SOFTWARE ENGINEERING

Software Mistakes and Tradeoffs

Lelek • Skeet

Every step in a software project involves making tradeoffs. When you're balancing speed, security, cost, delivery time, features, and more, reasonable design choices may prove problematic in production. The expert insights and relatable war stories in this book will help you make good choices as you design and build applications.

Software Mistakes and Tradeoffs explores real-world scenarios where the wrong tradeoff decisions were made and illuminates what could have been done differently. In it, authors Tomasz Lelek and Jon Skeet share wisdom based on decades of software engineering experience, including some delightfully instructive mistakes. You'll appreciate the specific tips and practical techniques that accompany each example, along with evergreen patterns that will change the way you approach your next projects.

What's Inside

- How to reason about your software systematically
- How to pick tools, libraries, and frameworks
- How tight and loose coupling affect team coordination
- Requirements that are precise, easy to implement, and easy to test

For mid- and senior-level developers and architects who make decisions about software design and implementation.

Tomasz Lelek works daily with a wide range of production services, architectures, and JVM languages. A Google engineer and author of *C# in Depth*, **Jon Skeet** is famous for his many practical contributions to Stack Overflow.

Register this print book to get free access to all ebook formats.
Visit https://www.manning.com/freebook

MANNING

"Great book that I wish I had earlier in my career. Many hard-learned lessons contained in these pages."
—Dave Corun, Avanade

"Clear and to-the-point summation of years of real-life experience in software engineering. A must-read for all newcomers to the software engineering world."
—Rafael Avila Martinez
Mastercard

"Shines a light on the intrinsic conflicts of the programming process and how they impact the code you write."
—Roberto Casadei
Università di Bologna

"Summarizes the main pain points for every software developer and presents solutions in a clear and didactic way."
—Nelson González, General Electric

Free eBook
See first page

ISBN-13: 978-1-61729-920-9

90000

9 781617 299209

Lelek
Skeet

Software Mistakes and Tradeoffs

MANNING

Software Mistakes and Tradeoffs

How to make good programming decisions

Tomasz Lelek
Jon Skeet

MANNING

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

Core concepts

Top warnings in this book

Top warning	Page number	Section
Using inheritance may sometimes introduce tight coupling that limits the flexibility and possibility to evolve our system.	38	2.5.2
The exception types thrown by your API are part of your contract. Beware leaking them.	55	3.4
Designing highly extensible code often increases the overall complexity of our solution.	94	4.5
Using an API (stream) without a complete understanding of its methods may lead to substantial performance degradation of our processing.	98	5.1.1
The lack of downstream components encapsulation allows us to deliver quicker at the beginning but limits the possibilities of evolution in the long term.	139	6.3.1
The idea that you can just store everything as UTC, and you'll never have time zone issues is a common myth. Storing UTC is fine in many cases, particularly timestamps, but loses important data in other cases. Don’t accept it as a silver bullet.	197	7.4.4
Our data partitioning may impact the possible ways of how we use the data. In the most serious cases, it may make some big data processing logic impossible to implement.	215	8.3.2
Some of the unchanged defaults of the library that you use can critically impact your application.	232	9.1
Some tools that work correctly in a one-node context may break its correctness in the multi-node environment.	270	10.3.1
Even if one of the systems in the pipeline offers effectively exactly-once, if our processing involves *N* remote system calls, those guarantees will not be held.	300	11.4.3
Breaking changes in a library incur cost throughout a community, even when properly versioned using Semantic Versioning. Consider the impact on consumers of the library—both direct and indirect.	319	12.2.2
Not all functional patterns should be used in a language that provides functional programming patterns, but is not a functional language from the ground up.	369	13.3.1