SECOND EDITION

BEGINNING Software Engineering

Rod Stephens

WILEY

BEGINNING SOFTWARE ENGINEERING

Continues

BEGINNING

Software Engineering

BEGINNING

Software Engineering

Second Edition

Rod Stephens

WILEY

Published by John Wiley & Sons, Inc., Hoboken, New Jersey.
Published simultaneously in Canada and the United Kingdom.

ISBN: 978-1-119-90170-9
ISBN: 978-1-119-90172-3 (ebk.)
ISBN: 978-1-119-90171-6 (ebk.)

Library of Congress Control Number: 2022944804

Cover Image and Design: Wiley

ABOUT THE AUTHOR

Rod Stephens started out as a mathematician, but while studying at MIT, he discovered how much fun programming is and he's been programming professionally ever since. He's a long-time developer, instructor, and author who has written more than 250 magazine articles and 35 books that have been translated into many different languages.

During his career, Rod has worked on an eclectic assortment of applications in such fields as telephone switching, billing, repair dispatching, tax processing, wastewater treatment, concert ticket sales, cartography, optometry, and training for professional football teams. (That's US football, not one of the kinds with the round ball. Or the kind with three downs. Or the kind with an oval field. Or the indoor kind. Let's just say NFL and leave it at that.)

Rod's popular C# Helper website (`www.csharphelper.com`) receives millions of hits per year and contains thousands of tips, tricks, and example programs for C# programmers. His VB Helper website (`www.vb-helper.com`) contains similar material for Visual Basic programmers.

You can contact Rod at `RodStephens@csharphelper.com`.

ABOUT THE TECHNICAL EDITOR

John Mueller is a freelance author and technical editor. He has writing in his blood, having produced 122 books and more than 600 articles to date. The topics range from networking to artificial intelligence and from database management to heads-down programming. Some of his current books include discussions of data science, machine learning, and algorithms. He also writes about computer languages such as C++, C#, and Python. His technical editing skills have helped more than 70 authors refine the content of their manuscripts. John has provided technical editing services to a variety of magazines, performed various kinds of consulting, and he writes certification exams. Be sure to read John's blog at `http://blog.johnmuellerbooks.com`. You can reach John on the Internet at `John@JohnMuellerBooks.com`. John also has a website at `www.johnmuellerbooks.com`.

ACKNOWLEDGMENTS

Thanks to David Clark, Christine O'Connor, Kenyon Brown, Judy Flynn, Barath Kumar Rajasekaran, and all of the others who worked so hard to make this book possible. David was this book's project manager. You'll learn what a project manager does in Chapter 4. It's a bit different for writing a book but not as different as you might think. Many thanks for your hard work, David!

Thanks also to technical editor and longtime friend John Mueller for giving me the benefit of his valuable experience. You can see what John's up to at `www.johnmuellerbooks.com`.

Special thanks to Mary Brodie at `https://gearmark.blogs.com` for letting me use her quote at the beginning of Chapter 18, "Iterative Models."

CONTENTS

INTRODUCTION

Programming today is a race between software engineers striving to build bigger and better idiot-proof programs, and the universe trying to build bigger and better idiots. So far the universe is winning.

—Rick Cook

With modern development tools, it's easy to sit down at the keyboard and bang out a working program with no previous design or planning, and that's fine under some circumstances. My VB Helper (`www.vb-helper.com`) and C# Helper (`www.csharphelper.com`) websites contain thousands of example programs written in Visual Basic and C#, respectively, and built using exactly that approach. I had an idea (or someone asked me a question) and I pounded out a quick example.

Those types of programs are fine if you're the only one using them and then for only a short while. They're also okay if, as on my websites, they're intended only to demonstrate a programming technique and they never leave the confines of the programming laboratory.

If this kind of slap-dash program escapes into the wild, however, the result can be disastrous. At best, nonprogrammers who use these programs quickly become confused. At worst, they can wreak havoc on their computers and even on those of their friends and coworkers.

Even experienced developers sometimes run afoul of these half-baked programs. I know someone (I won't give names, but I also won't say it wasn't me) who wrote a simple recursive script to delete the files in a directory hierarchy. Unfortunately, the script recursively climbed its way to the top of the directory tree and then started cheerfully deleting every file in the system. The script ran for only about five seconds before it was stopped, but it had already trashed enough files that the operating system had to be reinstalled from scratch. (Actually, some developers believe reinstalling the operating system every year or so is character-building. If you agree, then perhaps this approach isn't so bad.)

I know another experienced developer who, while experimenting with Windows system settings, managed to set every system color to black. The result was a black cursor over a black desktop, displaying black windows with black borders, menus, and text. This person (who wasn't me this time) eventually managed to fix things by rebooting and using another computer that wasn't color-impaired to walk through the process of fixing the settings using only keyboard accelerators. It was a triumph of cleverness, but I suspect she would have rather skipped the whole episode and had her two wasted days back.

For programs that are more than a few dozen lines long, or that will be given to unsuspecting end users, this kind of free-spirited development approach simply won't do. To produce applications that are effective, safe, and reliable, you can't just sit down and start typing. You need a plan. You need . . . <drumroll> . . . software engineering.

This book describes software engineering. It explains what software engineering is and how it helps produce applications that are effective, flexible, and robust enough for use in real-world situations.

This book won't make you an expert systems analyst, software architect, project manager, or programmer, but it explains what those people do and why they are necessary for producing high-quality software. It also gives you the tools that you need to start. You won't rush out and lead a 1,000-person effort to build a new air traffic control system for the FAA, but it can help you work effectively in small-scale and large-scale development projects. (It can also help you understand what a prospective employer means when he says, "Yeah, we mostly use scrum with a few extra XP techniques thrown in.")

WHAT IS SOFTWARE ENGINEERING?

A formal definition of software engineering might sound something like, "An organized, analytical approach to the design, development, use, and maintenance of software."

More intuitively, software engineering is everything that you need to do to produce successful software. It includes the steps that take a raw, possibly nebulous idea and turn it into a powerful and intuitive application that can be enhanced to meet changing customer needs for years to come.

You might be tempted to restrict software engineering to mean only the beginning of the process, when you perform the application's design. After all, an aerospace engineer designs planes but doesn't build them or tack on a second passenger cabin if the first one becomes full. (Although I guess a space shuttle riding piggyback on a 747 sort of achieved that goal.)

One of the big differences between software engineering and aerospace engineering (or most other kinds of engineering) is that software isn't physical. It exists only in the virtual world of the computer. That means it's easy to make changes to any part of a program even after it is completely written. In contrast, if you wait until a bridge is finished and then tell your structural engineer that you've decided to add two extra lanes and lift it three feet higher above the water, there's a good chance he'll cackle wildly and offer you all sorts of creative but impractical suggestions for exactly what you can do with your two extra lanes.

The flexibility granted to software by its virtual nature is both a blessing and a curse. It's a blessing because it lets you refine the program during development to better meet user needs, add new features to take advantage of opportunities discovered during implementation, and make modifications to meet evolving business requirements. Some applications even allow users to write scripts to perform new tasks never envisioned by the developers. That type of flexibility isn't possible in other types of engineering.

Unfortunately, the flexibility that allows you to make changes throughout a software project's life cycle also lets you mess things up at any point during development. Adding a new feature can break existing code or turn a simple, elegant design into a confusing mess. Constantly adding, removing, and modifying features during development can make it impossible for different parts of the system to work together. In some cases, it can even make it impossible to tell when the project is finished.