对于软件开发的真知灼见。”

——Don Reamey，微软公司软件工程师

“本书对于任何真正的软件开发人员来说都是无价之宝，其中充满了可以立即令你改善代码质量的实战经验。将它放在你的手边吧，你肯定会不断用到它的！”

——John Alexander，AJI 软件管理合伙人，微软区域总监

“这是每个 IT 专业人士必读的一本书，特别是使用托管代码的开发人员。书中不仅给出了最佳工程实践，并通过实际案例加以解析。”

——Andres Juarez，微软公司产品发布经理

“这是一本杰作，总结了许多最佳实践，可以避免许多常见的开发错误，形成行之有效的软件开发过程。作者为如何发现错误提供了许多实际的解决方案，并且很好地阐述了微软开发人员怎样进行软件开发和测试。”

——Venkat B. Iyeer，微软公司测试经理

“任何层次的开发人员（从初学者到有经验的）都会发现本书是一部佳作，它所阐述的开发实践适用于任何规模的开发团队甚至个人程序员。”

——John Macknight，独立软件工程师

Foreword

Software engineering is not engineering. As a software developer, I would love nothing more than to say I am an engineer. Engineers think through and build things that are supposed to work the first time due to careful planning. So having the word "engineer" in my job title would be very cool indeed.

Let's look at what would happen if the normal software engineering approach were applied to aerospace engineering. A plane is sitting at a gate boarding passengers, and an aerospace engineer—on a whim or forced by management—decides to replace the tail section. Because it's just a tail section, let's just rip it off and stick another one on right there at the gate. No problem, we can make it work! If aerospace engineering were approached like software engineering, I think the passengers would stampede to get off that plane as fast as possible. But those are the kind of changes that are made every day in major software projects the world over. The old joke is that "military intelligence" is an oxymoron, but I'd have to say that it fits "software engineering" as well. What makes this even more troubling to me is that software truly rules the world, but the approach nearly everyone takes to making it can in no way be called engineering.

Why is it that I know the physical computer I'm using right now will work, but the program I'm using, Microsoft Word, will screw up the auto numbering of my lists? While my electrical engineering friends will not be happy to hear this, hardware is easy. The electrical engineer has a limited number of inputs to work with, unlike the essentially unlimited number given to software developers.

Management also considers electrical engineering "real engineering," so management gives the appropriate time and weight to those efforts. The software business, as a distinct field, is not a mature industry; it really hasn't been around that long. In fact, I myself am slightly younger than the software business, so my youthful look reveals some of the problem. If I were as old as electrical engineering, I'd be writing this from the grave.

Another difficulty with software development can sometimes be the software developers themselves. Realistically, the barriers to becoming a software developer can be quite low. I'm a prime example: I was working as a full-time software developer before I had a bachelor's degree in computer science. Because I was able to "talk the talk" in interviews, I was given a job writing software. None of my employers really cared about my lack of education because they could hire me cheaper than someone with a degree.

All real engineering fields require you to achieve ambitious certification criteria before you can add the Professional Engineer (PE) designation to your name. There's nothing like that for the software industry. That's due in part to the fact that no one can agree what all software developers should know because of the newness of the industry. In other fields, the PE

designation appropriately carries huge weight with management. If a certified engineer says a design won't work, she won't sign off on the plans and the project won't go forward. That forces management to take the planning process much more seriously. Of course, by signing off on a project, the PE acknowledges liability for ethical and legal ramifications should things go wrong. Are you ready to sign off on the ethical and legal liability of your software's design? Until we get our industry to that point, we can't really call ourselves engineers in the traditional sense.

The good news is that even in the nearly 20 years I have been in the software development business I've seen huge changes for the better. Senior management is finally getting the message that software project failures cost companies serious amounts of money. Take a look at Robert Charette's "Why Software Fails" in the September 2005 issue of the *IEEE Spectrum* magazine (*http://www.spectrum.ieee.org/sep05/1685)* for a list of spectacular failures. With the costs so high, some senior management are finally committing real resources to get software projects kicked off, planned, and implemented right the first time. We still have a long way to go, but this buy-in for real planning from management is one of the biggest changes I've seen in my time in the industry.

On a micro level, the best change in software development is that nearly all developers are finally serious about testing their code. Now it's fortunately rare to hear about a developer who throws the code over the wall to the QA group and hopes for the best. This is a huge win for the industry and truly makes meeting schedules and quality gates achievable for many teams. As someone who has spent his career on the debugging and performance-tuning side of the business, I'm really encouraged about our industry becoming more mature about testing. Like all good change, the testing focus starts with the individual and the benefits work their way up the organization.

What's also driving change is that our tools and environments are getting much better. With .NET, we have an easy way to test our code, so that means more people will test. Also, the abstraction layers are moving up, so we no longer have to deal with everything on the computer. For example, if you need to make a Web service call, you don't have to manually open the port, build up the TCP/IP packet, call the network driver, wait for the data to return, or parse the return data. It's now just a method call. These better abstraction layers allow us to spend more time on the important parts of any software project: the real requirements and solving the user's problem.

We still have a long way to go before our field is a real engineering field, but the signs are encouraging. I think a big change will occur when we finally start treating testing as a real profession—one that is equal to or more important than development. While I probably won't see the transition to software engineering before I retire, I'm very encouraged by the progress thus far. Let's all keep pushing and learning so we can finally really be called engineers.

This book, *Solid Code*, is a great step in the direction of treating software as an engineering discipline. Bookstores' programming shelves groan under two types of development books. The first kind is the hand-waving software-management type, and the second is the gritty internals-of-a-technology type; I'm guilty of writing the latter. While those books have their uses and are helpful, the types of books we are missing are the ones that talk about real-world team software development. The actual technology is such a small part of a project; it's the team and process aspects that present the biggest challenges in getting a software project shipped. *Solid Code* does a great job of hitting that super hard middle ground between the management books and the technology books. By covering ideas from how to model software to security design to defensive programming, Donis and John, show you how the best practices you can apply to your development will make it even better. Reading *Solid Code* is like experiencing a great project lead by a top development manager and working with excellent coworkers.

The whole book is excellent; I especially loved the emphasis on planning and preparation. Many of the projects that my company, Wintellect, has had to rescue are the direct result of poor planning. Take those chapters to heart so you'll avoid the mistakes that will cost you tons of money and time. Another problem the book addresses is the tendency to leave performance tuning and security analysis for the very end of the project. As the title of Chapter 4 so succinctly points out, "Performance Is a Feature." The recommendations in those chapters are invaluable. Finally, the book's emphasis on real-world coding and debugging will pay dividends even when the code goes into maintenance mode. Even though I've been working in the field nearly 20 years, I picked up a lot of great ideas from *Solid Code*.

Every developer needs to read this book, but there are others in your company who need to read it as well. Make your manager, your manager's manager, and your manager's manager's manager read this book! The one question I always get from senior managers at any company is, "How does Microsoft develop software?" With the Inside Microsoft sections in most chapters of *Solid Code*, your management will see how Microsoft has solved problems in some of the largest applications in use today. Now start reading! It's your turn to help move our industry into a real engineering discipline!

John Robbins
Co-founder, Wintellect

Acknowledgements

Isaac Newton has been credited with the phrase, "If I have seen further, it is only by standing on the shoulders of giants." That statement is certainly applicable to this book, especially when considering the practices, perspectives, and experiences contained within it. More specifically, those shoulders belong to the many people who have contributed to this project. Although our names adorn the cover, we owe much of the credit to the individuals who have helped bring this book to life. We are grateful for their efforts and support throughout this project, and would like to acknowledge them individually.

For starters, we could not have done it without the team at Microsoft Press. We would like to thank Ben Ryan, Devon Musgrave, and Melissa von Tschudi-Sutton for ensuring a high-quality outcome and keeping the project on schedule. Additionally, we would also like to thank the technical editor Per Blomqvist and copy editor Cindy Gierhart for their invaluable contributions and feedback.

As mentioned, this book includes practices, perspectives, and experiences. Many of these elements would not have been included without the contributions, support, and feedback of the professionals from Microsoft and the industry. Specifically, we would like to thank the contributors and reviewers: Jason Blankman, Eric Bush, Jacob Kim, Don Reamey, Dick Craddock, Andres Juarez Melendez, Eric Schurman, Jim Pierson, Richard Turner, Venkatesh Gopalakrishnan, Simon Perkins, Chuck Bassett, Venkat Iyer, Ryan Farber, and Ajay Jha.

There is also a special acknowledgement for Wintellect. Wintellect is a consulting, debugging, and training firm dedicated to helping companies build better software faster through a concentration on .NET and Windows development. Its services include in-depth, multiday .NET on-site and open enrollment training as well as development and consulting services including emergency debugging. The company also produces Devscovery conferences—three-day multitrack events targeting the intermediate to advanced developer. For more information about Wintellect, visit *www.wintellect.com*.

John Robbins and Jeffrey Richter of Wintellect provided invaluable insights and timely feedback. Thanks!

Donis Marshall I have written several books. However, this is my first book with a coauthor. I have been left with one important question after the completing the book. Why did I not have a coauthor on earlier book projects? John Bruno was an incredible asset to this project. His broad knowledge and insights have made this book an important read for any technologist in the Windows arena. John also possesses a rare attribute among authors—timeliness.

John Bruno Writing a book is a commitment that often affects those closest to you. I would like to first thank my wife, Christa, and my two sons, Christopher and Patrick, for their patience, understanding, and sacrifice during the development of this book. Their love

and support inspire me to be the best man I can be, everyday. Additionally, I am grateful to Donis Marshall for inviting me to join him on this project. I sincerely appreciate his friendship and the opportunity to work with him on such an important subject. I have been fortunate throughout my life to have known many creative and insightful people. To those of you who have always been there to inspire, encourage, challenge, and support me, I thank you.

Introduction

Software development has evolved greatly over the past several years. Improvements in programming languages and rapid development tooling, like .NET and Visual Studio 2008, have driven the software industry to build higher-quality software, faster, cheaper, and with more frequent upgrades or refreshes. Despite this continued demand for more software and the evolution in tools and processes, building and releasing quality software remains a difficult job for all participants of software projects, especially developers. Fortunately, this title encapsulates the essence of the best-in-class engineering practices, processes, policies, and techniques that application developers need for developing robust code.

Solid Code explores best practices for achieving greater code quality from nearly every facet of software development. This book provides practical advice from experienced engineers that can be applied across the product development life cycle: design, prototyping, implementation, debugging, and testing. This valuable material and advice is further supplemented by real world examples from several engineering teams within Microsoft, including, but not limited to, the Windows Live Hotmail and Live Search teams.

Who Is This Book For?

Solid Code has something for every participant in the software development life cycle. Most specifically, it is targeted toward application developers who are seeking best practices or advice for building higher-quality software. Portions of this book illustrate the important role of the engineering process as it relates to writing high-quality code. Other parts focus on the criticality of testing. However, most of this book focuses on improving code quality during design and implementation, covering specific topics like class prototyping, performance, security, memory, and debugging.

This book targets both professional and casual developers. Readers should have a basic understanding of programming concepts and object oriented programming in C#. There are no skill level expectations. *Solid Code* is about the practical application of best practices for managed code application development. The topics discussed within the book should resonate with managed code developers of all skill levels.

Organization of This Book

Solid Code is organized similarly to the application development life cycle. The chapters are not separated into parts, but rather grouped according to four key principles. These principles are outlined in Chapter 1, "Code Quality in an Agile World", and include: Focus on Design, Defend and Debug, Analyze and Test, and Improve Processes and Attitudes.

- **Focus on Design** One of the great themes of this book is the importance of thoughtful design as a means to improve overall product quality. To support this theme, practices such as class design and prototyping, metaprogramming, performance, scalability, and security are explored.
- **Defend and Debug** Although great designs are critical to building a high-quality software application, it is equally important to understand the pitfalls that hinder delivery of bug-free code. Topics such as memory management, defensive programming techniques, and debugging are all discussed in the context of this principle.
- **Analyze and Test** Even the greatest programmers produce bugs despite following the recommended best practices. Therefore, it is important to discuss code analysis and testing as methods for further improving code quality.
- **Improve Processes and Attitudes** Beyond best practices, engineering processes and culture can have a great impact on the quality of the work being produced. We explore several key topics for improving the efficiency of the team as well as their passion for quality.

System Requirements

You will need the following hardware and software (at a minimum) to build and run the code samples for this book in a 32-bit Windows environment:

- Windows Vista, Windows Server 2003 with Service Pack 1, Windows Server 2008, or Windows XP with Service Pack 2
- Visual Studio 2008 Team System
- 2.0 gigahertz (GHz) CPU; 2.6 GHz CPU is recommended
- 512 megabytes (MB) of RAM; 1 gigabyte (GB) is recommended
- 8 GB of available space on the installation drive; 20 GB is recommended
- CD-ROM or DVD-ROM drive
- Microsoft mouse or compatible pointing device

The Companion Web Site

This book features a companion Web site that provides code samples used in the book. This code is organized by chapter, and you can download it from the companion site at this address: *http://www.microsoft.com/learning/en/us/books/12792.aspx*.

Find Additional Content Online

As new or updated material that complements this book becomes available, it will be published online to the Microsoft Press Online Developer Tools Web site. This includes material such as updates to book content, articles, links to companion content, errata, sample chapters, and more. This Web site is available at *http://www.microsoft.com/learning/books/online/developer* and it will be updated periodically.

Support for This Book

Every effort has been made to ensure the accuracy of this book and companion content. Microsoft Press provides corrections for books through the Web at the following address:

http://www.microsoft.com/mspress/support/search.aspx

To connect directly to Microsoft Help and Support to enter a query regarding a question or issue you may have, go to the following address:

http://support.microsoft.com

If you have comments, questions, or ideas regarding the book or companion content or if you have questions that are not answered by querying the Knowledge Base, please send them to Microsoft Press using either of the following methods:

E-mail:

mspinput@microsoft.com

Postal mail:

Microsoft Press
Attn: *Solid Code* editor
One Microsoft Way
Redmond, WA 98052-6399

Please note that product support is not offered through the preceding mail addresses. For support information, please visit the Microsoft Product Support Web site at

http://support.microsoft.com

Table of Contents