



Universidad de Oviedo



# Software Architecture

## Basic definitions



SOFTWARE  
ARCHITECTURE

2025-26

Jose E. Labra Gayo

# Contents

## Definitions about Software Architecture

About software architecture

Stakeholders

Quality attributes

Constraints

# What is architecture?

Ethimologically, from greek:

Architecture = ἀρχιτέκτων

ἀρχι- "chief"

τέκτων "creator"

Architecture = Process and the product of planning, designing, and constructing buildings or **other structures**.



# Vitruvius, "De architectura"

Written between 30 to 15 BC

3 pillars of good buildings

Utilitas (usefulness):

Be useful and function well for the people using it.

Firmitas (durability):

Stand up robustly and remain in good condition

Venustas (elegance/beauty):

It should delight people

Can be applied to software systems



# What is software architecture? (1)

## Architecture [ISO/IEC/IEEE 42010:2011, 3.2]

Fundamental concepts or properties of a system in its environment embodied in its elements, relationships, and in the principles of its design and evolution

## Architecture description

Explicit work product expressing an architecture of a system, usually via models, text and graphics.

## Architecting:

Process of creating an architecture

# What is Software architecture? (2)

Fundamental structures of a system...

...which comprise:

- software elements
- relations among them
- properties of both.

# Architecture vs Design

The distinction is not always clear-cut

Architecture focuses more on:

- High level structure of a software system

- Significant design decisions of a system that...

- ....if you have to change them  $\Rightarrow$  High cost

"All architecture is design but not all design is architecture"

G. Booch

# Buildings architecture vs software architecture

## Some similarities

- Complex systems

- Developed by teams/organizations

- Used by people

- Both employ styles, patterns, tactics...

- And are affected by trends



# Buildings architecture vs software architecture

## Some differences

### Buildings architecture

More stable environment

Physical product/service

Physical limits, difficult to change

Long tradition and history

Great examples to show



### Software architecture

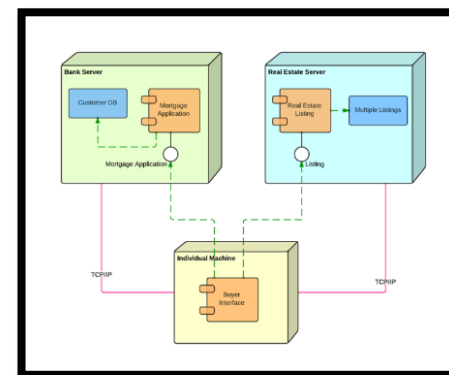
Environment changes very fast

Virtual product/service

No physical limits, easier to change

Relatively new discipline

and we can learn a lot from others



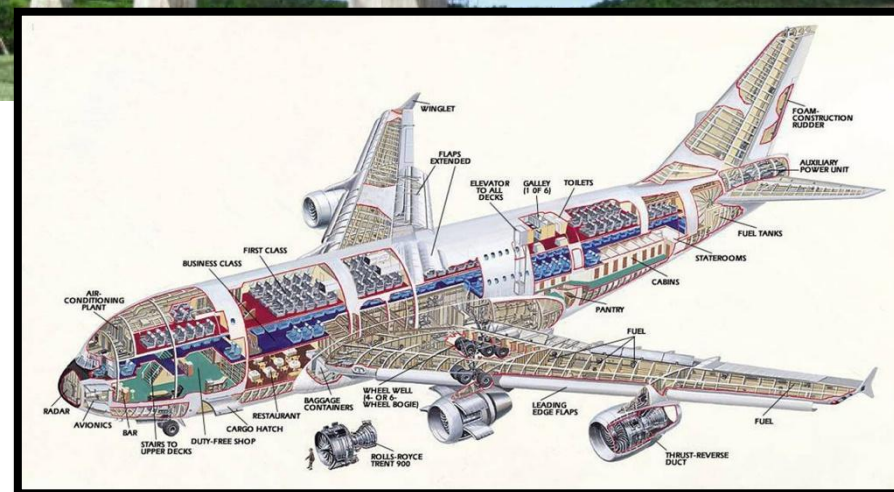
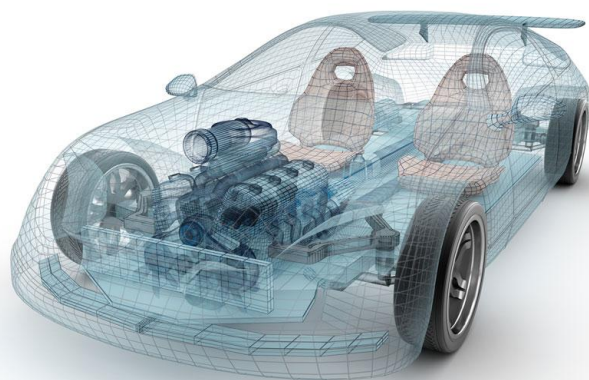
# Other similar disciplines

Civil engineering

Mechanical engineering

Aeronautics

...



# Other architectures

Business architecture

Enterprise architecture

Systems architecture

Information architecture

Data architecture

. . .

Common things about all: Structure and vision

# Benefits of software architecture

Provide a clear **vision and roadmap** for the team

Technical **leadership** and better **coordination**

Answer questions relating to **significant decisions**

Quality attributes, constraints and other cross-cutting concerns.

**Identifying and mitigating risk.**

**Consistency** of approach and standards

Leading to a well structured codebase.

**Firm foundations** for the product being built.

A structure to **communicate** the solution

At different levels of abstraction to different audiences.

# Challenges of software architecture

Architects at the ivory tower

Lack of communication

Centralization of all decisions

Bottleneck

Taking too many decisions

Deferring decisions may be better than reversing them

Big design up front

Too much unneeded diagrams and docs

Delays caused by architecting process

# Agile software architecture

Architecture that can react to its environment

- Adapting to ever changing requirements

- Also known as evolutionary architectures

- Good architecture enables agility

- Better understanding of trade-offs and decisions

Common anti-pattern:

- Adopting agile software development techniques that create non-agile software architectures

- Caused by too much focus on delivering functionality

# Laws of software architecture (\*)

## 1<sup>st</sup> law:

Everything in software architecture is a trade-off

### Corollary 1:

If an architect thinks he has found something that is not a trade-off, more likely he just haven't identified the trade-off yet

### Corollary 2:

All meaningful decisions have downsides

## 2<sup>nd</sup> law.

**Why** is more important than **how**

Question everything

Document architecture decisions





# Architecture design

## Problem domain

- Design Objectives
- Functional requirements
- Quality attributes
- Constraints
- Concerns

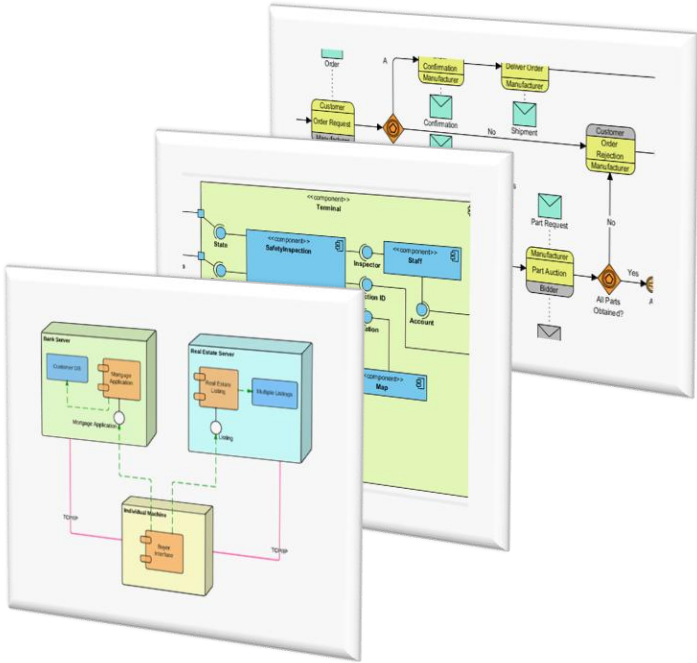
Architecture drivers (inputs)



Architect



## Solution domain



Design of the architecture (output)



# Architecture drivers

Inputs of the software architecture process

Design objectives

Functional requirements

Quality attributes

Constraints

Concerns

# Design objectives

What are the business goals?

**Why** you are designing that software?

Some examples:

**Pre-sales proposal:** rapid design of an initial solution in order to produce an estimate

**Custom system** with established time and costs which may not evolve much once released

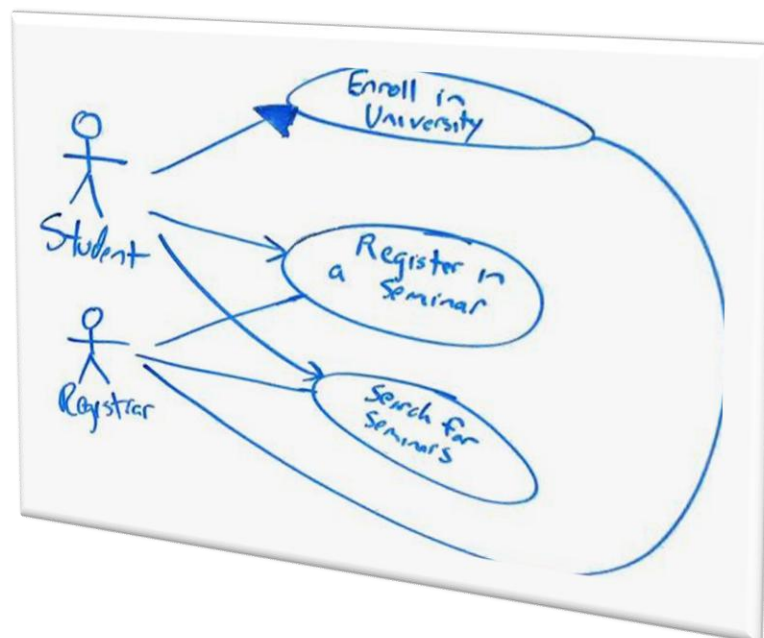
**New increment** or release of a continuously evolving system

# Functional requirements

Functionality that supports the business goals

List of requirements as use cases or user stories

## Use cases



## User stories



# Quality attributes

Measurable features of interest to users/developers

Also known as non-functional requirements

Performance, availability, modifiability, testability,...

Also known as -ilities

Can be specified with scenarios

Stimulus-response technique

*“If an internal failure occurs during normal operation, the system resumes operation in less than 30seconds, and no data is lost”*

ISO 25010: list of some non-functional requirements

List: [https://en.wikipedia.org/wiki/List\\_of\\_system\\_quality\\_attributes](https://en.wikipedia.org/wiki/List_of_system_quality_attributes)

# Quality attributes

Quality attributes determine most architectural design decisions

If the only concern is functionality, a monolithic system would suffice

However, it is quite common to see:

Redundancy structures to increase **reliability**

Concurrency to increase **performance**

Layers for **modifiability**

...

Quality attributes must be prioritized

By the client to consider system's success

By the architect to consider technical risk

# Constraints

## Pre-specified design decisions

- Very little software has total freedom

- May be technical or organizational

- May originate from the customer but also from the development organization

- Usually limit the alternatives that can be considered for particular design decisions

- Examples:

  - Frameworks, programming languages, DBMS,...

- They can act as “friends”

  - Identifying them can avoid pointless disagreements

# Concerns

Design decisions that should be made

Even if they are not stated explicitly

Examples:

Input validation

Exception management and logging

Data migration and backup

Code styles...

...

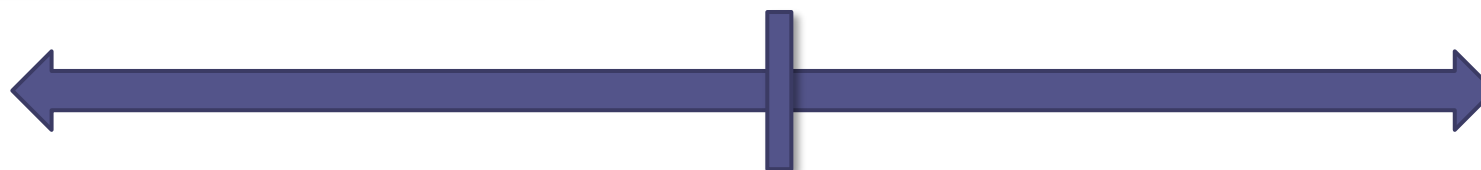
# Creativity vs Method

## Creativity

Fun  
Risk  
Can offer new solutions  
Can be unnecessary

## Method

Efficient in familiar domains  
Predictable result  
Not always the best solution  
Proven quality techniques



Architect





# Types of systems

Greenfield systems in novel domains

E.g. Google, WhatsApp,...

Less well-known domains, more innovative



Greenfield systems in mature domains

E.g. “*traditional*” enterprise applications,  
standard mobile apps

Well known domain, less innovative



Brownfield domains

Changes to existing system



# Software architect

Discipline evolves

Architect must be aware of

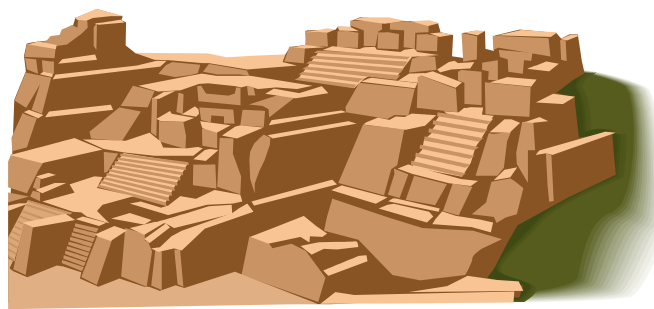
New development techniques

Styles and patterns

Best tool = experience (*no silver bullet*)

Self experience

Experience from community



Architect



# Role of software architect

