Clean Architecture: A Craftsman's Guide to Software Structure and Design

Robert C. Martin

A Literature Review

[SE1 - 2020]

Authors:

Roman Shevchuk

Kiryl Volkau

Illia Manzhela

Mohd Wafa

M Hasibur Rahman

Instructor: Maciej Bednarz

Content

1 Summary

1.1 Introduction

2 Review

2.1 Targeted audience

2.2 Overview

3 Critical Discussions

4 Personal Findings

5 Further Reading

1 Summary

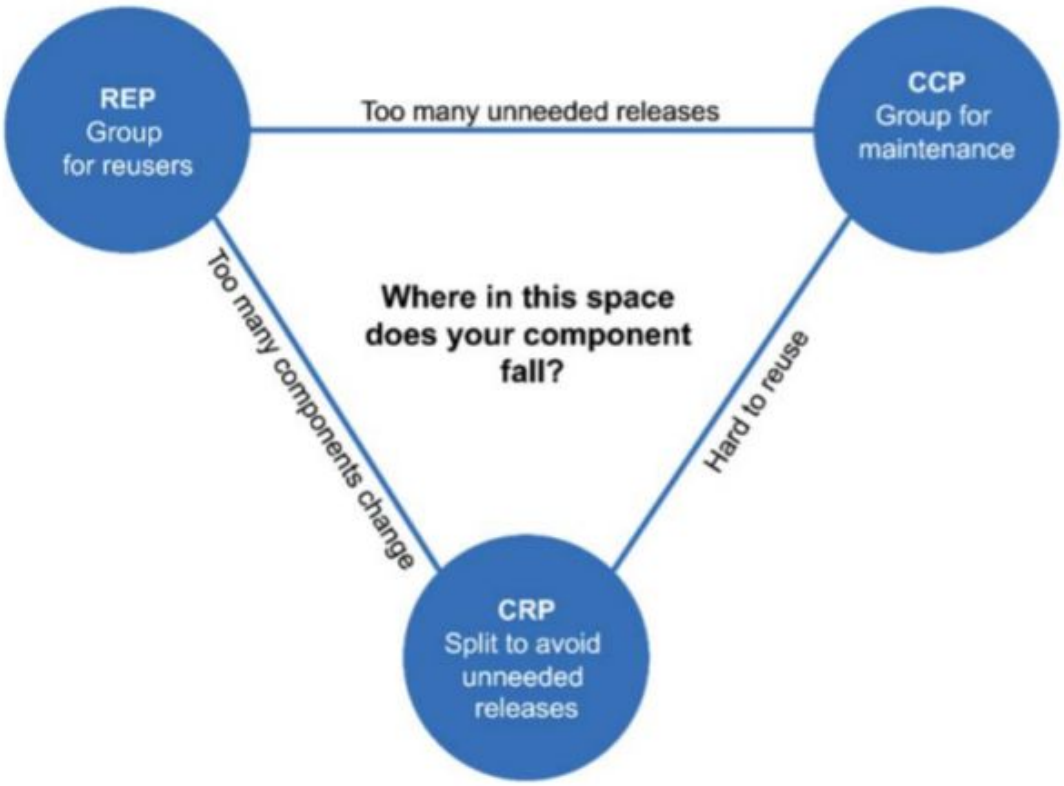
1.1 Introduction

The book essentially discusses crucial parts of software architecture and tools for determining what architectures are good and bad to maintain, deploy and develop. It provides core disciplines and etiquettes of system designs.

It starts from reasoning why it is always better not to be in a hurry at the early stages of development and pay significant attention to clean code, which personally impressed me considering that a lot of young programmers are a little sinful when it comes to clean code and have not discovered yet that such bad habits could lead to millions of dollars in losses. Thereupon, the book gradually narrates the history of programming and how it all started, exhibiting the main paradigms of programming. After that, the author starts discussing the smallest building blocks of today's programming - classes and their relations. Once classes and its abstract properties are discussed, he explores to further advanced structures - components and introduces some properties of components like stability and abstraction of components, those allow the tool to use simple and yet efficient techniques to determine whether those components are interconnected in a good manner and if it is an obstacle in the future.

Throughout the book, authors describe how certain choices are to be made and referred to them as - architectural detail. Author argues that things like database, GUI, or framework are details that do not matter much regarding software architecture. Choosing a particular database or framework should be considered as a second thought. This habit of pre-choosing a framework or database does have an affect on business logic of a project. Such other useful advice is further discussed in the book and will be touched on throughout this report.

In the last part of the book, the author defines what architecture is and what defines a good architecture. He stresses that a well-designed system is easy to develop, deploy and maintain. He introduces the three principles of component cohesion: The Reuse/Release Equivalence Principle, The Common Closure Principle, and The Common Reuse Principle. And he shows why it is better to alway remember about them. After the author shows the tension diagram for component cohesion which shows how the three principles of cohesion interact with each other. He underlines that an architect should never focus on some aspects of the diagram but alway take into account all of them and find the golden ratio for his project.



After that, the author focuses more on how architecture design works in the real world. He introduces business rules without which, claims the author, a software system exists. He explains what Clean Architecture is.

As students, we found this book helpful. It showed us that we even had some stereotypes about working in this field, and the book gave us a realistic glimpse of the insides of the industry. We would definitely recommend this book to young people in this beautiful and ever growing world of technology.

2 Review

2.1 Targeted audience and how well it is written

The main types of audience that the book is focused for are mostly as follows :

1. If you are interested in building tenable software.
2. You need arguments to change the mindset of your stakeholders to take some more time for creating better quality software than you used to.
3. You want some positive hope on building applications in a different way than the basic/default “3-layered architecture”.

2.2 Overview

Clean Architecture - A Craftsman’s Guide to Software Structure and Design by Robert C. Martin takes a step back from the details of programming and discusses the bigger picture of software Structure and Design.

The book begins with three myths we as software developers live in:

1. Myth 1: We just need to get the product first and we clean it up later.

2. Myth 2: Messy code makes us faster today and slows us later. Myth: We can switch mode from making messes to cleaning messes. (making messes is always slower, even in very short term as shown in example of a simple kata in the book)

3. Myth 3: Starting from scratch is a solution.

In my opinion, the book starts and ends very weak and was by far the low point in rating this book, but it was all brought back to a good level with a strong and meaningful middle part.

We have divided the book into 7 parts for better explanation of the review of the book :

Part 1 of the book gives a header of the topic by explaining architecture as the way to facilitate the development, deployment, and maintenance of a software and not as the means to make the software work by explaining code. Thus, good architecture will minimize the lifetime cost of a software.

Part 2 through Part 4 covers the basic three programming paradigms (structured programming, functional programming, and object-oriented programming) and a look at the SOLID principles that Martin himself has introduced in one of his papers back in the day. Also, Martin discusses some Component Principles around cohesion and coupling.

The 5th Part is titled “Architecture” and brings the most value from the book. The main topics of this part are boundaries and dependencies. Most of the chapters in this part discuss how to structure your code in order to cling to the SOLID principles and keep your software more flexible.

Part 6 discusses frameworks as small details to the overall main architecture and that it should be treated the same way by holding them at arm's length from your main domain code.

Finally, in the last part, Part 7, titled "Architecture Archaeology", Martin describes some of the software projects he's been working on the last 40-45 years and the lessons he learned from them. I found the pictures of room-filling computers with giant disk storage alongside a dry explanation of the software projects around those computers rather boring.

Especially software engineers that want to take a step toward more senior development roles will find real useful value in it for complex tasks at hand. For more experienced software engineers, most of the content will not be new, but it will probably bring some new ideas for current complex software projects.

3 Critical discussion

As for the missing/controversial parts of the book the first one is the lack of examples. Chapter 33 tells us about the e-commerce video sales application, however, the example feels too artificial. Although there is a set of really interesting projects from real life at the appendix - it would be great to have them included in the main part as part of the discourse.

Moreover, in real life the big part of the software is already written and it just needs to be maintained/improved. The book doesn't really say anything about improving already existing systems and it keeps silent on the matter of

refactoring and adding new functionality. The main postulate introduced by “Uncle Bob” (as it was to the authors of this report):

“The primary purpose of architecture is to support the life cycle of the system. Good architecture makes the system easy to understand, easy to develop, easy to maintain, and easy to deploy. The ultimate goal is to minimize the lifetime cost of the system and to maximize programmer productivity.”

Taking the following paragraph into consideration, the lack of information about the maintaining and improving existing system feels wrong.

Moving to the next rather acute problem of the book is the “under-emphasizing” of such factors as the size of the application (although there are parts including discussion of the trade-offs), time limits, project teams size, etc. Those factors are not really about architecture, but they play a significant role when developers choose the direction and pace for the implementation of architectural solutions.

One of the good questions asked on one of the forums was “should we always separate database access and business logic, especially when there is a limit of time?”. It brings up one of the most annoying problems of OOP - each structural problem could be solved by adding one more layer of abstraction. So, it would be nice to know when we can find a compromise between problems created by OOP and man hours that are available.

The next problem that was noted by the authors of this report (and many other people on the Internet) is the lack of organization of the book. In particular, the part about different coding paradigms feels a little bit off, since the book itself focuses more on the object-oriented approach. Nowadays, more and more general purpose languages as Go, Rust and Scala introduce

more functional approach (even C#, implicitly an OOP language, is adding more and more support for functional paradigm, in particular on the compiler optimisation layer and bringing some features from Rust in the upcoming .NET 6 [specifically - traits], and introducing immutable value types - records) and gain more and more popularity as they focus more on the problem solving (the great article “Object-Oriented Programming — The Trillion Dollar Disaster” by Ilya Suzdalnitski created a lot of controversy around itself, but also brought up the topic of how enterprise has fallen in love with OOP and how many problems have arisen from this) rather on abstractioning.

From this we can see how developers’ and software engineers’ preferences change over the years and that it is really important to stay up-to-date with the technologies and try to handle books like “Clean Architecture” with more and more caution, treating it as one of the many possible ways rather than the “this is the way” type of thing. Going back to the topic of the poor organisation - the book could be really shrunk to a 100-page size.

As for the summary of this part, we would like to point topics that we did not really like about the book:

- Lack of examples
- Under-emphasizing the environment in which architecture is developed
- Lack of the info for maintenance/improvement of existing architectures.
- Book structure and organisation
- The actuality of the methods described.

4 Personal Findings

The amazing thing about this book is that the author converts what you can find in other books as looking at "complex concepts" into really "easy concepts". Is amazing how the author simplifies day-to-day problems and displays examples to make everything clear. I was really impressed with the insight of this author.

The first half is really impressive and well worth the read of the book itself. Seeing that you can extrapolate Object-Oriented principles to the extent of architecture simplifies reasoning a lot and gives less room to overengineering.

However, what really bummed me down is the lack of references and the lack of comparisons to other authors' allegedly clean architectures. The realm of architecture seems much wider than what is described in the book and Robert Martin seemed to try to dodge that topic throughout the entire book.

Moreover, as students, we have found a lot of interesting things because before reading this book I did not really have any idea that programming is not only about coding but thinking about the architecture of the future project is the main part of success.

Reading this book, especially its first chapters we get the feeling that the author has met a lot of unprofessional programmers who did not want to write clean code and didn’t want to be responsible. The main message of the book is to be professional, always be interested in the topic of programming and always take responsibility for what you have coded.

5 Further reading

- “The design of everyday things" by Donald Norman. (https://www.amazon.com/Design-Everyday-Things-Revised-Expanded/dp/0465050654)
- “Cloud computing patterns” by Fehling, C., Leymann, F., Retter, R., Schupeck, W., Arbitter, P. (https://www.springer.com/gp/book/9783709115671)
- “Building Microservices: Designing Fine-Grained Systems” by Sam Newman (https://www.amazon.com/Building-Microservices-Designing-Fine-Grained-Systems/dp/1491950358/?tag=javamysqlanta-20)
- “Domain-Driven Design: Tackling Complexity in the Heart of Software” by Eric Evans (https://www.amazon.com/Domain-Driven-Design-Tackling-Complexity-Software/dp/0321125215/?tag=javamysqlanta-20)
- “Software Architecture in Practice (SEI Series in Software Engineering)” by Len Bass, Paul Clements, Rick Kazman (https://www.amazon.com/gp/product/0321815734/ref=as_li_tl?ie=UTF8&tag=ashanin-20&camp=1789&creative=9325&linkCode=as2&creativeASIN=0321815734&linkId=a35d633edf08483b3a23986c745d0510)