# Principles of Software Architecture Modernization

*Delivering engineering excellence with the art of fixing microservices, monoliths, and distributed monoliths*

*The Author:*

**Diego Pacheco**

*Co-author:*

**Sam Sgro**

www.bpbonline.com

First published: 2024

Published by BPB Online
WeWork
119 Marylebone Road
London NW1 5PU

**UK | UAE | INDIA | SINGAPORE**

ISBN 978-93-55519-535

www.bpbonline.com

# About the Authors

- The Author: **Diego Pacheco** is a seasoned, experienced Brazilian software architect, author, speaker, technology mentor, and DevOps practitioner with more than 20+ years of solid experience. I've been building teams and mentoring people for more than a decade, teaching soft skills and technology daily. Selling projects, hiring, building solutions, running coding dojos, long retrospectives, weekly 1:1s, design sessions, code reviews and my favorite debate club: architects community of practices and development groups for more than a decade. Live, breathe and practice real agile since 2005, coaching teams help many companies to discover better ways to work using Lean, Agile principles and methods like XP and DTA. I've led complex architecture teams and engineering teams at scale guided by SOA principles, using a variety of open-source languages like Java, Scala, Rust, Go, Python, Groovy, JavaScript and TypeScript, cloud providers like AWS Cloud and Google GCP, amazing solutions like Akka, ActiveMQ, Netty, Tomcat and Gatling, NoSQL databases like Cassandra, Redis, Elasticache Redis, Elasticsearch, Opensearch, RabbitMQ, libraries like Spring, Hibernate, and Spring Boot and also the NetflixOSS Stack: Simian Army, RxJava, Karyon, Dynomite, Eureka, and Ribbon. I've implemented complex security solutions at scale using AWS KMS, S3, Containers (ECS and EKs), Terraform and Jenkins. Over a decade of experience as a consultant, coding, designing, training people at big customers in Brazil, London, Barcelona, India, and the USA(Silicon Valley and Midwest). I have a passion for functional programming and distributed systems, NoSQL Databases, an obsession for Observability, and always learning new programming languages.

  Currently working as a principal Software Architect with AWS public cloud, Kubernetes/ EKS, performing complex cloud migrations, library migrations,server and persistence migrations, security at scale with multi-level envelope encryption solutions using KMS and S3. While still hiring, teaching, mentoring and growing engineers and architects. During my free time, I love playing with my daughter, playing guitar, gaming, coding pocs and blogging.

- Co-author: **Sam Sgro** is an experienced technologist, architect, and engineering leader with decades of hands on experience. Sam is a strong believer in how combining logic and reason with the right principles can bring about a better world, and his career has been about applying architecture and engineering in the best possible way, solving complex business and technology problems to help people.

  Sam's early background was a mix of both molecular biology and computer science, working in projects spanning open source cryptography and high performance computing. Slowly transitioning from having fun with Solaris and Linux to Java software engineering, Sam joined an early-stage bioinformatics and data analytics startup with a successful exit to Thomson Reuters. Since then, Sam has served as an engineering and architecture leader for teams across the US, Canada, UK, Spain, India, Ukraine, and Brazil, delivering multimillion-dollar growth and transformation initiatives across many industries, including pharmaceutical research and academic literature analysis.

  Sam's driving passions are solving problems, delivering software and helping people find their place in the world and hone their true capabilities. Sam loves delivering interesting and innovative tech solutions, such as in the early days using the Netflix stack, Cassandra and ElasticSearch, finding ways to creatively migrate technology to AWS, or building voice-based mobile applications to connect the world's knowledge. Sam loves running, hiking and spending time with his family and kids, fitting in video games when time permits.

  Sam currently serves as the architecture and consumer engineering lead of a FinTech company based in the Bay Area.

# About the Reviewer

**Garen Mnatsakanov** is a Director of Engineering at a FinTech company in the Bay Area. He holds two degrees B.S. in Computer Information Systems and Business Administration. Garen has worked in most areas of technology organization: development, testing, DevOps infrastructure, product, and design. He still stays hands-on and codes. Garen has experience building strong product engineering teams and a passion for fighting complexity, classical monoliths, and distributed monoliths at scale in the cloud.

*To Diego and Sam,*

I want to thank you for the opportunity to be the technical editor of this book. You guys are true software architects at heart, talented engineers, leaders, and mentors. The topics, analyses, case studies, references, and approaches you present in this book are invaluable. This book is also about building a strong engineering culture, enabling businesses to grow, succeed, and scale with technology instead of being burdened by it.

To my mom Larisa, my dad Victor, my sister Elina, and my awesome nephew Arsen thank you for your support, sacrifices, and love. And to my beautiful wife Andrea, who has been patient, supportive, and encouraging from day one, thank you, and I love you. I am sure you will enjoy this book as I have.

***— Garen, Technical Reviewer***

# Acknowledgements

- The Author: **Diego Pacheco**
  Thanks, God, Thanks, God, Thanks, God. I appreciate all my blessings, I wrote this book with love, passion, and lots of hard work. I wish we could share the same passion for software architecture, design, and complex problems. Deeply root that you can make a big impact in your organization and grow in your career and as a human being. Thank you for buying my book, I really appreciate it. I hope my experience and perspectives guide you in your journey. No matter if you are a software architect, software engineer, engineering manager, DevOps engineer, QA engineer, frontend engineer, director, VP, or CTO.

  I have a deep passion for technology, especially for software architecture. My passion could only happen due to the immense support of my loved family, my wife Andressa, and my dear daughter Clara. My dear friends Margarida, Adao, Israel and Tais, Jun, Adrian, Ty, and many other friends are not named here, but be sure you have a place in my heart… Brazil!

  *A small disclaimer: This book does not reflect the ideas, decisions, or opinions of any of my past or future employers or customers in my last 20+ years of experience with distributed systems and systems at scale, working for companies, and doing consultancy.*

- Co-author: **Sam Sgro**
  The ideas in this book were formed from over a decade of practical experience doing software architecture at scale with different teams, companies, and technologies. From the earliest days of a collaboration session in London's Green Park, Diego and I have grown the seed for many of the ideas of this book; we are delighted to share them with you. May they guide you towards doing work you are passionate and thrilled to do.

  To my friends and family across Canada, the US, Spain, the UK, and Brazil, and especially my beloved wife Claudia and children Kat and Erica, thank you for your patience and for giving me the space to do all the work needed to see these ideas hit print.

# Preface

**Why did we write this book?**

Software architecture is an amazing discipline, with many styles and forms coming in and out of fashion over time. Some tend towards centralization, like Blackboard and Monolith; others focus on distribution, like Event-Driven, Service Oriented Architecture (SOA, which is dear and warm to our heart), Microservices, Peer-2-Peer, REST, or Remote Procedure Call (RPC). Some styles are good for decisioning systems, like Rule-Based; others are great for concurrency and parallelism, like Share-Nothing and Actor, or focus on layering, like Client-Server, N Tier, or Component-Based. Some styles are good for data and long-running background tasks like SEDA and Streaming.

Architecture is cool, exciting, motivating, and warms our hearts, but there are traps too: bad practices, dark anti-patterns, and monoliths that live in the heart of that darkness.

How often did you hear complaints from the business or another engineer that a system is terrible, hard to maintain, and holding us back? How often have you heard from the business that the tech organization is slow and doesn't deliver? How often have you heard from engineers that they are drowning in technical debt and not as productive as they could be?

The answer is architecture; bad architecture created these problems, and good architecture will fix them. It's possible to have better systems, systems that can be maintained, sane, scalable, following proper design and architecture principles, and deliver what the business wants and what the engineers want. You can have your cake and eat it too. The work is not impossible, but we will not lie: this is difficult, complex, a never-ending battle that requires Homeric amounts of discipline and attention to detail. You can’t do such a task without care and passion: for yourself, your coworkers, your company, and the work that you do.

We wrote this book to share our passion and perspective about the common problems all companies face across all industries. Problems include technical debt, lack of correct principles, distributed monoliths, internal shared libraries, code and data migrations, and other essential concerns. Such problems are not new; they have been around for a long time and will likely be around as long as humans or AIs are writing software.

Our book will not give you easy answers or a magic formula for success; we are here to make you think, perform tradeoff analysis, and make the best-informed decision possible. If you want a magic wand to fix your problems, this book is not for you.

We will be very visual in this book, so expect a lot of diagrams to help convey our points. Our book is Java-centric but not code-heavy. You will see some pseudocode examples but do not expect complete applications built end to end. This is not a tutorial book.

Our goal with this book is to make sure you can fully understand the problems around monoliths and how you can approach them correctly and effectively. We will help you stop the bleeding, make sense of your reality, and have a path forward to better days and better systems using solid architectural principles and a bit of creativity.

We will be very technical in this book and will connect many different subjects, but don't worry. We will explain things in depth and with lots of practical scenarios and examples. The topics we will cover are wide-ranging, and sometimes we will review the same points from different angles to uncover different perspectives. We hope you like it, and thank you in advance for your readership.

While reading this book, you can expect:

- **Examples:** Practical examples from our experience in technology.
- **Tradeoff Analyses:** Architecture is all about tradeoffs, so expect many comparisons of pros and cons.
- **Figures:** Many diagrams and pictures to illustrate scenarios, tradeoffs, and options.
- **Multiple Options:** We will provide multiple options and the best analysis to make you consider the entire problem space.
- **Repetition:** We will repeat some principles over and over, analyzing them in different contexts to find new understanding.
- **Summary and Learning:** Every chapter will have a summary of things to remember from each chapter. This is a long book, and you might need to read it multiple times; make notes in whatever way works for you, and look back on what you find interesting or disagree with. (The authors like productive disagreements!) Remember that when you just passively read (input) you don’t learn as much as when you produce (output). You can write a blog post, run a lightning talk or presentation to your engineers or company, or talk to a friend. It is important to produce output, and we believe that’s the best way to learn anything, not only this book.

However, this book will not have:

- **Easy Answers:** There is no magic formula to fix your monoliths, just options we will help you navigate and digest to find your own answers. No quick fixes to your complex problems, just reality.
- **Tutorials:** You won't find step-by-step instructions on how to build applications. This is not a tutorial book.

## Code Bundle and Coloured Images

Please follow the link to download the
*Code Bundle* and the *Coloured Images* of the book:

**https://rebrand.ly/x39a8ep**

The code bundle for the book is also hosted on GitHub at
**https://github.com/bpbpublications/Principles-of-Software-Architecture-Modernization**.
In case there's an update to the code, it will be updated on the existing GitHub repository.

We have code bundles from our rich catalogue of books and videos available at **https://github.com/bpbpublications**. Check them out!

## Errata

We take immense pride in our work at BPB Publications and follow best practices to ensure the accuracy of our content to provide with an indulging reading experience to our subscribers. Our readers are our mirrors, and we use their inputs to reflect and improve upon human errors, if any, that may have occurred during the publishing processes involved. To let us maintain the quality and help us reach out to any readers who might be having difficulties due to any unforeseen errors, please write to us at :

**errata@bpbonline.com**

Your support, suggestions and feedbacks are highly appreciated by the BPB Publications' Family.

# Table of Contents

# Chapter 1 What's Wrong with Monoliths?

*Awareness is the great agent for change.*

— Eckhart Tolle

Monoliths are the great boogeymen of modern-day software engineering. Developers will gather around the (virtual) campfire, trading horror stories about that terrible Windows Desktop app that had a week of downtime because the JVM crashed every 24 hours or the 1990s banking application that required a team of 40 people to release. We've all had a coworker or friend commiserate while they are stuck in a big Monolith, wishing there was a way out.

Seeking better solutions, people embraced microservices.

**Result:** An 8-9 figure software development project, 100s of people, half the features scrapped, and now they have TWO terrible platforms to maintain, not just one. The cure can be worse than the disease.

Let us be the ones to tell you that there is hope. Real hope. Not one found in pursuing the latest and greatest software development trends, but real tactics, real progress that can be made to improve the software architecture and lives of developers, teams, and businesses worldwide.

But it all starts here. Understanding Monoliths is the key to being able to truly address some of their inherent flaws. After all, not everything is entirely bad or good. By understanding the problems and benefits of Monolithic architecture, you will avoid common pitfalls and provide real solutions for business and engineering.

## Structure

In this chapter, we will cover the following topics:

- What are monoliths?
    - Big codebase
    - Few deployment units
    - Other monolith smells
        - Centralized
        - Old
    - Sidebar: Does size matter?
- Issues with monoliths
- Patterns in software engineering
    - What is an anti-pattern?
- Living with Monoliths: anti-patterns, side effects, and amplifiers
    - Monolith anti-patterns
        - High coupling
        - Wrong abstractions
        - Lack of isolation
    - Monolith side effects
        - Difficult deployments
        - Insufficient testing
        - Lack of ownership
        - Slow adoption of new technologies
        - Slow development cycle
    - Sidebar: Automation and anti-patterns
    - Amplifiers
        - Broken windows/copy and paste
        - Microservice envy
        - Lack of Talent Density
        - Fear of change
- Are all monoliths bad?
    - Monoliths are a form of software architecture

- Monolith benefits
    - Refactoring and change impact
    - Simplify some migrations
    - A good starting point
    - Simplified infrastructure
- Types of monoliths
    - Classical, distributed, and modular monoliths
    - Modular monoliths: the good kind of monolith
    - Modular monoliths: Istio
    - Modular monoliths: mobile SuperApps
- Can bad monoliths be avoided?
- Things to remember

## What are monoliths?

First, we need to understand monoliths deeply to fight them. Otherwise, what are we fighting against? How do we know when we win? How can you define success? Monoliths are a very common theme in the software industry. Everybody has their definition of what a monolith is and many preconceptions. However, we need to start with a common understanding. Therefore, we need to define what a monolith is. So, what is a monolith? What comes to your mind when you think of a monolith?

To answer this question, refer to the following *Figure 1.1*, which consists of monoliths:

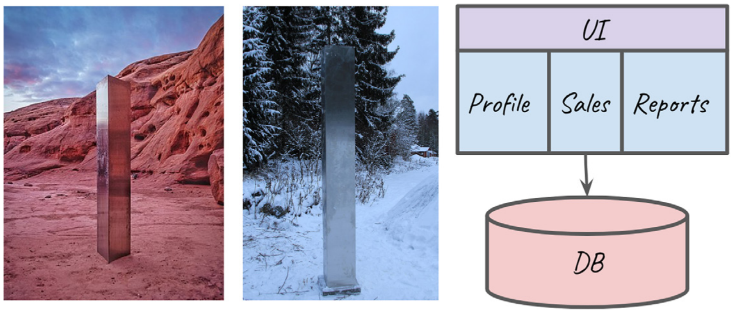

***Figure 1.1**: Monoliths*
*Source: https://en.wikipedia.org/wiki/List_of_works_similar_to_the_2020_Utah_monolith*

What do all these images have in common? Based on the preceding figure, we see big blocks that are hard to move and not easy to understand. We get the same traits when it comes to code.

Monoliths are a style of software architecture where a large amount of code is deployed using a small number of deployment units like exes, JARs, WARs, etc. In short, a big codebase with a few deployed artifacts. It is the natural state of most application development when it just gets started, having all the code in the same code repository and deployment unit simplifies your life in the early days where finding product-market fit trumps premature optimization.

## Big codebase

Monoliths typically have a lot of code. It could be in one source code repository or multiple repos for some types of monoliths which we will discuss later. But, there is ALWAYS a lot of code involved. Of course, what is big is relative. Big codebases are not a problem per se; it is more like a smell, but it happens often enough that it is a key characteristic of a monolith.

Sometimes monoliths are big in other ways. Monolithic applications can be quite complex with tons of features and often need big teams. Actively managing a monolith takes a lot of resources (and brave hearts!). Monoliths may or may not have many users or need lots of computing to accomplish their goals, but they have big, complex codebases.

## Few deployment units

When we build and release software solutions, we often have a system to do so. We use code versioning tools like git, for instance. We have issue trackers to track bugs, we capture requirements and needs in JIRA tickets, and we have ways to organize people to transform needs into solutions, often via Agile and Lean methods like Scrum, XP, Kanban, and many others. Teams use these and many other systems to divide their work into small, understandable chunks.

However, by its nature, a monolith shapes all of those diverse processes because of one simple fact: monoliths tend to have few deployment units[1], to the point of being just a single massive binary. Think about a massive .exe for a Windows desktop application or a JAR or WAR in Java.[2]

Having one or a few big artifacts similarly drives a huge process to get that deployment unit ready and tested. This reality applies regardless of whether the team is shipping a change in a single line of code or a massive feature; it still triggers the same army of developers and testers because it's all in one (or very few) packaged artifacts containing the entire application.

1 Deployment Unit represents how the application is deployed in production, usually in a server. Java for instance, could be an EAR, WAR or JAR file. For a windows application could be a .EXE file, for a Linux C++ lib could a .so file.

2 While we say binary here, as software often produces compiled code artifacts, but for some languages, it can also be a plain text code, interpreted at runtime. We will often refer to binaries in this book as frequently those artifacts are compressed, or otherwise live the bulk of their deployment lifecycle as a binary, but keep in mind the realities of diverse languages which our book applies to.

Here, you can see a great example of how software architecture shapes organizations. Structure and design influence the way in which a thing can be accomplished and who needs to do it. As an architect, you may think your role is limited to software, code, and how it is structured and deployed; in truth, your systems shape organizations and lie at the heart of an organization's ability to scale. Organizations will take the safe path, and if your architecture means the safe path to releasing code requires all teams to work night and day to release, debug, and then patch your monolith in that order… your organization will do that. This is not fun at all. Architecture is the great enabler or disabler, and a simple thing like "my app is deployed with 15 teams updating a single WAR in production" can lead to dysfunction and misery… misery only you can fix.[3]

## Other monolith smells

Monoliths are typically big and produce few deployment units, but there are other frequently found characteristics seen in monoliths. They are not a guarantee of a monolith, but they are so often found in monoliths that they are closely associated.

If you come across these, chances are you are dealing with a monolith. We call these sorts of items "smells". The term code smell[4] was coined by *Kent Beck* in the late 90s; a code smell is something that might lead to a deeper problem. In the context of monoliths, smells are more tricky, they may indicate a problem or a symptom, not a root cause, and the problem might be something else.

### Centralized

Monoliths favor centralization, with all features, configurations and tests all in one place. A simple way to think of it is that all our code will be running on one machine, like a desktop application. All our code is deployed together and just uses one machine to deliver all capabilities your business needs. All features, capabilities, and GUIs will share the same resources: CPU, memory, disk, and network.

The opposite of centralization is distribution, where everything is split apart. Imagine a microservices architecture where your software is running on multiple machines, accessing resources like databases or configurations that also reside on other devices.

Is centralization inherently bad? If it were, we would have an easy solution: divide the codebase. Let us introduce microservices and slice and dice our monolith into many pieces. The monolith would be gone, and our problem would be solved. Using the Microservices style of architecture (MSA), we move away from centralization and embrace distribution. We moved the needle from one extreme to another. But did this approach actually fix all

3 Using the tools in this book!
4 The concept of code smells was popularized by the Refactoring book, written by Martin Fowler in 1999. Kent Beck wrote an essay in Chapter 3 about code smells. https://wiki.c2.com/?CodeSmell