

Foreword

In software development, it's always the little things that cause all the big problems. Everyone has worked on a huge performance bug that held up the release of a product and that turned out to be nothing more than incorrectly using something innocuous in the system. Anyone who has been on a project team, moreover, has certainly taken part in one of those withering religious debates of tabs versus spaces. Even though I have debugged and tuned countless applications, I have yet to find any root causes that are new, unique, or, for that matter, very big. One thing I have found is that nearly all the problems I've worked on occur in the code construction phase. That's right: when the typing starts, the bugs begin.

Although I will, only rarely, find problems that were introduced during design—such as using server-side cursors in a database when they might kill scalability—those bugs nearly always show up quickly, mainly because there are enough “rules of thumb” sites you can Google for on big issues. Those lists are also relatively short. However, when you move down into the coding level, those rules of thumb get harder to find, and the number of rules expands exponentially. Given enough time with Google, you might eventually find all the rules, but you rarely have the time when you've got delivery dates to hit.

What makes life even more interesting for Microsoft .NET Framework developers is that .NET is essentially a brand new technology in the overall scheme of Microsoft Windows development. At the time I write this, .NET was announced publicly only four years ago and shipped only two and a half years ago. Even though Microsoft has bet the farm on .NET, less than half of Windows developers are actually using .NET on their projects today. You might think that's a small percentage, but I find it amazing that companies have moved to .NET so quickly, given its relatively short existence.

In any case, because .NET hasn't been around very long, many developers don't have that internalized “feel” that something they are typing in is going to cause performance problems or bugs, like most experienced C++ or Visual Basic 6.0 developers do. Because, as I mentioned earlier, most developers make nearly all their mistakes during coding, this potentially presents a big problem. Fortunately, *Practical Guidelines and Best Practices* solves this predicament!

Francesco and Giuseppe have brought together all of those best practices rules that you absolutely must follow when writing .NET code. The highest compliment I can pay any book is that I learned something. With this book, I not only learning something; I learned a ton!

Microsoft has promulgated its Design Guidelines, but these fall far short of what's needed in the real world. Francesco and Giuseppe have addressed the entire .NET code construction process and given you the best practices for everything from how to set up your projects to how to best use the XML classes to the best ways to handle ASP.NET state management. When I'm writing .NET code, I always have the book's table of contents open, and I give it a quick scan when I'm about to loop over a collection to see whether I'm going to do it correctly.

xxvi Foreword

Practical Guidelines and Best Practices is like having a .NET expert eXtreme Programming pair developing by your side.

Francesco and Giuseppe are true .NET experts because they've written more .NET code than nearly everyone outside of Microsoft. From the moment Microsoft introduced .NET, Francesco, Giuseppe, and the developers in their company, CodeArchitects, have been working on the largest and most complicated .NET applications I've seen. The more than 700 rules in this book have been learned through developing and, more importantly, shipping, big, real-world applications that use every possible feature in the .NET technology tree. With real teams coming up with these rules, you can rest assured that the rules in this book are not just what some theorist *thinks* are correct; they *are* correct.

After glancing through the book, you're probably thinking that its sole audience is the software developer down in the trenches. Actually, it's also perfect for development leads and project managers. Although you're probably a little confused by that assertion, think about the last code review you sat in. You probably had a group of folks read over the code, but because they weren't looking for anything specific—just the obvious stuff—they didn't find anything useful, and the whole review was probably a waste of time.

Great code reviews look to validate two key items: (1) Does the code meet the requirements set up for the developer? and (2) Does the developer use the technologies and approaches in the environment correctly? As long as your team knows the code requirements, checking for compliance should be relatively easy. Now that *Practical Guidelines and Best Practices* gives you the rules for all those hard .NET Framework and technology interactions, ensuring that the code is as correct as possible is just as easy.

No matter how you use *Practical Guidelines and Best Practices*, it will save you a tremendous amount of time by helping to reduce those insidious bugs and performance problems in your code. I've been using it with all my projects, and it's certainly made me a much better developer.

John Robbins

Cofounder, Wintellect

October 2004