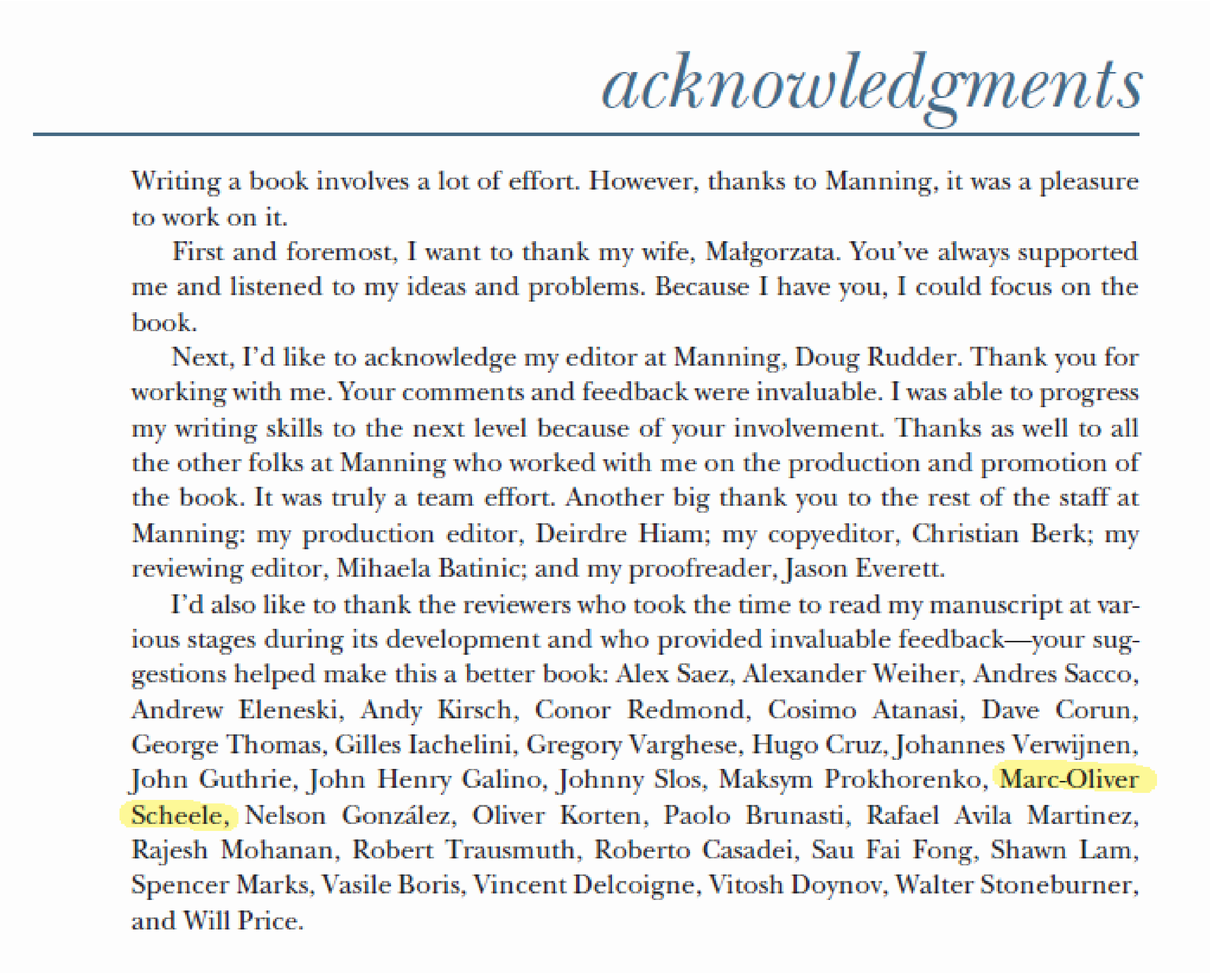
Software Mistakes and Tradeoffs

How to make good programming decisions

Tomasz Lelek
Jon Skeet

MANNING



acknowledgments

Writing a book involves a lot of effort. However, thanks to Manning, it was a pleasure to work on it.

First and foremost, I want to thank my wife, Małgorzata. You've always supported me and listened to my ideas and problems. Because I have you, I could focus on the book.

Next, I'd like to acknowledge my editor at Manning, Doug Rudder. Thank you for working with me. Your comments and feedback were invaluable. I was able to progress my writing skills to the next level because of your involvement. Thanks as well to all the other folks at Manning who worked with me on the production and promotion of the book. It was truly a team effort. Another big thank you to the rest of the staff at Manning: my production editor, Deirdre Hiam; my copyeditor, Christian Berk; my reviewing editor, Mihaela Batinic; and my proofreader, Jason Everett.

I'd also like to thank the reviewers who took the time to read my manuscript at various stages during its development and who provided invaluable feedback—your suggestions helped make this a better book: Alex Saez, Alexander Weiher, Andres Sacco, Andrew Eleneski, Andy Kirsch, Conor Redmond, Cosimo Atanasi, Dave Corun, George Thomas, Gilles Iachelini, Gregory Varghese, Hugo Cruz, Johannes Verwijnen, John Guthrie, John Henry Galino, Johnny Slos, Maksym Prokhorenko, Marc-Oliver Scheele, Nelson González, Oliver Korten, Paolo Brunasti, Rafael Avila Martinez, Rajesh Mohanan, Robert Trausmuth, Roberto Casadei, Sau Fai Fong, Shawn Lam, Spencer Marks, Vasile Boris, Vincent Delcoigne, Vitosh Doynov, Walter Stoneburner, and Will Price.

Core concepts

Top tips in this book

Top tip	Page number	Section
Always validate your assumptions about the code performance, depending on whether it is executed in the single or multithreaded context.	6	1.2
We can calculate the cost of coordination within teams using Amdahl's law.	22	2.2.1
It's hard to use functional exception handling when mixing it with an object-oriented approach. It's even more complicated if the object-oriented code does not declare what exceptions it may throw.	69	3.6.2
We can leverage the findings from the Pareto principle to find the code that brings the most value to our consumers and focus on optimizing that part.	105	5.2.1
Encapsulating the downstream components settings from our clients allows us to evolve without breaking the compatibility of our APIs.	145	6.5
Iterating on date and time requirements with product owners, using concrete examples with as many corner cases as you can think of, makes implementing those requirements much simpler.	169	7.2
Moving computations to data allows us to design big data processing that otherwise would be very slow or not even feasible.	205	8.1.1
It's essential to pick a library with a similar or the exact concurrency model as your application. The scalability and performance of your software will benefit.	238	9.2.1
It is crucial to understand whether or not operations in our system are idempotent. The more idempotent operations we have, the more resilient the system we can design.	263	10.1.3
It's often possible to tweak the consistency versus the availability of systems we use. So it's crucial to understand the consequences of those decisions.	291	11.3.1
Designing the versioning strategy for a network API from the start and documenting it publicly and clearly can give customers confidence and help them make their own versioning decisions.	331	12.3.2
Sometimes it's wiser to develop a do-it-yourself (DIY) solution with only needed functionality than using a heavy library that provides a required functionality but also a lot of other functions that we don't need.	362	13.2.1