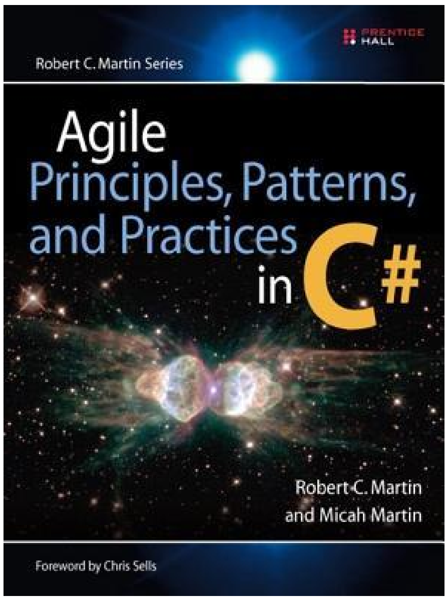

Agile Principles, Patterns, and Practices in C#

Robert C. Martin , Micah Martin

Agile Principles, Patterns, and Practices in C#

Robert C. Martin , Micah Martin

Agile Principles, Patterns, and Practices in C# Robert C. Martin , Micah Martin
With the award-winning book *Agile Software Development: Principles, Patterns, and Practices,* Robert C. Martin helped bring Agile principles to tens of thousands of Java and C++ programmers. Now .NET programmers have a definitive guide to agile methods with this completely updated volume from Robert C. Martin and Micah Martin, ***Agile Principles, Patterns, and Practices in C#.***

This book presents a series of case studies illustrating the fundamentals of Agile development and Agile design, and moves quickly from UML models to real C# code. The introductory chapters lay out the basics of the agile movement, while the later chapters show proven techniques in action. The book includes many source code examples that are also available for download from the authors' Web site.

Readers will come away from this book understanding

Agile principles, and the fourteen practices of Extreme Programming Spiking, splitting, velocity, and planning iterations and releases Test-driven development, test-first design, and acceptance testing Refactoring with unit testing Pair programming Agile design and design smells The five types of UML diagrams and how to use them effectively Object-oriented package design and design patterns How to put all of it together for a real-world project Whether you are a C# programmer or a Visual Basic or Java programmer learning C#, a software development manager, or a business analyst, ***Agile Principles, Patterns, and Practices in C#*** is the first book you should read to understand agile software and how it applies to programming in the .NET Framework.

Agile Principles, Patterns, and Practices in C# Details

Date	:	Published July 30th 2006 by Prentice Hall (first published July 1st 2006)
ISBN	:	9780131857254
Author	:	Robert C. Martin , Micah Martin
Format	:	Hardcover 732 pages
Genre	:	Computer Science, Programming, Software, Technical, Science, Technology, Nonfiction, Reference, Coding, Computers, Unfinished

From Reader Review Agile Principles, Patterns, and Practices in C# for online ebook

Arjay says

With the award-winning book *Agile Software Development: Principles, Patterns, and Practices,* Robert C. Martin helped bring Agile principles to tens of thousands of Java and C++ programmers. Now .NET programmers have a definitive guide to agile methods with this completely updated volume from Robert C. Martin and Micah Martin, ***Agile Principles, Patterns, and Practices in C#.***

This book presents a series of case studies illustrating the fundamentals of Agile development and Agile design, and moves quickly from UML models to real C# code. The introductory chapters lay out the basics of the agile movement, while the later chapters show proven techniques in action. The book includes many source code examples that are also available for download from the authors' Web site.

Readers will come away from this book understanding

Agile principles, and the fourteen practices of Extreme Programming Spiking, splitting, velocity, and planning iterations and releases Test-driven development, test-first design, and acceptance testing Refactoring with unit testing Pair programming Agile design and design smells The five types of UML diagrams and how to use them effectively Object-oriented package design and design patterns How to put all of it together for a real-world project

Whether you are a C# programmer or a Visual Basic or Java programmer learning C#, a software development manager, or a business analyst, ***Agile Principles, Patterns, and Practices in C#*** is the first book you should read to understand agile software and how it applies to programming in the .NET Framework.

About the Author

Robert C. Martin has been a software professional since 1970 and an international software consultant since 1990. He is founder and president of Object Mentor, Inc., a team of experienced consultants who mentor their clients in the fields of C++, Java, OO, Patterns, UML, Agile Methodologies, and Extreme Programming.

Micah Martin works with Object Mentor as a developer, consultant, and mentor on topics ranging from object-oriented principles and patterns to agile software development practices. Micah is the cocreator and lead developer of the open source FitNesse project. He is also a published author and speaks regularly at conferences.

> But Bob, you said you'd be done with the book *last* year.
> *—Claudia Frers, UML World, 1999*

Bob's Introduction

It's been seven years since Claudia's justifiable complaint, but I think I have made up for it. Publishing three books—one book every other year while running a consulting company and doing a lot of coding, training, mentoring, speaking, and writing articles, columns, and blogs—not to mention raising a family and enjoying a grandfamily can be quite a challenge. But I love it.

Agile development is the ability to develop software quickly, in the face of rapidly changing requirements. In order to achieve this agility, we need to use practices that provide the necessary discipline and feedback. We need to employ design principles that keep our software flexible and maintainable, and we need to know the design patterns that have been shown to balance those principles for specific problems. This book is an attempt to knit all three of these concepts together into a functioning whole.

This book describes those principles, patterns, and practices and then demonstrates how they are applied by walking through dozens of different case studies. More important, the case studies are not presented as complete works. Rather, they are designs *in progress.* You will see the designers make mistakes and observe how they identify them as mistakes and eventually correct them. You will see the designers puzzle over conundrums and worry over ambiguities and trade-offs. You will see the *act* of design.

Micah's Introduction

In early 2005, I was on a small development team that began work on a .NET application to be written in C#. Using agile development practices was mandatory, which is one of the reasons I was involved. Although I had used C# before, most of my programming experience was in Java and C++. I didn't think that working in .NET would make much difference; in the end it didn't.

Two months into the project, we made our first release. It was a partial release containing only a fraction of all the intended features, but it was enough to be usable. And use it they did. After only two months, the organization was reaping the benefits of our development. Management was so thrilled that it asked to hire more people so we could start more projects.

Having participated in the agile community for years, I knew a good many agile developers who could help us. I called them all and asked them to join us. Not one of my agile colleagues ended up joining our team. Why not? Perhaps the most overwhelming reason was the fact that we were developing in .NET.

Almost all agile developers have a background in Java, C++, or Smalltalk. But agile .NET programmers are almost unheard of. Perhaps my friends didn't take me seriously when I said we were doing agile software development with .NET, or maybe they were avoiding association with .NET. This was a significant problem. It was not the first evidence I'd seen of this problem, either.

Teaching week-long courses on various software topics allows me to meet a wide cross-section of developers from around the world. Many of the students I've instructed were .NET programmers, and many were Java or C++ programmers. There's no gentle way to put this: In my experience, .NET programmers are often weaker than Java and C++ programmers. Obviously, this is not always the case. However, after observing it over and over in my classes, I can come to no other conclusion: .NET programmers tend to be weaker in agile software practices, design patterns, design principles, and so on. Often in my classes, the .NET programmers had never heard of these fundamental concepts. *This has to change.*

The first edition of this book, *Agile Software Development: Principles, Patterns, and Practices,* by Robert C. Martin, my father, was published in late 2002 and won the 2003 Jolt Award. It is a great book, celebrated by many developers. Unfortunately, it had little impact on the .NET community. Despite the fact that the content of the book is equally relevant to .NET, few .NET programmers have read it.

It is my hope that this .NET edition acts as a bridge between .NET and the rest of the developer community.

I hope that programmers will read it and see that there are better ways to build software. I hope that they will begin using better software practices, creating better designs, and raising the bar for quality in .NET applications. I hope that .NET programmers will not be weaker than other programmers. I hope that .NET programmers achieve a new status in the software community such that Java developers are proud to join a .NET team.

Throughout the process of putting this book together, I struggled many times with the concept of my name being on the cover of a .NET book. I questioned whether I wanted my name associated with .NET and all the negative connotations that seemed to come with it. Yet I can no longer deny it. I am a .NET programmer. No! An agile .NET programmer. And I'm proud of it.

About This Book A Little History

In the early 1990s I (Bob) wrote *Designing Object-Oriented C++ Applications Using the Booch Method.* That book was something of a magnum opus for me, and I was very pleased with the result and the sales.

The book you are reading started out as a second edition to *Designing,* but that's not how it turned out. Very little remains of the original book in these pages. Little more than three chapters have been carried through, and those have been massively changed. The intent, spirit, and many of the lessons of the book are the same. In the decade since *Designing* came out, I've learned a tremendous amount about software design and development. This book reflects that learning.

What a decade! *Designing* came out just before the Internet collided with the planet. Since then, the number of acronyms we have to deal with has doubled. We have EJB, RMI, J2EE, XML, XSLT, HTML, ASP, JSP, ZOPE, SOAP, C#, and .NET, as well as Design Patterns, Java, Servelets, and Application Servers. Let me tell you, it's been difficult to keep the chapters of this book current.

The Booch connection In 1997, I was approached by Grady Booch to help write the third edition of his amazingly successful *Object-Oriented Analysis and Design with Applications.* I had worked with Grady before on some projects and had been an avid reader and contributor to his various works, including UML. So I accepted with glee and asked my good friend Jim Newkirk to help out with the project.

Over the next two years, Jim and I wrote a number of chapters for the Booch book. Of course, that effort meant that I could not put as much effort into this book as I would have liked, but I felt that the Booch book was worth contributing to. Besides, at the time, this book was simply a second edition of *Designing,* and my heart wasn't in it. If I was going to say something, I wanted to say something new and different.

Unfortunately, the Booch book was not to be. It is difficult to find the time to write a book during normal times. During the heady days of the dot-com bubble, it was nearly impossible. Grady got ever busier with Rational and with new ventures such as Catapulse. So the project stalled. Eventually, I asked Grady and Addison-Wesley whether I could have the chapters that Jim and I wrote to include in *this* book. They graciously agreed. So several of the case study and UML chapters came from that source.

The impact of Extreme Programming In late 1998, XP reared its head and challenged our cherished beliefs about software development. Should we create lots of UML diagrams prior to writing any code? Or should we eschew any kind of diagrams and simply write lots of code? Should we write lots of narrative documents that describe our design? Or should we try to make the *code* narrative and expressive so that ancillary documents aren't necessary? Should we program in pairs? Should we write tests before we write production code? What should we do?

This revolution came at an opportune time. During the middle to late 1990s, Object Mentor was helping quite a few companies with OO design and project management issues. We were helping companies get their projects *done.* As part of that help, we instilled into the teams our own attitudes and practices. Unfortunately, these attitudes and practices were not written down. Rather, they were an oral tradition that was passed from us to our customers.

By 1998, I realized that we needed to write down our process and practices so that we could better articulate them to our customers. So I wrote many articles about process in the *C++ Report.*1 These articles missed the mark. They were informative and in some cases entertaining, but instead of codifying the practices and attitudes that we used in our projects, they were an unwitting compromise to values that had been imposed on me for decades. It took Kent Beck to show me that.

The Beck connection In late 1998, at the same time I was fretting over codifying the Object Mentor process, I ran into Kent's work on Extreme Programming (XP). The work was scattered through Ward Cunningham's *wiki*2 and was mixed with the writings of many others. Still, with some work and diligence, I was able to get the gist of what Kent was talking about. I was intrigued but skeptical. Some of the things that XP talked about were exactly on target for my concept of a development process. Other things, however, such as the lack of an articulated design step, left me puzzled.

Kent and I could not have come from more disparate software circumstances. He was a recognized Smalltalk consultant, and I was a recognized C++ consultant. Those two worlds found it difficult to communicate with each other. There was an almost Kuhnian3 paradigm gulf between them.

Under other cir...

Gabrielam13 says

Design patterns explicate în jurul unor exemple de aplica?ii, principiile SOLID exhaustiv prezentate, o introducere general?, dar suficient? a UML, clean design la nivel de componente ?i pachete, toate legate prin lupa principiilor agile sunt p?r?ile constituente ale c?r?ii, pe care le-am savurat ?i sorbit cu mult mai mult entuziasm decât cele din ”Clean code”. Aceasta se poate datora faptului c?, fiind mai ampl?, Uncle Bob ?i-a permis s? explice mai îndetaliat toate principiile, dar ?i faptului c? am înv??at lucruri noi dincolo de poezia deja repetitiv? a codului curat (traducerea sun? amuzant în român?).

R?mân negre?it fana lui Robert C. Martin, dar a?tept cu ner?bdare s? descop?r ?i al?i profesioni?ti în ale program?rii. Am s? m? dedic s? îi urm?resc în continuare video-urile, ale c?ror inten?ii le în?eleg acum mult mai clar.

Joe says

If you can buy only one book on agile development in C#, this is the book you should buy. It is well written and a pleasuer to read. It has many examples in C# illlustrating agile development with patterns.

Sylvain says

Covers a (too) vast territory. Long to read. Not everything is interesting...

Kevin Garner says

After finishing this book and thinking about how useful its contents would be for me in the workplace right now in the (almost) final quarter of 2014, I have concluded that this book is a valuable addition to my programmer bookshelf, albeit a mixed bag of good and stale bits. The good aspects of this book will remain useful. However, the stale parts are... well... a little too stale and beg for a new edition, which it seems Uncle Bob doesn't plan to undertake.

The good points:
The book has aspects that will keep this book on the shelf, ready to crack open at a moment's notice: justification for denouncement of excessive documentation and diagramming, encouragement of realistic / sustainable work hours, explanations of design patterns in a (loosely) C# context, explanations of agile PPPs from an angle I had not previously considered, demonstrations on how to produce reliable software development estimates, emphasis on test-first design / development as the foundation on which this book is written, the list could go on (but not much further).

The stale points:
To be fair, any language specific examples will become stale pretty quickly, as technology is always advancing at breakneck speeds. The Author(s) do state early on that they're not telling you "how to do C#!"; rather, they're extending a lofty olive branch to the .NET developer community (thanks....?) and wanting to discuss more language agnostic topics (refer to the good points listed above). A lot of reviewers point out that C# is really just a namesake in this book. Therefore, let's just go ahead and say it: the C# is really stale; there are such amazing things C# can do now that have made the language much more expressive, maintainable, dynamic, and capable. Funnily enough, some of these advancements were called for in Jack Reeves' article included in Appendix B of this book when he was referring to C++'s success at its onset.

In a nutshell:
This book is worth having around. I will refer back to this book often for inspiration on various design patterns. However, I would like a more modern C# publication on implementing these patterns in a way that actually leverages the language and avoids (now) known anti-patterns. It would be grand if someone took the content of this book to the next level with more modern insight.

After having finished Agile C#, I experienced enlightenment in some ways, but at the same time, I need a C# palate cleanser.

LIBBY says

Awesome read! Great example of how to go about TDD and examples of design patterns.

Jeremy Morgan says

I think this is a pretty decent book. Had I read it 10 years ago I think I would have been really impressed with it.

This book covers a lot of good patterns in it, sprinkled in amongst Agile stuff. I don't think the "fit" was quite made though.

Mid to senior developers may not get as much out of this, because you're likely already familiar with the patterns that are discussed in the book. However, beginners can get a good introduction to the concepts and it's not a complete waste of time.

Some parts were clumsy and the book contradicts itself in places leaving unanswered questions. But it's not a bad book and it contains good information.

Qusai Sabri says

A great book filled with best practices, not easy to read, as a programmer when I pick a book about programming I expect more technical/coding maybe even small projects from A to Z, this book has more definitions and stories,

I would definitely recommend this book to experienced programmers and specially team leaders as they have the ability to apply those design patterns and force the best practices on the whole team.

This book is not for junior programmers (in my opinion), "Clean Code" by the same author might be a better book to read.

Vinicius Saueia says

Excelent book, the exercices/examples are nicely contextualized and the reading flows naturally. The concepts that uncle Bob shows are universal and are not questions of C# only.

Jakub Zalas says

This is THE book that should be read by any developer who claims to be Agile to confront this view with reality. Amazing overview of test driven development, solid principles of object oriented design, rules of package design and practical design patterns.

Tom says

Great, great book. The first section describing agile development is useful for anyone in the software industry, while the remainder of the book is a must-read for all software developers. The descriptions of patterns and principles are thorough yet clearly explained, supplemented by plenty of code and diagrams. Definitely a book that I'll keep nearby when writing code.

Note: Martin's Agile Software Development: Principles, Patterns, and Practices is very similar to this book. I think Agile Principles, Patterns, and Practices in C# is an updated version of the other book, with more chapters on UML diagrams and all of the example code in C#.

Emil Petersen says

I hate to say it, but this book was almost too practical. I don't know how he did it. It starts out really well with great and sensible ways to work together to make software. Then a flood of C# code enters. Every design principle and pattern is nuked with examples, which should be great but somehow is not. I did not need the specifics in code when I read and so it was a bit of a nuisance after the first 5 principles. It's a bit harsh, but the lay of the land is I ended up liking the book, but not more. Would probably still recommend it - depending on how Code Complete turns out.

Ueliton Freitas says

This book is simply spectacular. In my opinion, with its simple and didactic dialogue focused on OO programming, provides the necessary security to develop any software project. Including project management, requisites analysis, architecture, code representation (UML), testing and TDD. All concepts with examples and source code, and even if the reader does not know C#, learning a new language.

Marko Kunic says

Besides design pattern, here you will also learn about package designs, different types of UML diagrams, testing and much more. If you have time give it a read, it is worth it.

Evan Hoff says

This book serves as a great introduction to both sound design and sound development practices. I've recommended this book to several colleagues without hesitation.
