

Session 1

Architecture Models of Software Systems



This work is licensed under a Creative Commons Attribution – NonCommercial – NoDerivatives 4.0 International License.

Objectives

- Software solution and its **architecture**

- What is a software solution?
 - What is a software architecture?

- Landscape of the main **styles of architecture**

Depending on the application type and the level of abstraction

- **Why** is it useful to define architecture(s)?

Quality and beauty of code, maintenance, security, etc.

Software and Architecture



Software

- A **software** is a part of a computer system

Lies on top of the operating system and the hardware

- Executable/interpretable **instructions** and **datasets**

Compared to the physical hardware on which the system is built

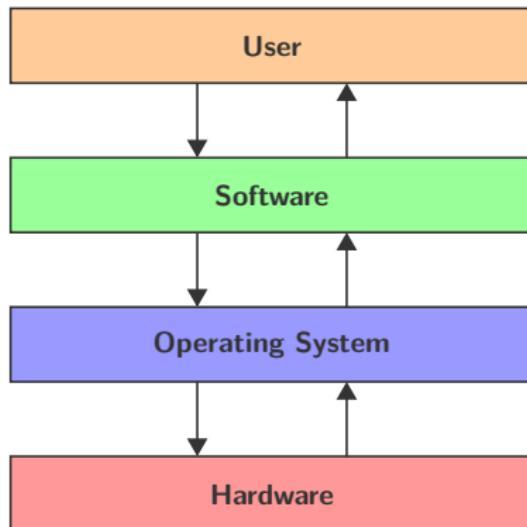
- Several abstraction levels for the **code of a software**

From machine language to high-level programming language

Computer System Structure

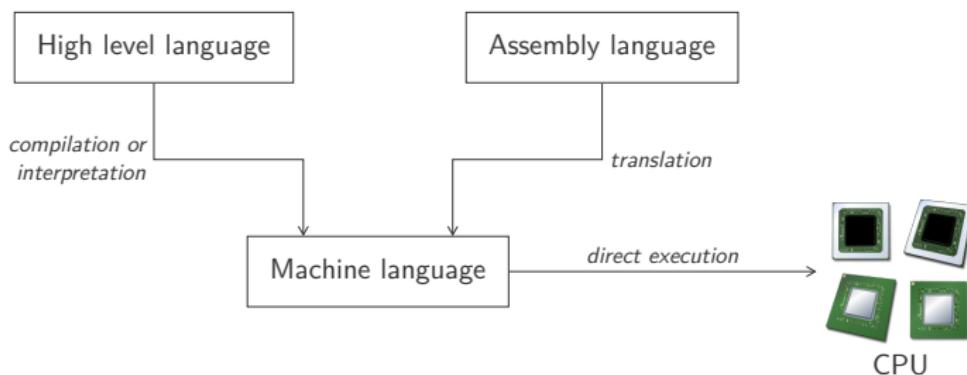
- A **computer system** can be seen as a four-layer system

Use of lower level services to provide services to higher level



Executable Program

- Transformation from programming language to a machine one
 - From a language with a more or less high abstraction level*



Types of Software

- Depending on the **purpose** or the area of use of the software
Application, system program (OS, driver, utility), malware, etc.
- Depending on the **nature** or the execution domain
Desktop/server application, script, plugin, embedded application, mobile application, microcode, etc.
- Programming **tools**
Compiler/interpreter, linker, editor, debugger, IDE, etc.

Software System

- Software system made of intercommunicating components

These components are based on software

- Several types of components, not only software

- Programs and configuration files
- System documentation describes system and its architecture
- User documentation describes how to use the system

- Several categories of software systems

*operating (OS), database management (DBMS), embedded (ES)
content management (CMS), central reservation (CRS), etc.*

Software Industry History



Architecture (1)

- **Architecture** of a software (system) describes its structure
Software elements, relations between these and properties
- Make important **choices** on the fundamental structure
Difficult to rewind because very expensive to change
- Analyses to be conducted at several **abstraction levels**
 - From code to components, libraries, etc. for developers
 - Functional blocks for team leaders
 - Very high level entities for customers

Architecture (2)

- **Functional analysis** identifies the functions to develop

WHAT to do in the software system?

- **Architecture** identifies the structure to give

HOW to do it in the software system?

Example: Pythia Platform (1)

- Automatic assessment system for student codes
Code execution, test suites, feedback generation
- Open source project and code available on GitHub

<https://github.com/pythia-project>



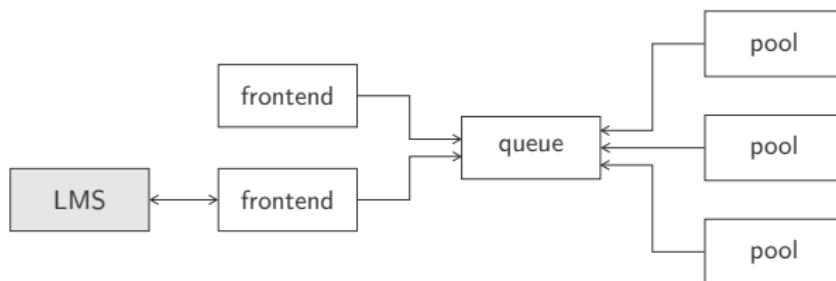
Example: Pythia Platform (2)

- Overview of the **architecture** of the Pythia platform

Used to quickly understand the structure of the software system

- Three **components and communication links** between them

- the queue receives jobs to execute
- the pool executes the job in a virtual machine (VM)
- the frontend interfaces with the user



Definition

- Not a unique **definition** of what an architecture is

Symbolic and schematic description of the different components of one or more computer programs, their interrelations and their interactions

- **Abstract representation** of a software (system)

- Software component represents a functional unit
- Connectors provide interaction between components

Code

- The code is not worth **a cent**, as ideal as it is...
 - The only importance is the implemented business functionality
 - Or rather what people are willing to pay for it
- A software and its code must **solve a problem**
...or provide entertainment for its customers

“A **stakeholder** is a person or organisation that has rights, share, claims or interests with respect to the system or its properties meeting their needs and expectations.”

Communication Tool

- Several **stakeholders** around a software system
 - The developer writes the code of the system
 - The business manager integrates the system in the enterprise
 - The final user uses and interacts with the system
 - The infrastructure manager installs and deploys the system
- Each stakeholder has its own **concerns**

Several different views of the same software system are needed

Quality

- Strong link between software system architecture and **quality**

A given architecture ensures a set of quality criteria

- Numerous **quality criteria** can be desired

*Fault tolerance, compatibility, maintainability, availability
security, extensibility, reliability, scalability, etc.*

- Stakeholder concerns translated into **requirements**

With strong/soft constraints on several quality attributes

Standard Technique

- Standard techniques to solve **recurring problems**

Just base solutions to adapt to each situation
- Depending on the **abstraction level** of the problem to solve
 - Architecture style (client-server, SOA, etc.)
 - Principle or tactic
 - Reference architecture (Java EE, etc.)
 - Pattern (MVC, n -tier, layer, etc.)

Conceptual Integrity (1)

- Overview of what the system needs to do and how

Vision to completely separate from the system implementation

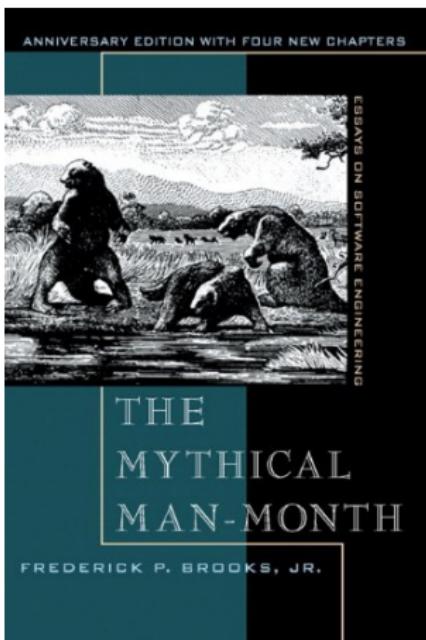
- Software architect ensures we stay in the same global vision

Changes remain in line with the architecture

- Abstraction of a complex system

It must be “intellectually understandable”

The Mythical Man-Month



Conceptual Integrity (2)

- Frederick Brooks, **The Mythical Man-Month**, 1995

Referred to as “The Bible of Software Engineering”

*“Conceptual integrity is the **most important consideration** in system design. It is better to have a system omit certain anomalous features and improvements, but to reflect **one set of design ideas**, than to have one that contains many good but independent and uncoordinated ideas... **Conceptual integrity** in turn dictates that the design must proceed from one mind, or from a very small number of agreeing resonant minds.”*

Motivation

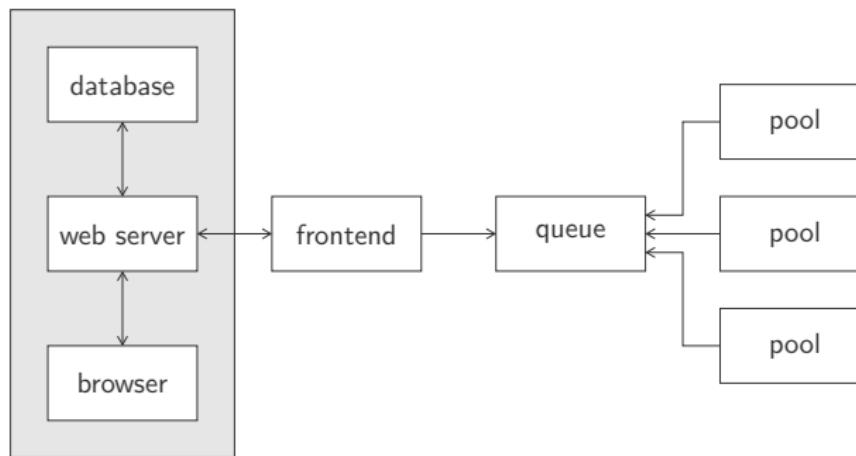
- Several **major interests** derive from architecture
 - Analysis of the behaviour of a system before implementation
 - Using the same architecture or its elements
 - Force important decision to manage schedule and budget
 - Communication tools between stakeholders
 - Risk and cost analysis and management
- Role of the **architect** in collaboration with the team leader

Software system architecture as the main tool

Example: Pythia Platform (3)

- Identification of the structure of the LMS component

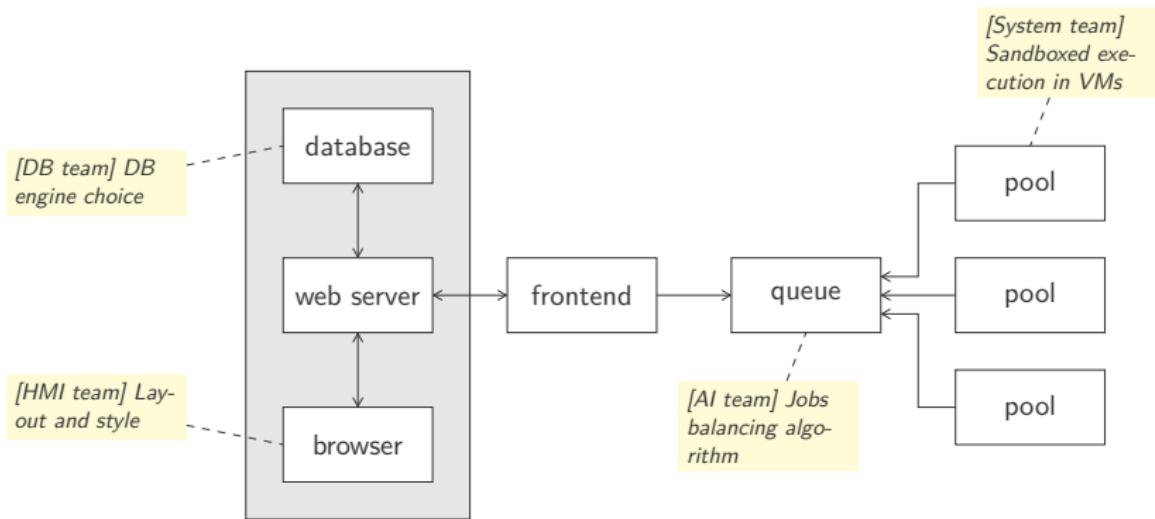
Lowering the abstraction level of the global architecture



Example: Pythia Platform (4)

- Definition of a **work plan** for the different teams

With precise specifications of the different components



Activity

1 Analysis

Establish system requirements and understand the environment

2 Synthesis

Create an architecture for the system

3 Evaluation

Evaluate whether proposed architecture meets the requirements

4 Evolution

Maintain and evolve the architecture in the face of change

ISO/IEC 42010

- Systems and software engineering — **Architecture Description**
Architecture description standard for a software (system)
- Define a **terminology and basic concepts**
 - Architecture, stakeholder, concern, etc.
 - Architectural view, architecture description language, etc.

Viewpoint and View

- Architectural **viewpoint**
 - Convention to construct, interpret, use and analyse a view
 - Operational, system, technical, logical, deployment, process
- Architectural **view**
 - Express system architecture from stakeholder's perspective
 - Addressing specific concerns following viewpoints convention
 - Consisting of one or more architecture models

Architecture Style



Architecture Presentation

- **Definition** of a software system architecture

Identification of the components and their relationships

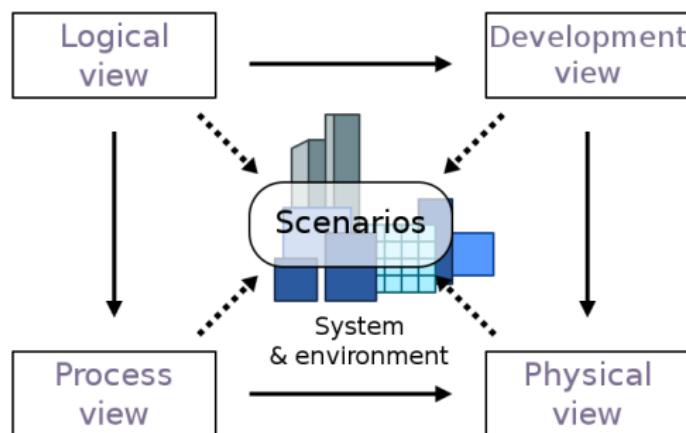
- **Representations** of an architecture

Set of views presenting specific perspectives

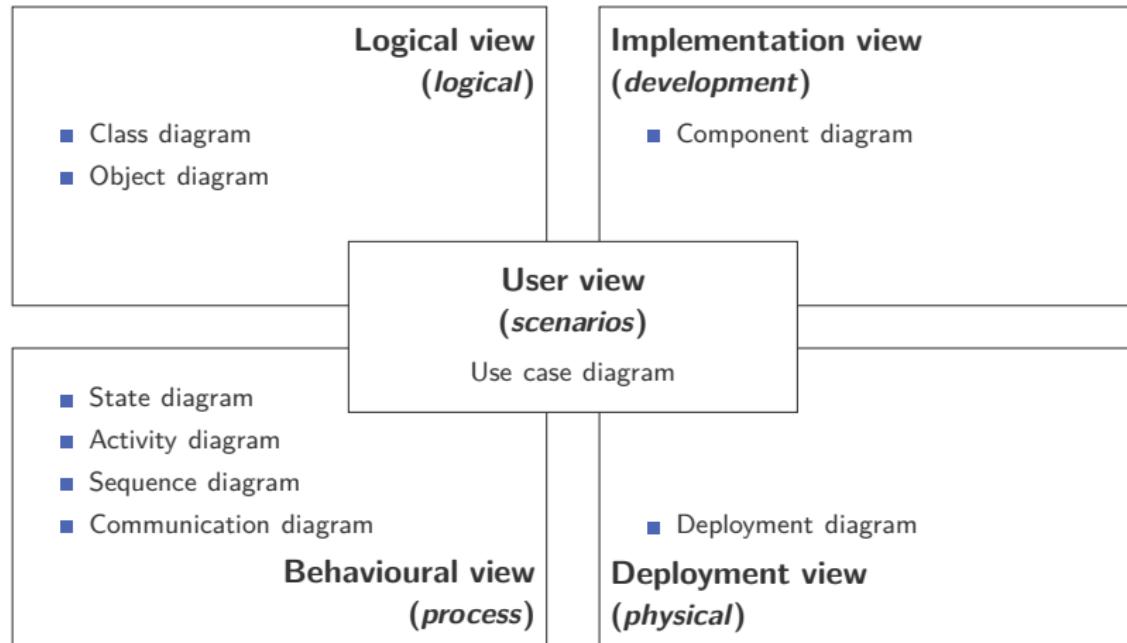
4+1 architectural view model (1)

- View model designed by Philippe Kruchten

Architecture description based on multiple concurrent views



4+1 architectural view model (2)



4+1 architectural view model (3)

- **Conceptual** view of the system
 - Logical: describes the features
 - Behavioural: describes performance, extensibility, throughput
- **Physical** view of the system
 - Implementation: describes configuration management
 - Deployment: describes topology, communication
- **User view** links the different views between them

Serves as a starting point to test architecture prototype

Style

- Architecture style related to **data**
 - Data centric: database, blackboard
 - Data flow: batch, pipe and filter
- **Hierarchical** organisation of the system

Construction with successive layers of abstraction
- Description by **implicit invocation** between components

Event-oriented, Model-View-Controller (MVC)

Data Centric Architecture

- Central repository for data communicating with customers

Read/write requests on the data

- Goal

Maintain and guarantee data integrity

- Use case

Shared and frequently used data between components

Data Flow Architecture

- Successions of data **transformations**
 - Process broken down into successive processing steps
 - Sequential style (each one in turn) or pipeline (simultaneous)
- **Goal**

Reuse and evolutivity
- **Use case**

Scientific computations workflow, stages processing

Layer Architecture

- Hierarchical organisation into a **set of layers**
 - Important to properly define interfaces between layers
 - A layer servers the higher ones and is client of lower ones
- **Goal**

Complexity ↓ and modularity, reusability and maintainability ↑
- **Use case**

Division in successive abstraction levels

Tier Architecture

- Distribution of **tiers** at the physical level

Level change through a communication middleware

- Several possibilities to **define tiers**

2-tiers: client-server, 3-tiers: client-server-database, etc.

- **Goal**

Decoupling, distribution, maintainability

- **Use case**

Distribution of a software on several machines

Implicit Invocation

- Reaction to **events** rather than explicit calls

Listeners listen to events produced by sources

- **Goal**

Flexibility of reaction

- **Use case**

Model with observers that are notified with events

Model-View-Controller

- Separating the interface from other parts of the system

The interface is more likely to change than the rest

- Made up of three components
 - **Model:** gathers domain data
 - **View:** presents/displays data
 - **Controller:** controls user-view/model interactions



Why an Architecture?

Software Quality (1)

- Architecture strongly influences the **properties of the system**

Nearly definitive choice at the architectural level

- Favours or penalises non-functional properties

Performance, security, availability, maintainability, etc.

- No single choice that optimises all criteria

Criteria relevant to the system must be chosen

Example: Pythia Platform (5)



Analysis of two choices for the web server and database

- On the **same machine**
 - More security
 - More performance (faster communication)
- On two **different machines**
 - More maintainability
 - More reliability (replication)

Compromise

- An architecture is a **compromise** of choice

Influence the quality of the system, but not a guarantee

- **Number of components** in the system

- **Few components**: communication ↓ , performance ↑
- **Many components**: performance ↓ , maintainability ↑

- **Redundancy** increases reliability/availability

But will have a negative impact on compactness and security

Software Quality (2)

- Two solutions to **improve software quality**
 - Create more unit tests to fully cover the code
 - Ask to developers to make more “*manual tests*”
- **Positive dynamics** of software improvement
 - 1 Choose quality criteria
 - 2 Establish how to measure these criteria
 - 3 Define what is a good quality software

Quality Criterion and Measure (1)

- Common language to model the qualities of a software

Vocabulary and properties in “-ity”

- Five main criteria from the ISO/CEI 9126 standard

- **Functionality**

- Respect of specifications, solving the problem of the user
 - Measured with feature coverage

- **Reliability**

- Resistance to user or external services errors/faults
 - Measured with number of passed tests (unit, integration, etc.)
 - Measured with code coverage

Quality Criterion and Measure (2)

- Five main criteria from the **ISO/CEI 9126** standard

- **Usability**

- Should be easy/intuitive for the user to use the UI
- Measured with user feedback
- Measured with tests on groups of people

- **Efficiency** response time, memory, processor, etc.

- Measured with response time, CPU/memory consumption
- Measured with execution time of long operations

- **Maintainability** standards, documentation, etc.

- Standard, documentation and maintenance at a lower cost
- Measured with code complexity, code compliance

Software Quality Criteria

- Autonomy
- Compatibility
- Composability
- Efficiency
- Extensibility
- Reliability
- Integrity
- Interoperability
- Maintainability
- Portability
- Reusability
- Simplicity
- Transparency
- Validity
- Verifiability
- ...

Maintenance and Evolution

- An architecture ease the **maintenance and evolution**
 - Easy identification of components to be modified/fixed
 - Easy extension through interfaces
 - Make it possible to reuse components
- Work on **individual parts** in isolation
 - Teamwork, updates of parts of the system*
- Development with **good intentions** under difficult conditions
 - Deadlines, different qualifications in teams, old code, etc.*

Technical Debt (1)

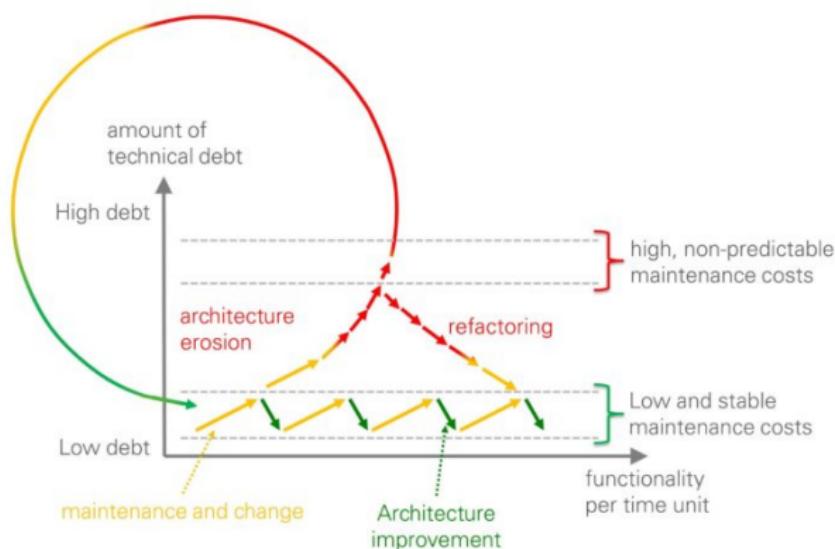
- Need for high-quality architecture with little **technical debt**

Maintenance should be structured programming, not defensive
- Due to wrong or sub-optimal **technical decisions**
 - Always lead to additional costs and very expensive maintenance
 - Implementation debt and design/architecture debt
- **Sustainable software architecture** follows cognitive mechanisms
 - Chunking, formation of hierarchies and structure of schemata
 - Modularity, hierarchical layering and pattern consistency

Technical Debt (2)

- Maintenance is an opportunity for **architecture improvement**

In addition to quick bug fixes since only few errors

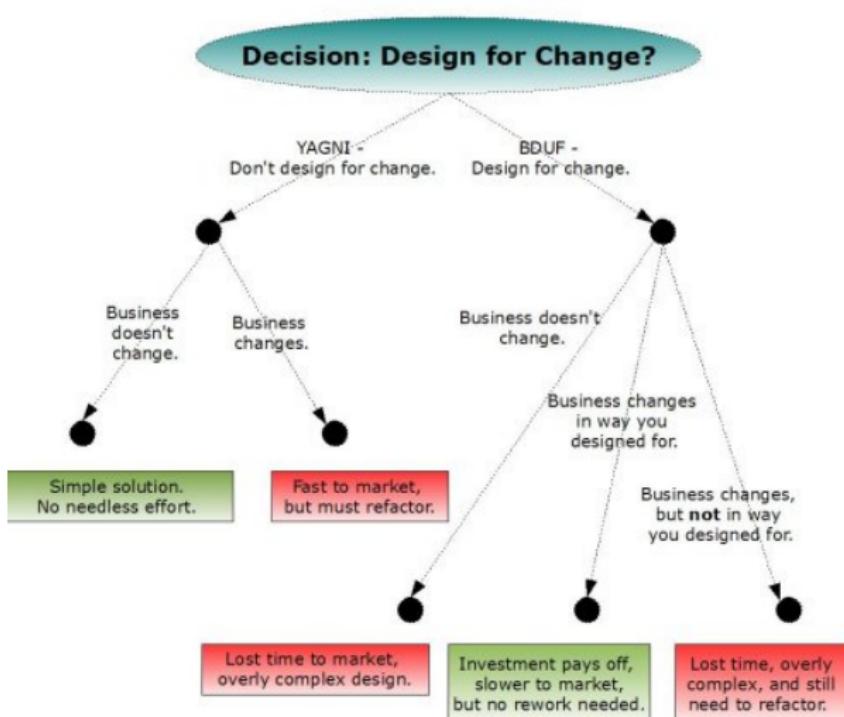


Architect Dilemma (1)

"Your system needs to do X now, but in the future it may need to do Y as well. Knowing that X and Y are closely related, do you spend a little extra time now and design X in such a way that adding Y in the future would be easy? Or do you keep things simple, focus on the more pressing X now, and ignore Y until the day when you really need it?"

- Two opposing sides to tackle this problem
 - Big Design Up Front (BDUF)
Invest for dividends in maintainability, extensibility, quality...
 - You Arent Gonna Need It (YAGNI)
Agile approach, Y maybe not necessary anymore, changed to Z...

Architect Dilemma (2)



Good Architecture

- An architecture may be **inherently good**
 - Contribute to a better extensibility, scalability, reliability, etc.
 - If we ignore the eternal debate BDUF versus YAGNI
- *“A good architecture is one which meets the needs of the entity that funds its existence.”*
 - The “good” is not an intrinsic property of a system
 - Sacrifices can be made at the request of business

Software Architect

- Role of the **software architect** is to define an architecture
 - which satisfies the **requirements** of the customer
 - given the **constraints** imposed on the system
- **Communication** and promotion of the system architecture

Bridge between the different stakeholders of the system
- Defending the **conceptual integrity** of the architecture

While regularly criticising and refining it

References

- Martin Campbell-Kelly, *From airline reservations to Sonic the hedgehog: A history of the software industry*, The MIT Press, London 2003. (ISBN: 978-0-262-53262-4)
- Nikolay Ashanin, *Stakeholders in Software Architecture*, November 4, 2017.
<https://medium.com/@nvashanin/stakeholders-in-software-architecture-6d18f36250f9>
- Frederick P. Brooks Jr., *The Mythical Man-Month: Essays on Software Engineering*, 2nd edition, Addison-Wesley, 1995. (ISBN: 978-0-201-83595-3)
- Anubha Sharma, Manoj Kumar, & Sonali Agarwal, *A complete survey on software architectural styles and patterns*, 2015, Procedia Computer Science, Vol. 70, pp.16-28.
- Ashish Kumar, *Software Architecture Styles: A Survey*, 2014, International Journal of Computer Applications, Vol. 87, No. 9.
- Architecture, The How of any system, *The importance of Conceptual Integrity*, October 29, 2011.
https://architecture.typepad.com/architecture_blog/2011/10/the-importance-of-conceptual-integrity.html
- Carola Lilienthal, *Sustainable software architecture*, July 25, 2019.
<https://jaxenter.com/sustainable-software-architecture-technical-debt-160372.html>
- Ben Northrop, *The Architect's Dilemma*, January 20, 2008.
http://www.bennorthrop.com/Home/Blog/2008_01_20_decision_tree.php
- Ben Northrop, *What is 'Good' Software Architecture?*, April 30, 2009.
http://www.bennorthrop.com/Essays/2009/what_is_good_architecture.php

Credits

- Book pictures from Amazon.
- Andreas Wecker, July 23, 2010, <https://www.flickr.com/photos/wecand/4820878341>.
- <https://openclipart.org/detail/100267/cpu-central-processing-unit>.
- Игорь М, November 12, 2011, <https://www.flickr.com/photos/44353614@N02/7008492333>.
- Wikimpan, July 14, 2016, https://en.wikipedia.org/wiki/File:4%2B1_Architectural_View_Model.svg.
- Bud Ellison, January 10, 2015, <https://www.flickr.com/photos/budellison/16220593436>.
- Carola Lilienthal, July 25, 2019, <https://jaxenter.com/wp-content/uploads/2019/07/Figure-1-e1563972108757.jpg>.
- Ben Northrop, January 20, 2008, http://www.bennorthrop.com/Home/Blog/yagni_medium.jpg.