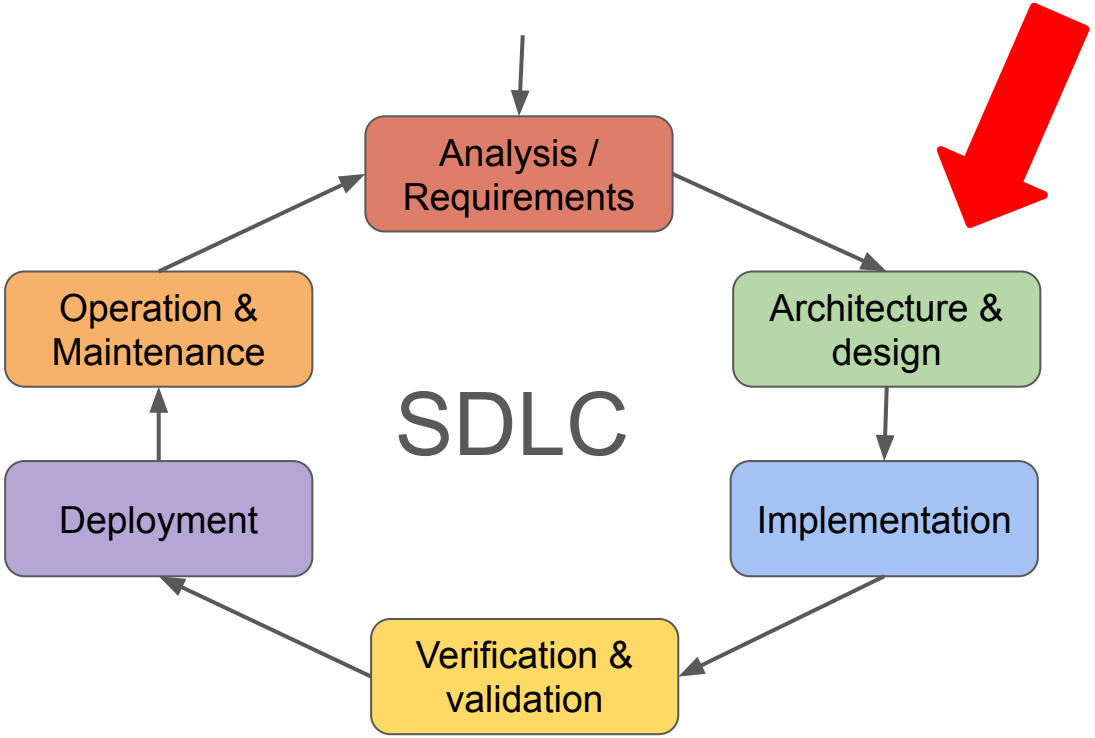
Architecture & design

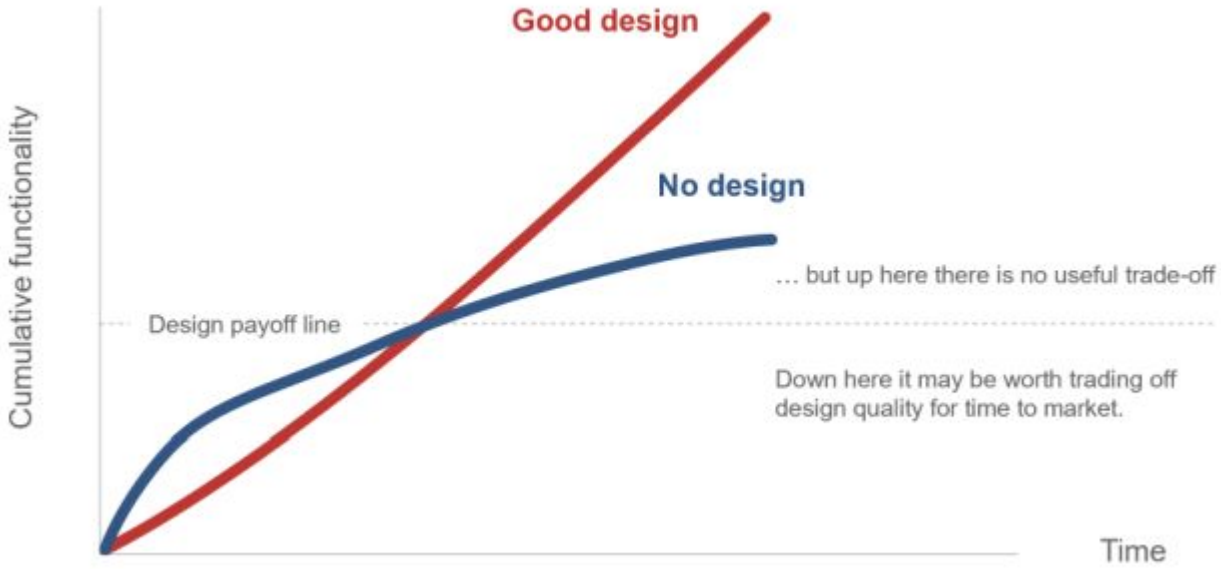
Intro, 4+1 architectural view model

Jana Kostičová, 22.10.2025


Analysis / Requirements
Architecture & design
Implementation
Verification & validation
Deployment
Operation & Maintenance
SDLC

- 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? [5]



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 modular, cohesive unit of a software system that encapsulates related functionality
- Components serve as the building blocks for the structure of a system
- Components are connected via well-defined 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

Architecture vs design

Software architecture

- High-level structure of the entire system and its division into a set of components

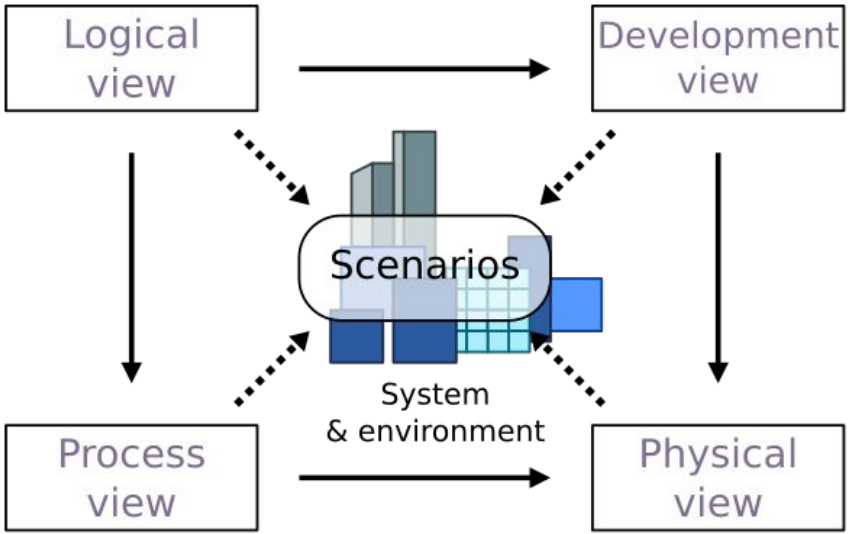
Software design

- Internal structure of individual components

4+1 architectural view model

= a model "describing the architecture of software-intensive systems, based on the use of multiple, concurrent views"

Introduced by Philippe Kruchten in 1995 [7]



The “4+1” view model is rather “generic”: other notations and tools can be used, other design methods can be used, especially for the logical and process decompositions, but we have indicated the ones we have used with success.

Philippe Kruchten

4+1 architectural view model: LOGICAL VIEW

- Describes the system in terms of components, their relations, and the functionality they provide
- The perspective of end users (and stakeholders in general)
- Mapping of requirements to logical components (e.g., modules, subsystems)
- Overlap with “Requirements” phase
- *UML: Use Case diagrams, UML Class diagrams, UML Component diagrams, …*

Logical components - identification

Example: E-commerce store logical components

Product Catalog
- Browse/search products, filter, view details

Shopping Cart
- Add/remove/update cart items, calculate totals

Order Management
- Create orders, manage order status, history

Payment Processing
- Payment gateway integration, transaction management

User Account / Authentication
- Login, registration, profile, address book

Inventory Management
- Track stock levels, availability

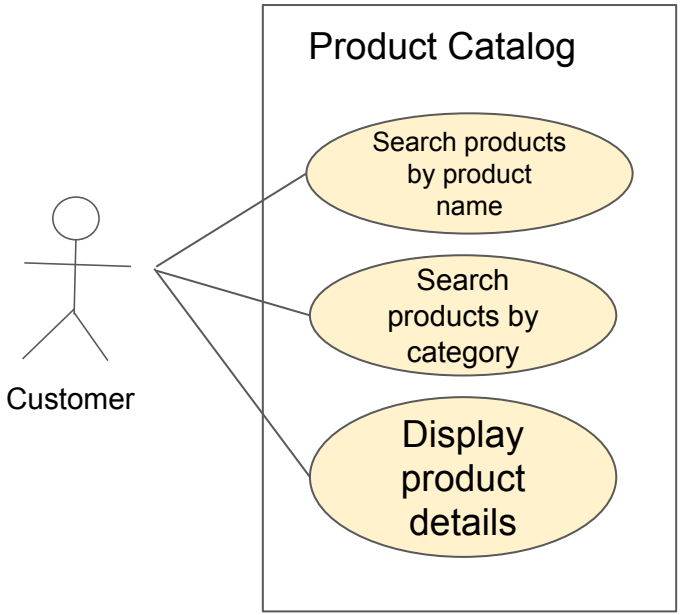
…

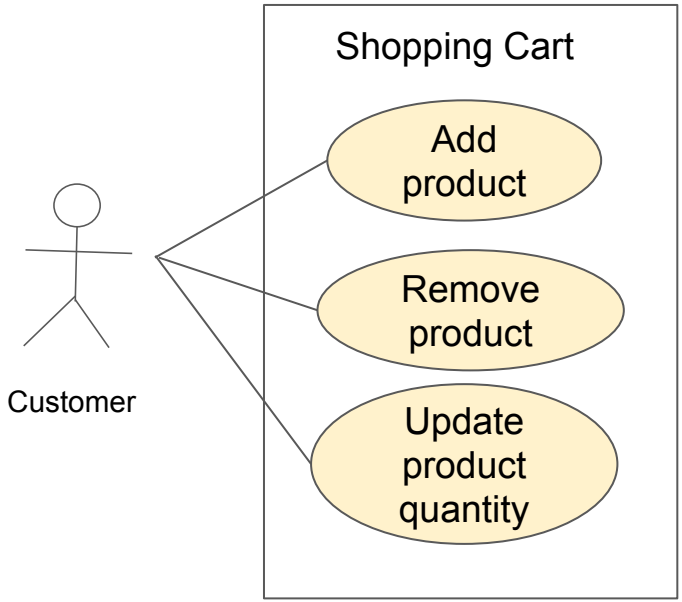
How to identify logical components?

- By analyzing requirements and domain model.
- Following the key principles:
  - **Strong cohesion** - a logical component encapsulates closely related functionality
  - **Loose coupling** - a logical component has minimal dependencies to other components and interact with them through well-defined interfaces

Logical components - UML Use Case Diagram

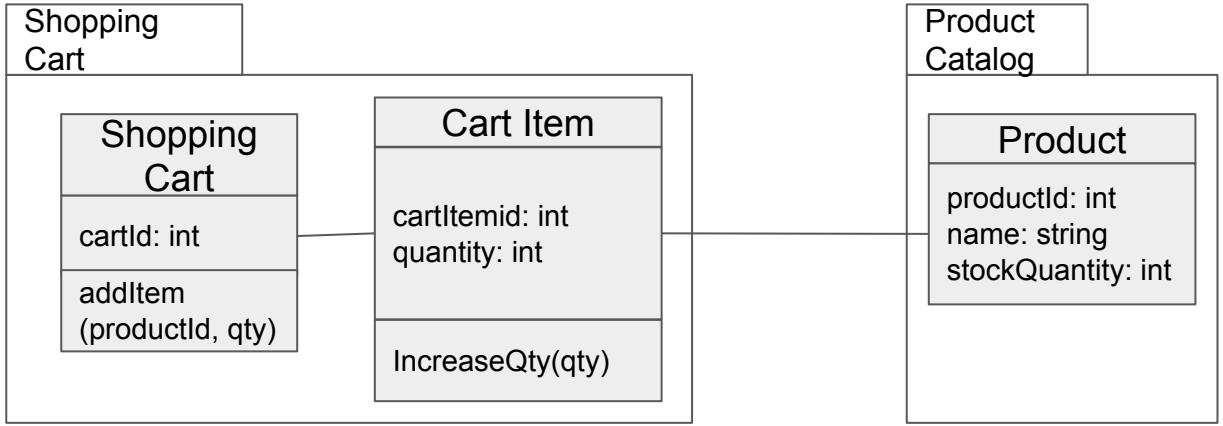
- The “system boundary” actually defines a logical component (subsystem)
- A similar approach can be used for requirements in the form of a structured text - text hierarchy is based on logical components





Logical components - UML Class Diagram

- “UML packages” are used to depict logical components and group related functionality (classes)
- Combination of architecture and design - components are white boxes, their content is shown here



Logical components - UML Component Diagram

Example With comments Without comments

- A UML Component Diagram depicts components and the interfaces through which they interact
- Components can form a hierarchy — for example, a system may be divided into subsystems (higher-level components), each containing several related components
- Lowest-level components are black boxes - their content is not shown here

Component

= a modular unit with well-defined interfaces [1]

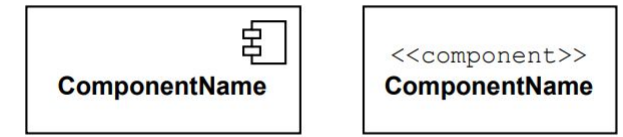
- **Notation:** Specialized class (may be nested)
- Interaction points - ports

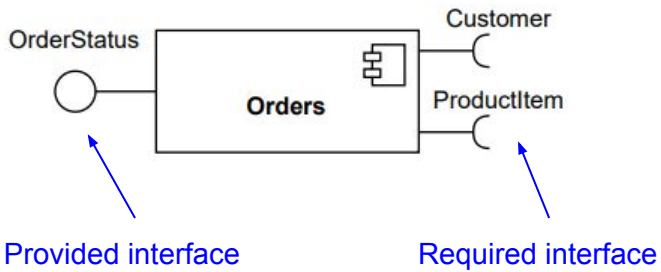
Interfaces

- Provided and required interfaces
- **Notation:** Ball and socket
- Assembly connector vs dependency

Delegation connector

- Parent component delegates the responsibility for fulfilling that interface to one of its internal parts

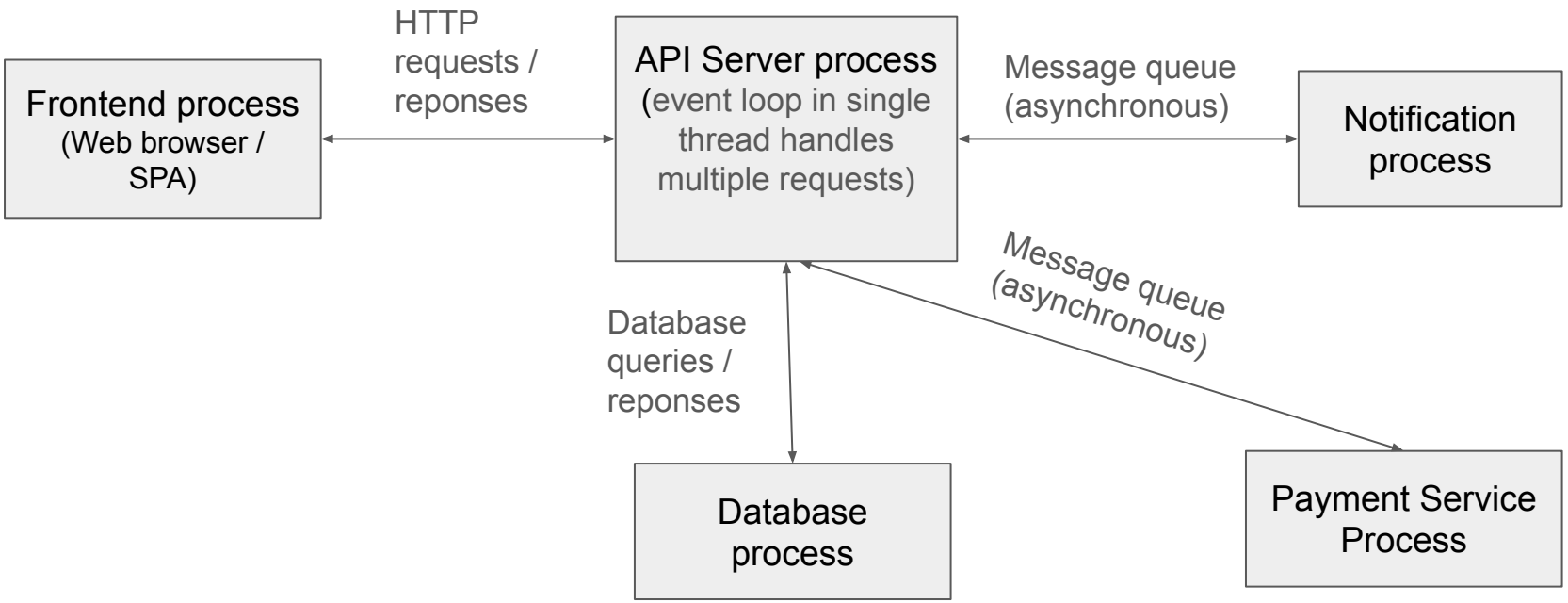




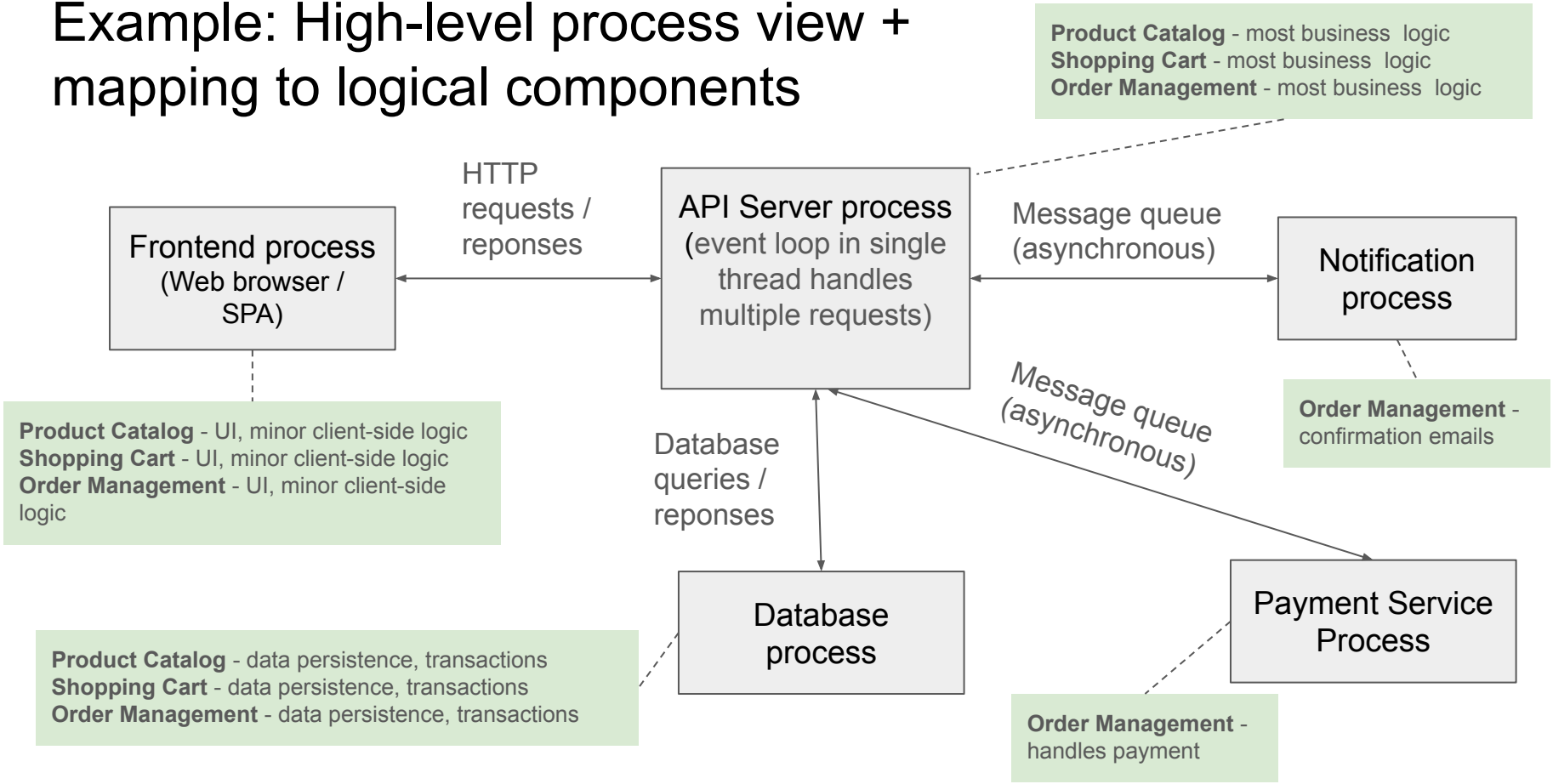
4+1 architectural view model: PROCESS VIEW

- Captures dynamic behavior of the system - the interactions and collaborations among processes, tasks, threads, and components during runtime
- Mapping of logical components to their runtime realization as processes and threads
- Important for understanding concurrency, performance, and resource utilization.
- *UML: Sequence diagram, Communication diagram, Activity diagram, …*

Example: High-level process view



Example: High-level process view + mapping to logical components



Example: Sequence diagram

Facebook user authentication

More detailed process view

4+1 architectural view model: DEVELOPMENT VIEW

- Describes software organization - repositories, SW modules and components, their relationships, source code organization, ….
- Mapping of logical components into implementation
- The perspective of developers
- *UML: Package diagrams, Component diagrams*

Example: Implementation

(Git) repository ecommerce-store-frontend

```
/src
  /assets      (static resources)
  /components  (reusable page components)
  /services    (business logic/state management)
     productClientService.js
     cartClientService.js
     orderClientService.js
     …
  /pages
     productPage.js
     cartPage.js
     orderPage.js
     …
```

(Git) repository ecommerce-store-backend

```
/config       (db connection params, …)
/src
  /middleware (request/response handlers - auth,
               Logging, error handling...)
  /models     (data models, DB schemas, ORM)
    product.js
    cart.js
    order.js
    …
  /services  (business logic/use case implementation)
    productAppService.js
    cartAppService.js
    orderAppService.js
    …
  /routes    (api)
    productRoutes.js
    cartRoutes.js
    orderRoutes.js
    …
```

Example: Implementation + mapping to logical components

(GitHub) repository ecommerce-store-frontend

```
/src
  /assets       (static resources)
  /components   (reusable page components)
  /services     (business logic/state management)
     productClientService.js
     cartClientService.js
     orderClientService.js
     …
  /pages
     productPage.js
     cartPage.js
     orderPage.js
     …
```

"Product Catalog" maps ~ to highlighted codebase elements

- Assets and page components: some of the included files may be a part of the mapping
- The example does not contain full imlementation

(GitHub) repository ecommerce-store-backend

```
/config       (db connection params, …)
/src
  /middleware (request/response handlers - auth,
               Logging, error handling...)
  /models     (data models, DB schemas, ORM)
    product.js
    cart.js
    order.js
    …
  /services  (business logic/use case implementation)
    productAppService.js
    cartAppService.js
    orderAppService.js
    …
  /routes    (api)
    productRoutes.js
    cartRoutes.js
    orderRoutes.js
    …
```

Implementation - what to consider

E-commerce store

- Monorepo vs multi-repo approach
- Layered architecture pattern - technical separation of concerns
  - **Presentation layer** - frontend: src/assets, src/components, src/pages
  - **Application layer** - frontend: src/services, backend: src/routes, src/middleware
  - **(Domain layer)** - backend: src/model (part), src/services (part)
  - **Data access layer** - backend: config.js, src/model (part), src/services (part)
- Code structure: layers vs logical components (or combination)

4+1 architectural view model: 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*

Artifacts and deployment targets

- **Artifact:** any deployable or executable piece of software or configuration

  (JAR file, .NET DLL / EXE, Python wheel / package, Node.js bundle, Docker image, DB migration scripts, configuration files, assets, …)

- **Deployment target / node:** a physical or virtual computational resource that executes artifacts

  Virtual - VM, Docker container, Software-based server…

  Physical - physical server, mobile device, workstation, various network physical elements, …

Examples

Deployment in virtual environment

```
Physical Server (physical)
  └──> Virtual Machine (virtual)
    └──> Container (virtual)
      └──> Apache Tomcat Server (virtual)
        └──> application app.war (artifact)
```

Deployment directly onto the host operating system

```
Physical Server (physical)
  └──> Operating System (Linux/Windows) (virtual)
    └──> Apache Tomcat Server (virtual)
      └──> Application app.war (artifact)
```

Network Infrastructure

- UML deployment diagrams could be used
  - May not provide enough detail when representing network-level components
- Often custom format
- Some common network-level components:
  - Routers, Switches, Gateways
    - Manage the traffic flow between networks and devices
  - Firewalls and Proxies
    - Secure the network by controlling access and monitoring traffic
  - Load Balancers and CDNs
    - Optimize application delivery by distributing traffic and caching content
  - DNS servers
- Modern cloud architectures
  - Automatic scaling, elastic load balancing, CDNs, managed databases and storage solutions, security, …

→ Strong focus on non-functional requirements (scalability, data backup, data durability, response time, security, …)

Network Infrastructure - examples

- Network Architecture Diagrams
- Amazon AWS
- Google Cloud

4+1 architectural view model: SCENARIOS

- “+1” aspect of the model
- Illustrate how the system functions in real-world situations, using a small set of use cases (scenarios)
- Cover also internal functionality of the system (unlike scenarios in “Requirements” phase)

User clicks Checkout in SPA

1. FrontendProcess validates cart and sends checkout request to APIServerProcess
2. APIServerProcess:
   a. Validates order
   b. Persists order in DatabaseProcess
   c. Sends payment request to PaymentServiceProcess
3. PaymentServiceProcess
   a. Processes payment (success/failure)
   b. Responds to APIServerProcess
4. APIServerProcess:
   a. Updates order status in DB
   b. Publishes OrderConfirmation message to NotificationProcess via MQ
5. NotificationProcess sends email to user asynchronously