

Introduction

As weird as it might sound, you are reading a book that was born almost accidentally. When we began to work on this material, we weren't even thinking of writing a book. Our initial, quite unpretentious goal was to define a list of guidelines for internal use in Code Architects, the software company we founded in 2002.

We founded Code Architects when we realized the extent of the Microsoft .NET Framework potential and the impact it would have on the developers' community and on the way enterprise-level applications are designed and implemented. Before long, we found ourselves working on software projects that included 400,000 lines of code, mainly written in C# but with some portions developed with Visual Basic. Projects of this size are simply too large for just one or two programmers, and you need more than plain good will to write them in an orderly way. Instead, you need coding discipline and, above all, a set of well-defined and proven guidelines. These guidelines are essential when many developers with different expertise levels and knowledge backgrounds work at the same project.

In the long run, our initial checklist of recommended practices grew in size and included special cases and exceptions meant to accommodate the imperfect world of software development. We also added code examples and short sample projects. But foremost, we discussed nearly all the guidelines in our internal forum and weighed the opinions of Code Architect's team of .NET experts, which includes renowned writers and conference speakers (who are mentioned as they deserve in the acknowledgment section at the end of this introduction).

We finally realized that many developers all around the world might benefit from our efforts, so we proposed the book to Microsoft Press. To our astonishment, they accepted the proposal within a couple of hours. Wow! That was fast, even in the fast world of the Internet!

Mission (Almost) Impossible: Writing Quality Software

The purpose of this book, and of all guideline and best practice collections, for that matter, is to help you write great applications. Which brings up the obvious question: what makes an application a *great* application?

Microsoft Visual Studio and the .NET Framework have simplified the production of software in ways that would have been inconceivable only five years ago. On the other hand, the programming world has become more complex and, above all, more dangerous. In the pre-Internet era, writing a robust program mainly meant protecting your code from end-user mistakes (the mythical *fool-proof* code). Today's users are maybe less naive, but you have to defend your application against malicious hackers, exploits, script kiddies, spoofing, SQL injection, and cross-site scripting attacks, just to name a few of the new threats. Writing secure code is now imperative for Web

xviii **Introduction**

applications and even for applications running in a seemingly and relatively protected environment such as an intranet.

Robustness and security are just the first and most important qualities that great software should have, but there are many other features that concur to make a successful application, for example, scalability and performance. An application should have a coherent user interface and a smooth learning curve so that end users can become productive as soon as possible. Also, a perfect application should be easily customizable and extendable, to meet the end user's expectations and reduce support costs.

Other features aren't visible to the end user, but they are very important from the perspective of the software company that produces it. For example, source code should be written in an orderly and standardized way and should be reusable in other projects if possible. Code reuse is essential in reducing the time-to-market and development costs.

At the end of the day, you see that writing high-quality software isn't impossible, it's only very difficult. You need to be an expert in many programming areas, prepare a state-of-the-art test plan, keep yourself up-to-date on freshly discovered issues and bugs, and keep reading about new product releases. But mostly, you must be very serious about adopting a coherent set of coding guidelines and best practices on which all the developers in your team agree.

Once you are convinced that you need a set of guidelines, you have a practical problem to solve: *where can you find such guidelines?* On the Internet, you can find hundreds of articles and white papers of varying quality and depth, which discuss .NET Framework best practices. Understandably, no single article can cover all the facets of Visual Studio .NET and the .NET Framework, but worse than missing coverage is that many articles contradict each other. In a few cases you can find white papers that explain the advantages and disadvantages of a given recommendation; only rarely do they mention that there might be important exceptions to a given rule.

Microsoft has published some great articles containing what should be considered *the official guidelines* for all .NET developers, and we used them as the basis of our own recommendations (and you'll find many links pointing to them in this book). For example, all developers agree that the name of attribute types should end with *Attribute* and that fields should be made private and be wrapped in public properties. Unfortunately, these official guidelines don't really cover all aspects of the code production process, and the existing gray area has favored the proliferation of all sorts of flames in newsgroups and in blogs.

However, the key word in this book's title is *practical*. We didn't want to create yet another list of guidelines that only the most motivated developers use in their applications. Our goal was to come up with general directives that every developer can customize to fit his or her needs and coding style and to convince you that these rules can help you achieve more than just a pretty-looking piece of code.

We would bet that some developers won't agree with some of our guidelines, but it isn't our intention to fuel endless (and unproductive, in some cases) discussions. To the contrary, we recognize that many guidelines have both advantages and disadvantages, and therefore they shouldn't be considered as absolute guidelines. In fact, you'll often find a **Why** section that explains the pros of a rule and a **Why not** section that explains its shortcomings so that you can make an informed decision. In some cases, you'll even find an **Exceptions** section, where we list the cases when adopting a given guideline would harm your application's overall quality.

It is sometimes difficult to say which rule is the best rule under any given circumstance. This often happens when you discuss a coding style guideline that has no effect on the code's performance, robustness, or maintainability—for example, the prefix you should use for private fields or whether you should put curly braces on a line of their own. In these cases, we decided to add an **Alternative rule** immediately following our “official” guideline, explaining the whys and whynots of each. Our rationale is, when the overall quality won't suffer, it isn't really important which guideline you decide to abide by. Just pick one that fits your coding style, but after you choose it, use it consistently. Your motto should be: *Any guideline is better than no guideline.*

Once again, you might disagree with some guidelines. If so, drop us an e-mail at fbalena@codearchitects.com or gdimauro@codearchitects.com. We're also planning to devote a forum or section on <http://www.dotnet2themax.com> from where you can download more material related to this book and guidelines in general. To stay updated, just subscribe to our newsletter at <http://www.dotnet2themax.com/newsletter/subscribe.aspx>. Guideline Priority

In the first part of this book, we have gathered coding guidelines related to Visual Studio .NET and the two main .NET programming languages, C# and Visual Basic. Here, you'll find naming rules or suggestions about how you might organize your classes, as well as tips about writing code that is both robust and efficient. Most rules apply to both languages, but a few are relevant only under C# or Visual Basic.

In the second part of the book, you can find a vast collection of best practices to be used in the various portions of the .NET Framework, including Windows Forms, ASP.NET, ADO.NET, PInvoke and COM Interop, serviced components, and remoting.

Each of the more than 700 rules in the book can help you in pursuing one of the following objects, listed in decreasing order of importance:

1. **Robustness** Code must work as intended and must not cause fatal errors.
2. **Security** The program must not be subject to attacks from malicious users or, in the case of class libraries, used in an improper way.
3. **Scalability** The application should scale well when the number of users grows, especially in the case of server-side components and database-intensive applications.

xx Introduction

4. **Efficiency** Code must consume few resources and be as fast as possible; it shouldn't unnecessarily stress the CPU, the .NET garbage collector, the file system, and the database engine.
5. **Usability** The application should be easy to use and learn—for example, menu commands should be organized in a logical way and use standard shortcut key combinations. If it is a class library, its programming interface should be coherent and in line with the .NET Framework guidelines.
6. **Code reuse** It should be possible to reuse code in other projects easily.
7. **Extensibility** If necessary, it must be possible to extend a class or the entire application with newer and more powerful versions or to customize it for a specific user, or localize for a different language, with minimal or no impact on applications that have already been deployed.
8. **Maintainability** Source code should be readable; indenting and commenting style should be uniform; member names should be used in a consistent way; code shouldn't rely on undocumented features or behaviors, and so on.
9. **Language interoperability** A class library should be callable from any .NET programming language without any restriction. As a secondary goal, source code should rely as little as possible on specific language features so that it can be reviewed by as many developers as possible and translated to a different language with minimal effort.
10. **Ease of development** When all the available alternatives have no impact on any of the previous points, it is always preferable to adopt a technique that helps to deliver code in less time and that has a narrower margin for human errors. (Yes, developers *are* humans, after all.)

Not surprisingly, some of these goals are often mutually exclusive. For example, a robust and secure application can't be the most efficient application at the same time because the action of checking user input data surely slows down execution speed. Whenever we had to make a choice, we gave a higher priority to goals near the top of the list. For this reason, we usually recommend a more efficient technique only if it doesn't decrease the application's scalability.

At the end of the list, you find a goal that many developers—either consciously or not—place at the top of their own priorities. All developers would love to write great code in little time, but this is rarely a realistic goal. The sad truth is: *writing quality software takes time*. On the other hand, you should be convinced that all the energy you spend to design your software and make it as robust, safe, and reusable as possible pays off when it's time to sell it, support it, or release a new version. At least, this is what we have learned in the last two decades spent writing applications of all sorts.

We haven't cataloged our guidelines according to their importance, and we don't distinguish critical rules from simpler style recommendations. Let us explain why. Except for a few uncontroversial cases, the actual value of a rule depends heavily on the context in which the rule is

applied and, in some cases, on the kind of end user who will run the application. For example, all the rules that have to do with security should be considered critical, but their actual importance depends on whether they are applied to a Web application that is accessed over the Internet, or in an intranet behind a firewall, or in a Windows Forms application installed on the LAN, or in a simple console application meant to be run from the local hard disk.

Instead of using a label or a number that would establish how critical a rule is, we opted for the many flavors of the English language, for example, using adverbs such as *always* and *never* for stronger recommendations, or verbs such as *favor*, *consider*, or *avoid* for rules that might or might not be the best choice in a given circumstance. This approach won't delight those who see a black-and-white world, but it is the most correct one in our opinion.



Important All the code Visual Basic and C# code samples that are longer than a few lines can be downloaded from the Internet. Read Appendix C for more details.

Who Should Read This Book

Many expert developers mistakenly believe that abiding by a set of guidelines is a limitation to their creativity because they believe that writing software is a form of art that shouldn't be caged with these sorts of artificial rules.

Granted, writing software is *also* a creative act. If it weren't, our job would be really tedious and unexciting. But it is also true that you must transform this creative act into manufacturing to create a successful software factory and be competitive in the third millennium. With this goal in mind, you should favor standardization over extreme creativity and personal style because the former improves your productivity and flexibility. For example, if all the developers in your company agree on a given coding style, it is easier to move one or more persons from one project to another without any negative impact on deadlines or code quality.

In writing this book we had the following kinds of readers in mind:

- Individual developers who want to write robust, fast, and scalable code and are already proficient with the .NET Framework and either Visual Basic or C#
- Developers who are asked to write enterprise-level, mission-critical applications that must scale well and resist attacks of all sorts
- Team leaders who wish to define a common set of guidelines for a group of developers with varying expertise levels and coding styles
- Consultants who are asked to debug and review code written by other developers and who need a list of all the most common problems and related solutions



Important One category of developers should *not* buy this book:

This book is neither for beginner developers nor for developers who have never worked with the Microsoft .NET Framework.

Let us state it once again: This book doesn't teach you the basics of the .NET Framework, Visual Basic, and C#. Quite the opposite: we expect that you have a good familiarity with these languages, with object-oriented programming in general, and with the most important types in the .NET Framework.

Where to Look for More Information

This book can't replace a real tutorial on .NET Framework programming. If we had written a tutorial-oriented textbook, space constraints would have forced us to leave out too many important topics. However, we tried to explain all the techniques and concepts that, in our opinion, aren't commonly known among developers.

Fortunately, many great sources of information are available for free on the Internet, ranging from introductions to .NET programming techniques to detailed white papers on very specific topics. We routinely scan the Internet for these articles and have scattered many useful links throughout this book. If you think you need more information about a specific topic or technique, we suggest that you pay a visit to one of the following Web sites:

- The MSDN online Web site (<http://www.msdn.microsoft.com>) is the starting point for all searches related to the .NET Framework.
- The MSDN Magazine Web site (<http://msdn.microsoft.com/msdnmag/>) includes hundreds of great articles on .NET-related topics.
- The Visual Studio Magazine's site (<http://www.visualstudiomagazine.com>) provides access to the current issue and all past issues of this popular magazine. You can also read the .NET-2-the-Max monthly column, written by members of the Code Architects team.
- The .NET-2-the-Max Web site (<http://www.dotnet2themax.com>) is where you can find many tips, articles, and ready-to-use routines written by Code Architects experts, available in both Visual Basic and C# flavors. This is the successor of the popular <http://www.vb2themax.com> site, which has been active since 1999. (We recently launched an Italian version of this site, which you can reach at <http://www.dotnet2themax.it>.)
- The Code Project Web site (<http://www.codeproject.com>) has published many great articles on the .NET Framework, mostly written by and for C# developers.
- The CodeGuru Web site (www.codeguru.com) is another good repository of C#-oriented articles, and it hosts many contributions from Wintellect guys.

- ..And, of course, Google (<http://www.google.com>) is the place to go when everything else fails. Or even before trying anything else!

For a more comprehensive discussion of .NET-related technologies, you might consider the following books:

- *Programming Microsoft Visual Basic .NET 2003* by Francesco Balena (Microsoft Press, 2003), a 1400-page guide for the Visual Basic language and in-depth overview of all portions of the .NET Framework, including Windows Forms, ASP.NET, ADO.NET, serviced components, remoting, and Code Access Security.
- *Applied Microsoft .NET Framework Programming* by Jeffrey Richter (Microsoft Press, 2002), the reference guide for .NET internals such as memory management and exception handling. All examples are written in C#. If you don't feel at ease with curly braces, try out the Visual Basic version: *Applied Microsoft .NET Framework Programming in Microsoft Visual Basic .NET* by Jeffrey Richter and Francesco Balena (Microsoft Press, 2002).
- *Programming Microsoft ASP.NET* by Dino Esposito (Microsoft Press, 2003), a must-have for all serious ASP.NET developers.
- *Windows Forms Programming in C#* by Chris Sells (Addison-Wesley, 2003), with tons of useful techniques for GUI-oriented programmers.
- *Programming .NET Components* by Juval Lowy (O'Reilly, 2003), where you can learn more about components, COM+, and security.

Acknowledgments

First and foremost, we want to thank John Robbins for the most enthusiastic foreword that a book author could ever hope for. Last year, John happened to review a few applications we wrote at Code Architects, so we knew he could prepare an informed foreword. But John went further than that: he actually read every page of the manuscript and provided suggestions and hints, his opinions on controversial guidelines, and ultimately made *Practical Guidelines and Best Practices* a better and more useful book. Having a guru like John Robbins review your book is like having Michael Schumacher give you driving lessons!

Many other persons helped us in creating and polishing this book. At the top of the list is Marco Bellinaso, a great author himself who has also written several best-sellers for the publisher Wrox. Marco tech reviewed the manuscript, tested each and every code snippet (in both Visual Basic and C#, of course), but above all provided countless hints and suggestions about how to improve the material. We think of him more as a coauthor than a tech reviewer.

Other members of the Code Architects team contributed to this book in crucial ways. Enrico Sabbadin helped us to better understand the subtle world of security, remoting, and serviced components; Alberto Falossi supported us in writing many chapters in Part I, and Dino

xxiv Introduction

Esposito made great suggestions to improve the ASP.NET section. We are so proud to work with such a great team of experts.

A big thank you also to Natale Fino, who spent many of his evenings and weekends translating the manuscript into Italian and double-checking it for consistency and accuracy at the same time.

Our initial idea would have never become a concrete book without the behind-the-scenes, relentless work of Ben Ryan, our acquisitions editor at Microsoft Press. An acquisitions editor is a bit like the author's advocate in all the meetings when the publisher decides whether a book should be published, how many pages it should have, and so forth. It's to Ben's credit that we could publish exactly the book we had in mind.

Even if it isn't a very large book, we faced many editing challenges in writing *Practical Guidelines and Best Practices*. For example, many rules reference other rules and there is an unusually high number of cross references that had to be checked. Each rule is split into subsections and we needed to maintain a consistent style across completely different topics. We could have never delivered the manuscript on time if it weren't for Kathleen Atkins and Christina Palaia, our editing guardian angels. It was pure pleasure to work with you.

Many friends and readers ask us where we find the time, energy, and motivation to run a company, create applications, teach, consult, manage a couple of Web sites, write articles and books, *and* have a social life at the same time. In this case, the answer is simple: each of us has a wonderful family behind him.

Thank you Adriana, Francesca, Andrea, and Lucia.

—Francesco Balena, Giuseppe Dimauro