

Software Architecture and Techniques

Why Agile Architecture and Design?

Every Company is now a software company.

– *Forbes Magazine*

Software is eating the world, in all sectors.

In the future every company will become a software company.

– *Marc Andreessen, Wall Street Magazine*

The future is already here. It is just not evenly distributed.

– *William Gibson*

Form follows function.

– *Louis Sullivan*

SWAT Course Description

Teaches the **basics** of systematic **agile design** of an adequate software architecture for a selected application.

Methods of **agile quality assurance** and **software craftsmanship** are explored using predefined or self-selected projects provided by students.

Organization (1/4)

Lecturer	Marcel Baumann
Lectures	3 hours per week, and 14 weeks
Room/Time	<i>see technical university course site</i>
Description	Software Architecture and Techniques
Acronym	SWAT

Organization (2/4)

- SWAT weights 3 ECTS – *90 hours*
 - Lectures → $14 * 2.5$ **35 hours**
including 1/3 practice time
 - Examination preparation **15 hours**
 - Learning and exercises **40 hours**
- All documents are available on the lecture platform
- Questions and Answers in SWAT lecture forum (Microsoft Teams)

Organization (3/4)

- Slides are in English and provide hints on the theory
 - *Attend the lecture and read the literature, slides are not enough*
- Assumes you know Java 21 and higher, OOP, Scrum
- 2/3 theory and paper exercises and 1/3 computer exercises
(*source code is written in Java 21 or higher, actual regular JDK is preferred*)
- Literature
 - References articles *must be read*
 - Historical articles *could be read*
 - References books *could be read*

Organization (4/4)

- Lecture attestation
 - Refactoring project and presentation to your peers (history in git)
 - Architecture examples and participation in exercise coaching
 - Test automation (TDD, ATDD, CI/CD/CD) concepts and examples
 - Presentation of code during the practical part of the lecture
 - Proficiency with GitLab or GitHub
- Examination (after successfully completing lecture)
 - **Oral** examination with questions about the course slides, mandatory articles, and exercises

Student Portfolio

- Refactoring project and presentation (history in git) based on your source code
- Architecture examples based on your source code
 - with diagrams e.g. [PlantUML](#), [C4](#)
- Test automation (TDD, ATDD, CI/CD) concepts and examples based on your source code

Reference Books

Clean Architecture: A Craftsmanship Guide to Software Structure and Design

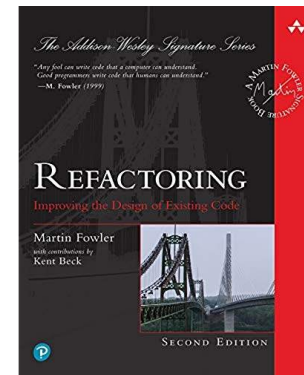
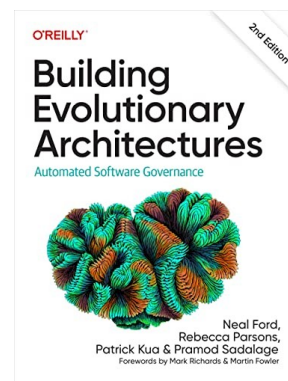
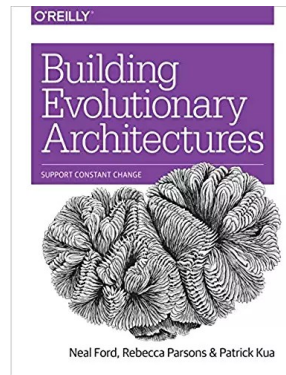
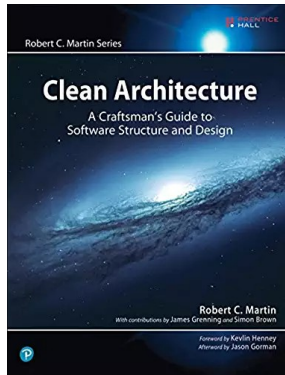
Robert Martin, Prentice Hall, 2018

Building Evolutionary Architectures

Rebecca Parsons, O'Reilly, 1st Edition 2017, 2nd Edition 2023

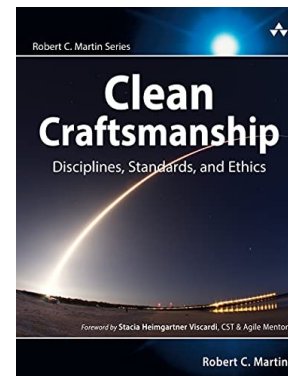
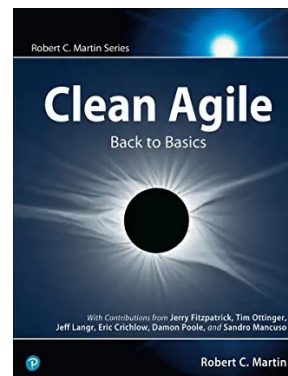
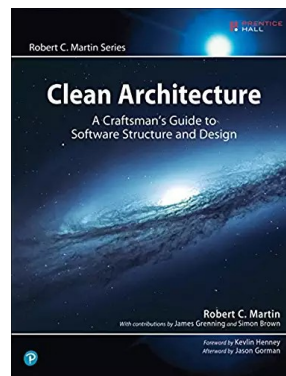
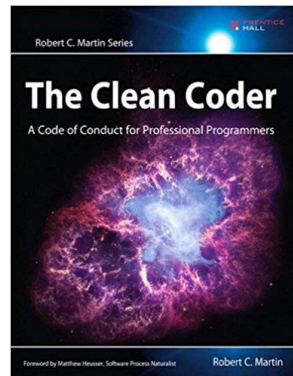
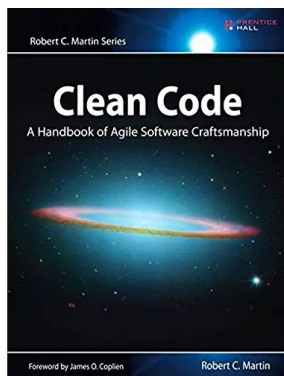
Refactoring: Improving the Design of Existing Code

Martin Fowler, Addison Wesley, 1999-2015



Historical Famous Books

- *Clean Code: A Handbook of Agile Software Craftsmanship*, 2008
- *Clean Coder: A Code of Conduct for Professional Programmers*, 2011
- *Clean Architecture: A Craftsmanship Guide to Software Structure and Design*, 2017
- *Clean Agile: Back to the Basics*, 2019
- *Clean Craftsmanship: Disciplines, Standards, and Ethics*, 2021



Goals

- Understand architecture as a **compromise** to fulfill functional and non-functional requirements
→ *design engineering*
- Have a **toolbox** to define an emergent and evolving architecture
- **Understand** the advantages and consequences of agile and lean approaches
- Be able to **work** on an agile product development initiative

Principles

- Science and its practical application “**engineering**” are vital tools in making effective progress in technical disciplines.
- Our discipline is fundamentally one of **learning** and **discovery**, so we need to become **experts at learning** to succeed, and **science and engineering** are how we learn most effectively.
- Finally, the systems that we build are often **complex** and are increasingly so. Meaning, to cope with their development, we need to become **experts at managing that complexity**.

Farley, David. Modern Software Engineering (p. xxiii)

Designers are Experts of Learning

- Iteration
 - Testability, Deployability, Speed, Variability Control, Continuous delivery
- Feedback and *Transparency*
- Experimentation and *Inspection*
- Empiricism and *Adaptation*

Designers are Experts of Complexity

- Modularity
- Cohesion
- Separation of Concerns
- Abstraction
- Loose Coupling

Lecture Content

- Why Agile Architecture and Design?
- Evolution of Software Architecture over the last Decades
- What is Agile Architecture?
- Agile Approaches with Scrum, XP, LeSS
- Refactoring
- Errors, Vulnerabilities, Smells in Source Code
- Architecture of Components and Subsystems
- Verify Functional Features
- Validate Quality Attributes of Software Architecture
- Architecture Documentation
- Architecture Trends I
- Architecture Trends II
- Domain-Driven Design Workshop
- Team and Technical Excellence for Architects

Participants Expectations

- What are your expectations?
- Which themes would you like to hear about?

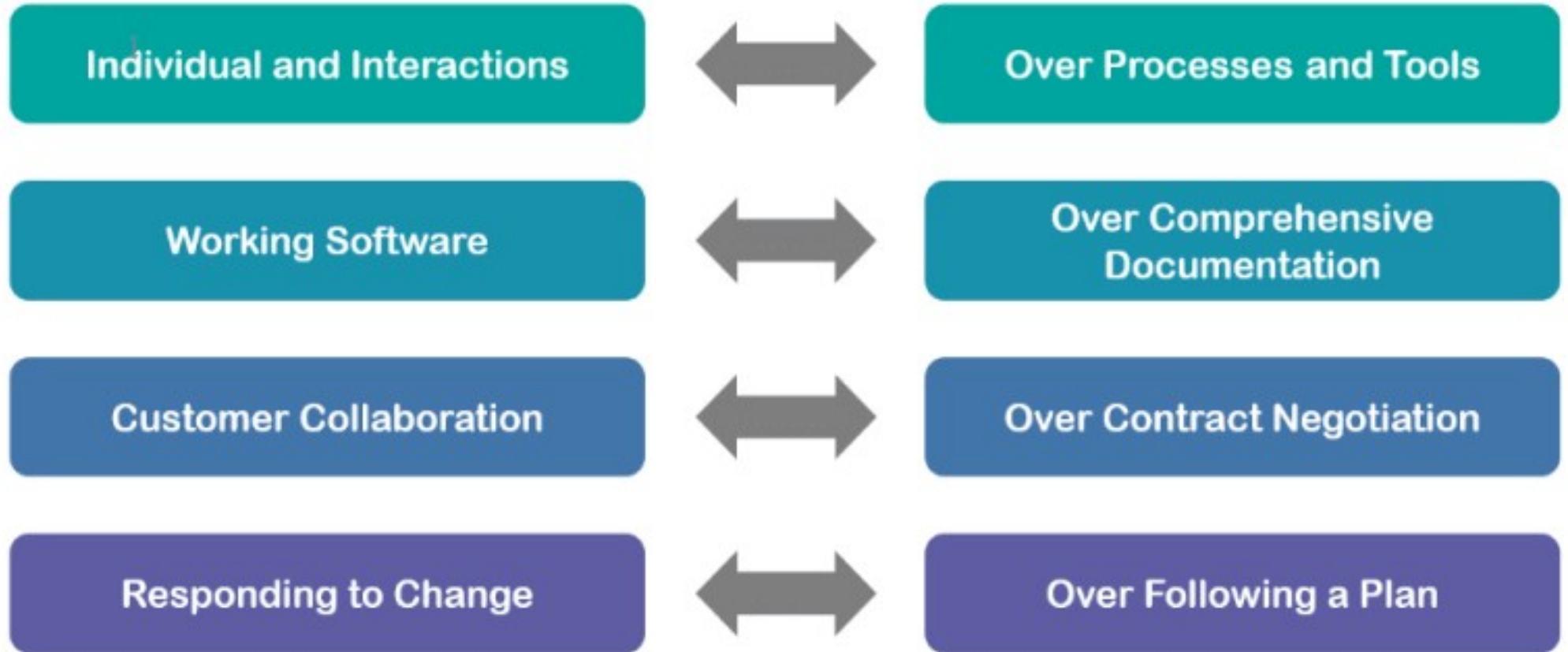
Why Agile Architecture and Design?

- Most of digital product development uses **agile** approaches
- As an architect, designer, developer you must:
 - Implement **functional** requirements
→ *Build the correct product*
 - Fulfill **non-functional** requirements
→ *Build the product correctly*
 - Respect **legal** and **governance** rules
→ *legal, social, and ethical responsibility*

Agile Architecture

- 1) The sum of all the **source code** is the true **design blueprint** or software architecture.
- 2) The real software architecture **evolves** (better or worse) every day of the product, as people do **programming**.
- 3) The real living architecture needs **to be grown every day through acts of programming** by **master** programmers.
- 4) A software architect who is not in touch with the evolving source code of the product is out of touch with reality.
- 5) **Every programmer is some kind of architect** - whether wanted or not. Every act of programming is some kind of architectural act - good or bad, small or large, intended or not.

Agile Manifesto



Agile Manifesto Principles

12 Agile Principles

@OlgaHeismann

Satisfy the customer through early and continuous delivery of valuable software.



Welcome changing requirements, even late in development.

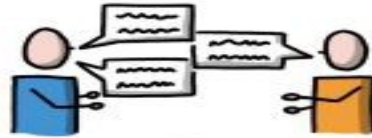


Deliver working software frequently.

Business people and developers must work together.



Build projects around motivated individuals. Give them the support they need. Trust them.



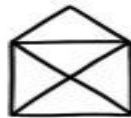
The most efficient and effective method of conveying information is face-to-face conversation.

Working software is the primary measure of progress.



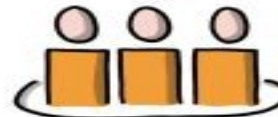
The sponsors, developers, and users should be able to maintain a constant pace indefinitely.

Continuous attention to technical excellence and good design.



Simplicity — the art of maximizing the amount of work not done — is essential.

The best architectures, requirements, and designs emerge from self-organizing teams.



The team reflects on how to become more effective and adjusts its behavior accordingly.



Some Agile Manifesto Principles

- Our highest priority is **to satisfy the customer** through early and continuous delivery of **valuable software**.
- Business people and developers must work together **daily** throughout the project.
- Build projects around **motivated individuals**. Give them the **environment** and support they need, and **trust them** to get the job done.
- **Working software** is the primary measure of progress.
- **Continuous** attention to **technical excellence** and **good design** enhances agility.
- **Simplicity** - the art of maximizing the amount of work not done - is essential.
- The **best** architectures, requirements, and designs **emerge** from **self-organizing teams**.

The source code is the architecture

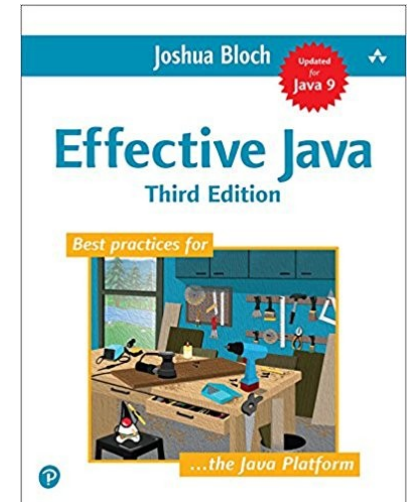
- Difference between to *architect* (process) and an *architecture* (result)
- If you think good architecture is *expensive*, try bad architecture
- ***Waste*** in architecture and design
- Good architects should be ***good developers***

Amazon Service Architecture

- 1) All teams will henceforth expose their data and functionality through **service interfaces**.
- 2) Teams **must** communicate with each other through these interfaces.
- 3) There will be **no other form of interprocess communication allowed**: no direct linking, no direct reads of another team's data store, no shared-memory model, no back-doors whatsoever. The only communication allowed is via service interface calls over the network.
- 4) **It doesn't matter what technology they use**. HTTP, CORBA, Pub/Sub, custom protocols -- doesn't matter. Bezos doesn't care.
- 5) All service interfaces, without exception, must be designed from the ground up to be externalizable. That is to say, the team must plan and design to be able **to expose the interface to developers in the outside world**. No exceptions.
- 6) Anyone who doesn't do this will be **fired**.
- 7) Thank you; have a nice day! (By **Jeff Bezos**, CEO Amazon, 2002)

Anti pattern: Cowboy Programmer

- *“From the brain to the terminal”*
- Spaghetti code, huge classes, huge methods
- No automatic tests
- No automatic build and delivery
- Has **never** read “Effective Java”



Anti pattern: Gold Plated Architecture

- No running code after weeks of work
 - *Architects are incapable or refuse to write code*
- Tons of UML diagrams
 - *You have a kickback from the UML tool or the printer company*
- Software Architecture Document *SAD* with hundreds of pages
 - *Can be often found in Swiss companies*
- Proof of concept on paper
 - *Value is zero!*
- All the patterns of the Gang of Four
 - *Wow, the architect can read*
- Copy Amazon or Netflix architecture for an internal product
 - *Incredible, you have 2'000'000'000 customers, congratulations*

Doing Agile instead of Being Agile

- Measuring velocity is odd because you should measure **outcome**
– *means value - instead of output*
- **Undone Department** - Can You *really* ship?
 - You build it, You deploy it, You run it, *and you document it!*
- **Doing** Scrum **damns** you to deliver mediocre software every two weeks
- Being Agile means Scrum, eXtreme Programming, Lean → TDD, ATDD, DevOps, Agile Architecture, Refactoring, etc.
 - Learning and improving daily
 - Simple Test: How effective is your retrospective?
 - Simple Test: Is your CI/CD always green?

Good Principles (1/2)

- KISS – *Keep It Simple Stupid*
- DRY – *Don't Repeat Yourself*
- YAGNI – *You Aren't Gonna Need It*
- Architecture is like gardening
 - Clean Architecture
 - Legacy solutions → Violation of clean approaches
 - Geriatric solutions → Time to leave

Good Principles (2/2)

- **SOLID** - Five Design Principles
 - **Single responsibility Principle**
 - Open/Close Principle
 - Liskov Substitution Principle
 - **Interface Segregation Principle**
 - **Dependency Inversion Principle**

Architecture Styles: Old (technical)

Various architecture styles exist. Here some examples

- Batch – [Java Batch Module JSR-352](#)
- Pipe and Filter – Streams in Java
- [Blackboard](#)
- [Client Server](#) – JEE servers
- [Layered Systems](#) (3-tier, N-tier, multi-tier architecture)

Architecture Styles: New (business)

- Micro-architecture
 - Bounded Domains – parallel development
 - Build on Docker and Kubernetes
- Hexagon and Onion Design
 - Business Domain Model
- Reactive and Event Based – JavaRX
 - Java Future, FutureCompletion, ...

Architecture Styles: Other (DevOps)

- Server Architecture
 - Barebone servers, virtual machines, docker images, Serverless
- Resilience
 - Redundancy, P2P, Serverless
- Security
 - Trusted, Secure and Untrusted Approaches

Architecture Concepts

- Modularity
- Cohesion
- Separation of Concerns
- Information Hiding and Abstraction
- Managing Coupling

Links

- Git Introduction - [Switch Tube HSLU Video](#) -
- [What Software Architecture Should Look Like?](#), David Farley, GOTO 2022
- Wikipedia [Agile Architecture](#)
- What should a [Scrum Developer](#) know?

Exercises (1/2)

- Read article “Who Needs Architects?” written by [Martin Fowler](#)
- Short [YouTube video](#) and long [YouTube video](#) by Martin Fowler
- Study [Agile Manifesto](#) and [12 Principles](#)
- Explore [LeSS architecture page](#)
- How are **you** doing design and architecture?
*(team discussion and identify **your improvements** based on Manifesto principles)*

Exercises (2/2)

- Install [IntelliJ](#) IDEA IDE and GitHub Copilot
- Use Analyze Code of IntelliJ and install [SonarLint](#) Plugin
- [PlantUML](#) – optionally plugin for IntelliJ IDEA
- [AsciiDoc](#) – optionally plugin for IntelliJ IDEA
- You could also use [SonarQube](#) cloud account
- You should also use official school Gitlab to host your project
- Import your code and do *Analyze Code* with *IDE* and *SonarLint*. *Your project must be under git (use university gitlab).*