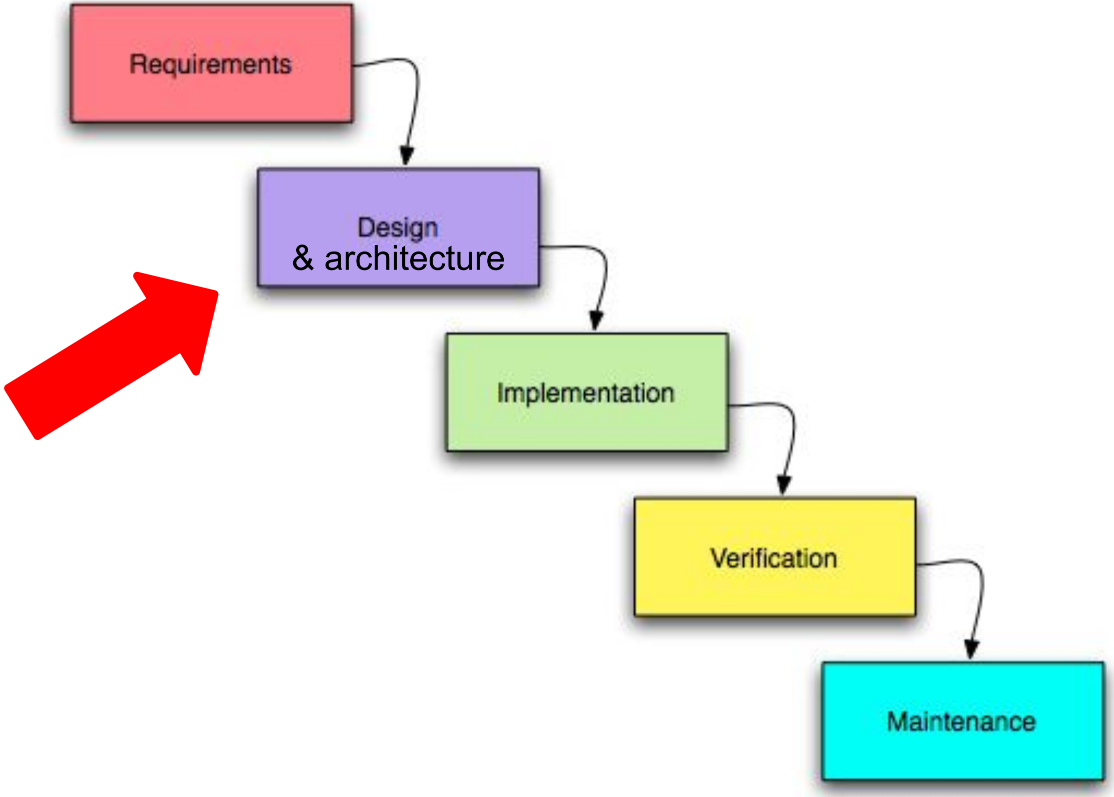
Architecture

Princípy tvorby softvéru, FMFI UK

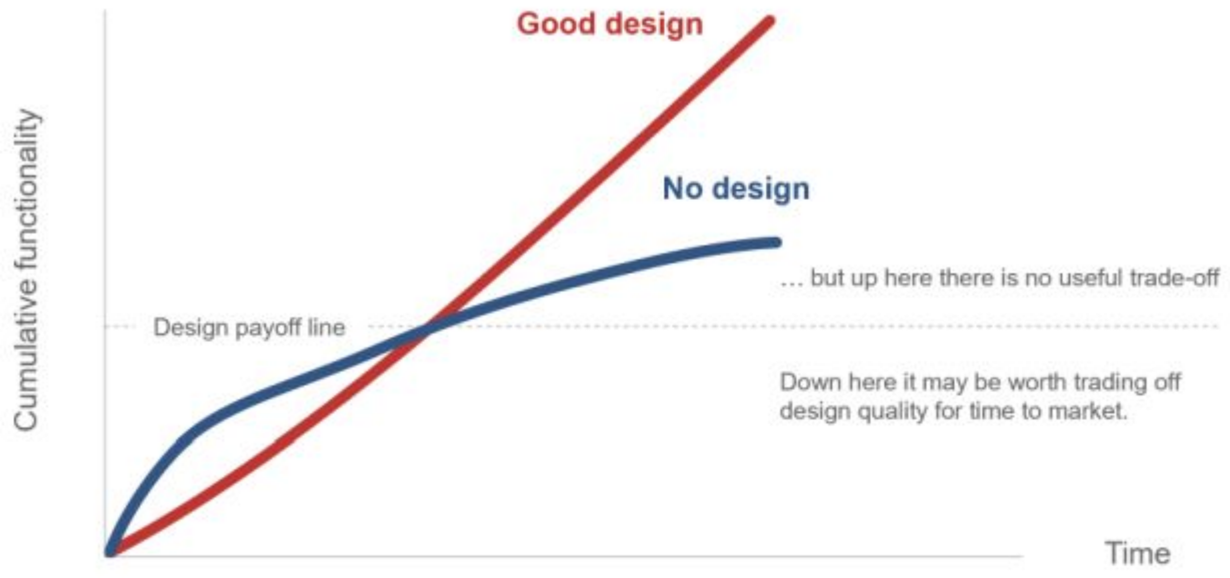
Jana Kostičová, 11.10.2023

SDLC - Waterfall



- Camelot is based on the **client-server model** and uses remote procedure calls both locally and remotely to provide communication among applications and servers."
- "**Abstraction layering** and system decomposition provide the appearance of system uniformity to clients, yet allow Helix to accommodate a diversity of autonomous devices. The architecture encourages a **client server model** for the structuring of applications."
- "We have chosen a **distributed, object-oriented approach** to managing information."
- "The easiest way to make the canonical sequential compiler into a concurrent compiler is to **pipeline** the execution of the compiler phases over a number of processors. . . . A more effective way [is to] split the source code into many segments, which are concurrently processed through the various phases of compilation [by multiple compiler processes] before a final, merging pass recombines the object code into a single program."

Is it worth the effort to design software well?



Software architecture

= the organization or structure of a system, where the system represents a collection of components that accomplish a specific function or set of functions.

- A (software) component is a part of a software system that encapsulates a specific piece of functionality, e.g., libraries, modules, web components, plugins, ..
- Components serve as the building blocks for the structure of a system
- Components are connected via interfaces
- Components are typically specified in different views to show the relevant functional and non-functional properties of a software system

Interface

(Software) interface is a shared boundary across which two or more separate (software) components of a computer system exchange information.

- ABI - Application binary interface - typically not relevant (created by compiler / other tools).
- API - Application programming interface
- User interfaces

4+1 architectural view model

1. Logical view
   - Describes the system in terms of components, their interactions, and the functionality they provide
   - The perspective of end users (and stakeholders in general}
   - *UML: Use Case diagrams, UML Class diagrams, …*
2. Process view
   - Captures dynamic behavior of the system - the interactions and collaborations among processes, tasks, threads, and components during runtime
   - Important for understanding concurrency, performance, and resource utilization.
   - *UML: Sequence diagram, Communication diagram, Activity diagram, …*

4+1 architectural view model

3. Development view
    - Describes software organization - SW modules and components, their relationships, source code organization, ….
    - The perspective of developers
    - *UML: Package diagrams, Component diagrams*

4. Physical view
    - Describes the system's physical architecture, including hardware components, network topology, and distribution of software components across different machines or nodes
    - Addresses concerns related to deployment, scalability, and performance optimization
    - *UML: Deployment diagram*

4+1 architectural view model

5. Scenarios

- “+1” aspect of the model
- Illustrate how the system functions in real-world situations, using a small set of use cases (scenarios)

(Modern) principles of software architecture

- Separation of Concerns (SoC)
  - Keeping different aspects of the system's functionality or behavior separate and well-defined
  - Each part of the codebase has a single responsibility that makes the code more maintainable and understandable
- Modularity
  - Organizing software into discrete, interchangeable components or modules
- Avoid Big Design Up Front (BDUF)
- Build to change instead of build to last
- Use consistent principles within the components / layers / subsystems
- …

Architectural styles and patterns

Similar to design patterns:

- Provide abstract framework for a family of systems
- Help communication

Architectural styles and patterns

Architecture addresses a wide variety of issues

We have various types of styles/patterns and some of the, can be mutually combined

- Deployment
- Structure
- Communication
- Domain
- Network
- ….

Common architectural styles / patterns

- Client-server model
- Peer-to-peer model

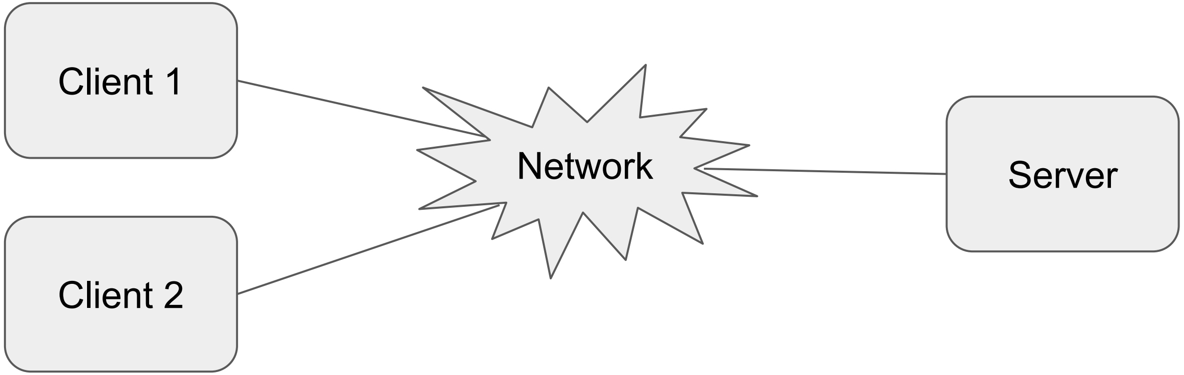
- Component-based architecture
- Service-oriented architecture
- Microservices architecture

- Layered architecture
- Domain-driven development
- Model-view-controller

Client-server model

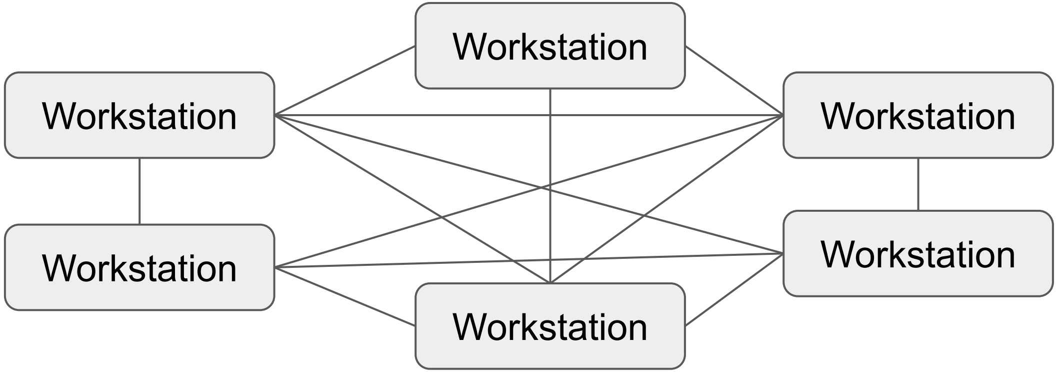
- Used in networking / distributed systems
- One or more clients connected to a server over a network or internet connection.
- The server hosts, delivers and manages most of the resources and services to be consumed by the client

- Example:
  web browsing



Peer-to-peer (P2P) model

- Used for distributed systems
- Computers, devices, or nodes within a network (peers) communicate and **collaborate directly** with each other without the need for a centralized server or hierarchy of control
- Each peer has equivalent capabilities, and they can act both as clients and servers, sharing resources, data, or services with one another.



Component-Based architecture

- System consists of loosely coupled **components**
- “Separation of Concerns” is applied
- Components are primarily intended for use within a single application, their interactions are local, within the boundaries of that application

=> modularity, reusability, interoperability, encapsulation, maintainability, scalability, …

Service-Oriented architecture (SOA)

- **Services** are the fundamental building blocks of the system
- Service = a self-contained unit of functionality that represents a specific business process or capability
- Services can be considered as **software components** that expose well-defined **interfaces** (usually via standardized protocols like HTTP or SOAP) and can be invoked by other services or applications

- Common approaches to implement SOA
  - SOAP (Simple Object Access Protocol)
  - REST (Representational State Transfer)

Microservice architecture

- A variant of SOA
- Loosely coupled, **fine-grained** services
- Microservices focus on one thing and operate **independently**

- Commonly used e.g., in cloud-native applications

Medium - SOA and Microservices Architecture comparison

Domain Driven design (DDD)

- The software systems is build “around” the core domain knowledge and concepts of a business

- The structure and language of software code (class names, class methods, class variables) and data entities **should match the business domain**

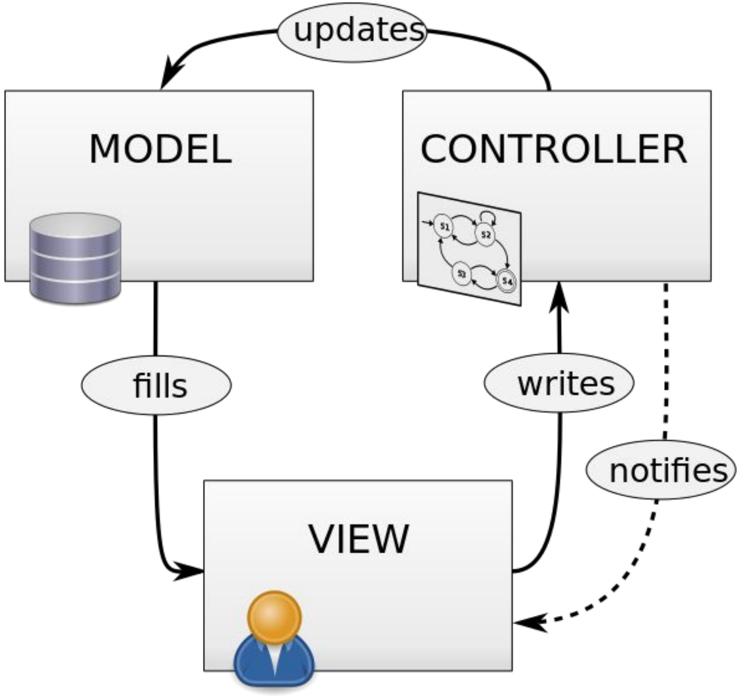
- Example: if software processes loan applications, it might have classes like *"loan application"*, *"customers"*, and methods such as *"accept offer"* and *"withdraw*".

Layered architecture

- Components within the layered architecture pattern are organized into **horizontal layers**
- Each layer performs a specific role within the application
- 3-layer architecture:
  - Presentation layer (UI layer)
  - Application layer
  - Data access layer
- Physical view - 3-tier architecture - example:
  - Presentation layer (UI layer) - web browser
  - Application layer - web server(s)
  - Data access layer - database server(s)

Model-view-controller

- **The model** manages the data of the application.
- **The view** renders presentation of the model in a particular format (chart, table, ..)
- **The controller** processes the user input and updates the data model objects.



Resources

- SWEBOOK v3
- Ian Sommerville: Software Engineering (10th edition)
- Robert Lukotka: Architecture
- Martin Fowler: Design Stamina Hypothesis
- Wikipedia: Domain-driven design
- MVC Diagram (Model-View-Controller) byXinfe, CC BY-SA 3.0