Software Architectures and Patterns

Basics, Principles and Questions for processing them

8.2.2024

Juhani Välimäki

Software Architecture – one Definition

- *”The fundamental organization of a system embodied in its **components**, their **relationships** to each other, and to the **environment**, and the **principles** guiding its **design** and **evolution**.”*

(IEEE 1471:2000 standard)

Software Architecture – another Definition

- *”Architecture represents the **significant design decisions** that shape a system, where significant is measured **by cost of change**”*

  (Grady Booch, 2006)

Software Architecture – third Definition

- *“The set of design decisions that must be* ***made early****”*,
  (Martin Fowler)

Software Architecture – fourth Definition

- *“Architecture, in the field of software development are decisions that are **hard to reverse**”,*

  (Matthew Parker)

Characteristics of a good software architecture (as the heart of good information system <u>and</u> SW development)

- Scalable
- Reliable
- Efficient
- Secure
- Maintainable
- Extensible
- Testable
- Observable

What Software Architecture consists of?

- Structures
- Division to sub-systems
- Communication models (E.g. SPA fullstack may have REST api between Front and back, and e.g. SQL-connection with some driver between backend and database. Or a IoT system with totally asynchronous communication using message buffers/queues). ”Messaging patterns”
- See how the communication model above also includes the timing / execution model
- Choices on ready-made environments and technology stacks
- Design patterns in code level https://en.wikipedia.org/wiki/Software_design_pattern
- Agreed principles and practices
- Naming convention and coding conventions
- Nowadays also Continuous Integration (CI) affects the architecture. Think of e.g. cloud-native, container orchestration & configuration, microservices and serverless options
- Thus, architecture is related also (but not only) to the development and development time.

Examples of Software Design decisions

- ’more-synchronous’ API communication (e.g. REST or GraphQL?)
  OR: totally asynchronous communication with message systems like Kafka or MQTT
- Data storage solution (Can/should be hidden behind some kind of ’data access layer’ but it still affects e.g. business process rule handling)
- Container-based Kubernetes & Docker
  OR: Serverless solution with AWS Lambda
- What tech stack to use? (Not the choice itself, but it often affects some architectural decisions. Usually there are hinted / agreed / proven ways of doing standard solutions in each technological environment or framework. If you do not follow them, there might not be e.g. a largely proven security framework)

What types of models/diagrams exist? E.g. in UML

- **data** model, the case **business information** structure
- **processes** and **actions**, **states** and transitions
- **features** as use cases or user stories and **roles** using the features
- algorithm **programming logic**
- **communication** and **data flow**
- **software components** and packages
- **servers** and other **deployment** related plans.

These just on one sitting without looking at materials, so even more exist.

Do you need to model all these? Nowadays we only model what is necessary and/or beneficial.

Software Design patterns

- https://en.wikipedia.org/wiki/Software_design_pattern

You have already used several. Possibly e.g. facade, singleton, iterator, factory, strategy

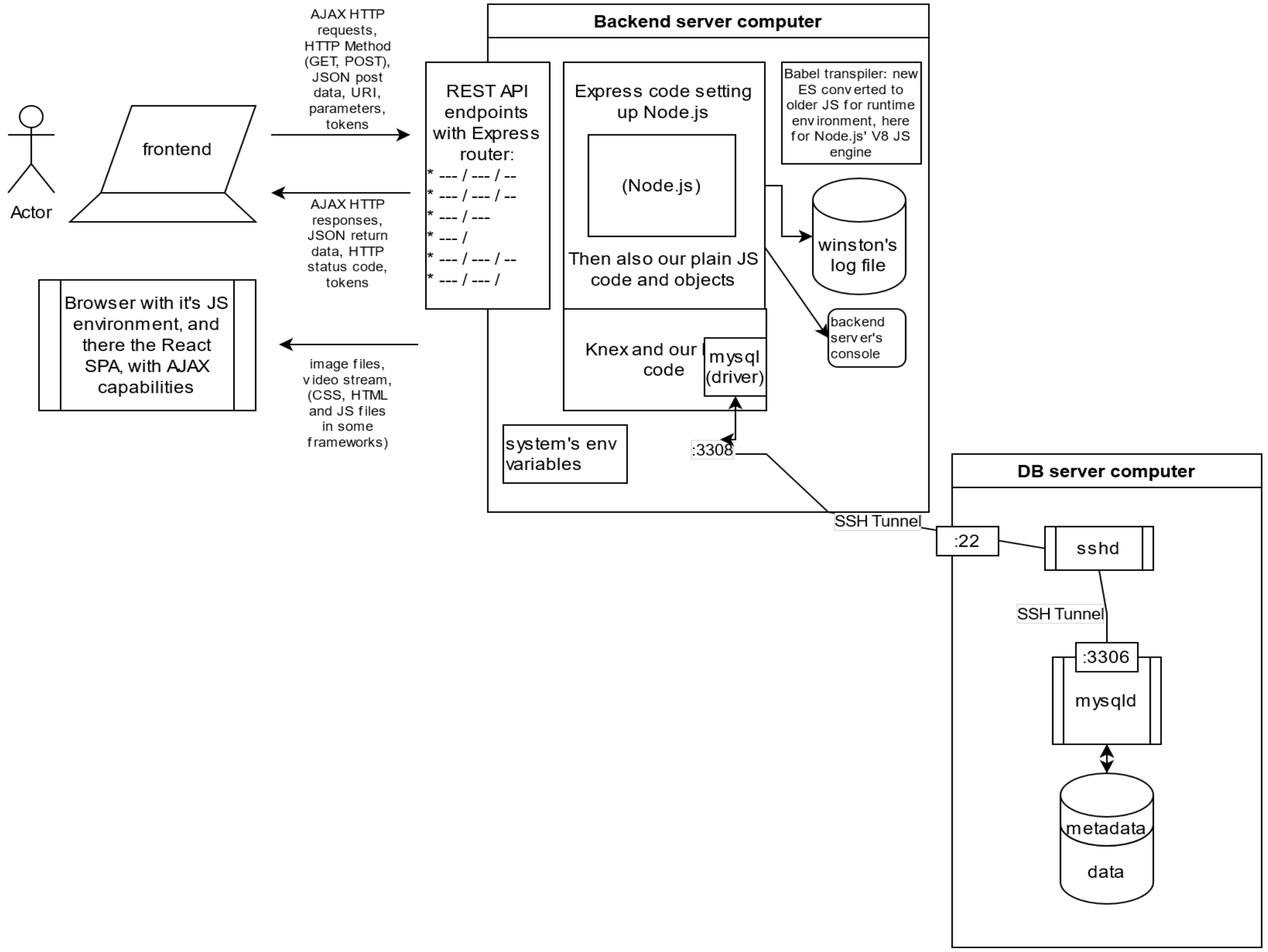
- The page above has link to books if interested. Including the 1994 ”Gang of Four” book, written by Gamma et al.
- E.g. when you have defined a Comparator type for your items, e.g. HouseComparatorBySize, HouseComparatorByYear, and give the comparator you want to use this time to the sort( ) function, you are using the **strategy** design pattern.

- Software design patterns are a Good way to challenge yourself and twist your brain if needed.

”Traditional full-stack architecture example”

Serves the purpose still for some information systems. E.g. some customer projects are still done this way, if the nature of the communication between the systems does not require something else.

Also this could be part of some bigger system that would have other kind of architectures. Possibly joined via Database or Backen API


Actor
frontend
Browser with it's JS environment, and there the React SPA, with AJAX capabilities
AJAX HTTP requests, HTTP Method (GET, POST), JSON post data, URI, parameters, tokens
AJAX HTTP responses, JSON return data, HTTP status code, tokens
image files, video stream, (CSS, HTML and JS files in some frameworks)
Backend server computer
REST API endpoints with Express router:
* --- / --- / --
* --- / --- / --
* --- / ---
* --- /
* --- / --- / --
* --- / --- /
Express code setting up Node.js
(Node.js)
Then also our plain JS code and objects
Babel transpiler: new ES converted to older JS for runtime environment, here for Node.js' V8 JS engine
winston's log file
backend server's console
Knex and our code
mysql (driver)
system's env variables
:3308
SSH Tunnel
DB server computer
:22
sshd
SSH Tunnel
:3306
mysqld
metadata
data

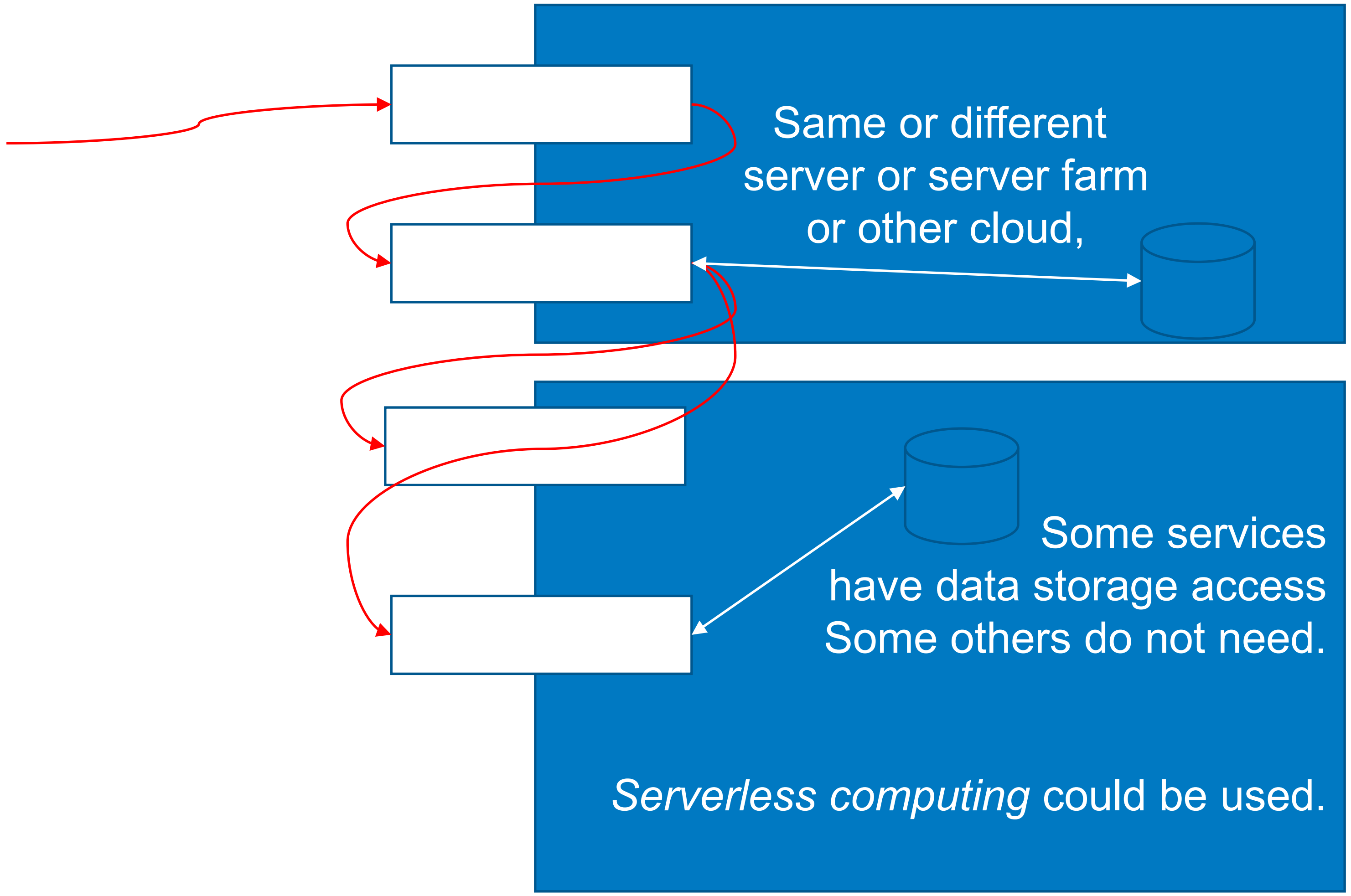
Gold vault example (Juhani) of microservices. Just fast (incomplete) example of SRP/SOC.

1. One service receives images from security camera(s) and relays them to the people recognition service as images.
2. People recognition service identifies (with ML based AI) which images contain human beings and relay those image sub-parts cropped to the face recognition service
3. The face recognition service will try to identify known people and unknown people. Known people recognitions will be indicated to the known people log service. Unknown people to some other service.
4. Known people log service will make markings to the database of where the known person (employee, guard) has been spotted and when.
5. Unknown people service will log the time and face image to the database and will send notification to the control room service
6. Control room service will sound an alarm and show the unknown person face image with timestamps on the screen.

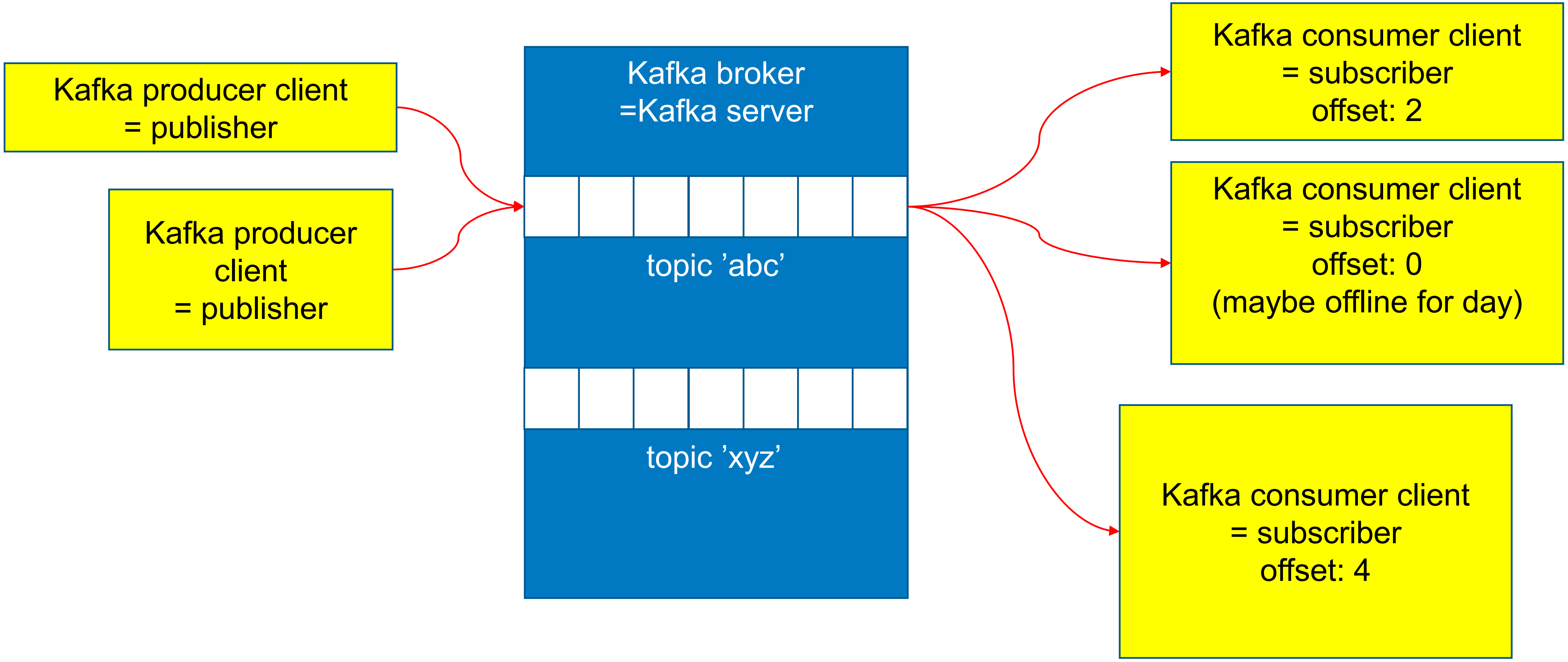
What benefits you can see in this? There are multiple. Possible to list 5+ for sure.

The microservice architecture as principle picture

Services that follow the SRP/SoC principles call other services.


Same or different server or server farm or other cloud,
Some services have data storage access Some others do not need.
Serverless computing could be used.

Distributed Event Stream architecture – e.g. Kafka



Haaga-Helia

Notice that real systems usually have mixed architecture

- The previously presented three architectural models are naturally just simplifications
- The real information system architecture you will be creating e.g. for customers in our projects might well be a **mix of even all of these**. And nowadays often is.
- But look at these as tools in your toolbox. Then you are free to use the one that fits the best in each need.
- It may still be also just the good old full-stack, though that starts to be more rare in our customer cases

Three phases in programming a case (at least in school cases)

1. Technical studies
   - Fast experiments of any kind. Speed in learning is important. But also deep understanding of at least the core concepts. Mickey mouse examples. Closer to 0% customer value, but aiming at closer to 100% technical ability development
   - Only think of the customer case from technical point of view. What are the technical challenges that need to be addressed right now?
2. Archictecture building
   - No hurry. Refactoring, refactoring.
   - Creating basis for the smooth shared development by multiple developers.
   - Not just modules and files, but also ways to do and principles to follow
3. Feature development, real business case user story implementation
   - Now this is the customer-centric part
   - Now the development should be fast, as tech is known (from 1, OR earlier project) and architecture built (2 OR earlier).

What happens (especially in school projects) if students jump over the phases?

- Then they are trying to eat an elephant as a whole
- Leads often to the analysis-paralysis
  - ”We cannot start because we don’t know exactly what customer needs”
  - We do not need to know exactly, to start with the phases 1 and 2!
  - In agile development we do not need to know exactly in the phase 3 either!
    - Prototyping: Create first lean version of what customer might want.
    - Iterate: Ask feedback and start working on it
    - Should be easy, if you did not skip phases 1 and 2

Software Architecture Opinion – and Questions

- There are no correct answers to these questions
- Robert C. Martin said, based on Ivar Jacobsen book, something like this:
    - ”*Architecture should be based on the business use cases of the customer* – customer-centered design.”
- What would be the pros and cons of that approach?
- Would you create your architectures _based_ on use cases / user stories?
- Or would you do more traditional way, let the use cases affect **the choice** of the architecture and how it is audited, but still base the architecture on many other factors.
    - Software company does not exist for the customers. It exists to make income to the owners.
    - Customer satisfaction and retainment is a strategy to stay alive and grow the income
    - What happens to a Software company who doesn’t do things efficiently and e.g. use the same architecture and tech stack in many projects for multiple customers? Note, the other competing SW companies do.

Notes on Robert C. Martin, aka ’Uncle Bob’

(These are just my own opinions with little more knowledge than listening to many of his videos. So take these with 2kg of salt ☺)

Author of e.g. the ”Clean code” book

- 80-90% of what he says is brilliant and should be learned from. 10-20% should be taken with a kilogram of salt. It's opionated and highly debatable.
- Thus, use his opinions and speeches as stimulant but not as source of the truth. And he is great in inspiring. The videos are to big part entertainment. And very good entertainment with lot of good advice to developers.
- But I do not know how much he e.g. has worked on modern DevOps, frameworks and cloud-native apps? Or what is his database development understanding level?
- Separate note, not related to him:
  - I have seen very famous and respected gurus reveal in their demonstrations that they do not understand basic things about something outside of their immediate expertise area. E.g. databases. True.

Why to have good Software Architecture?

- We spend some effort in the beginning, to make the development *easier, safer, faster, less tiring* later in the project.
- Finally this can be measured also in €uros. The software company will have a better foundation to serve also future customers.
- Investment + ROI (Return on investment). And are we talking about one week, one month, one year or multiple-year ROI?