2DV604

Software Design and Modeling Languages

Linnæus University

Today’s takeaways

Some design decisions have more impact than others

Architecture is design at a high level

- components,
- interfaces
- responsibilities

Let’s have a 2nd look at the challenge!

Architecture …what?!?

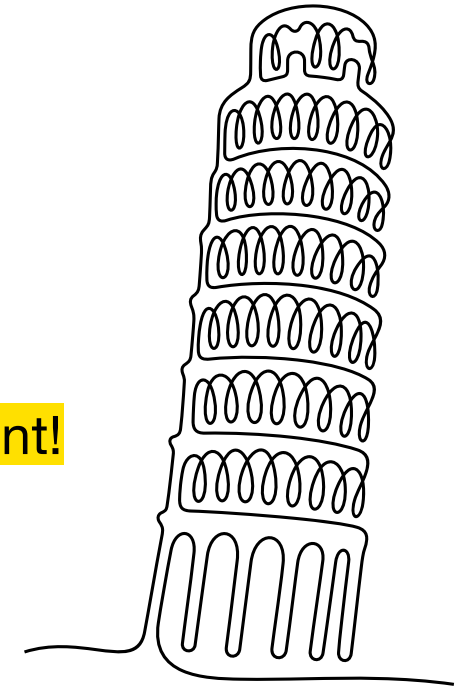
Decisions – at all levels

In a software development project, some decisions have more impact.

You may rank decisions.

- according to importance for achieving the business goals.
- according to how expensive it is to change the decision.

Design decisions with high impact are architecturally significant!



Design Decisions – Examples

Architecturally significant

Communication strategy in a distributed system

Identification and Authentication mechanism for security

Not architecturally significant (with exceptions of course)

Choice of data structure or type
Class internal decisions
Choice of algorithm, for example, password encryption

Adding to the problem

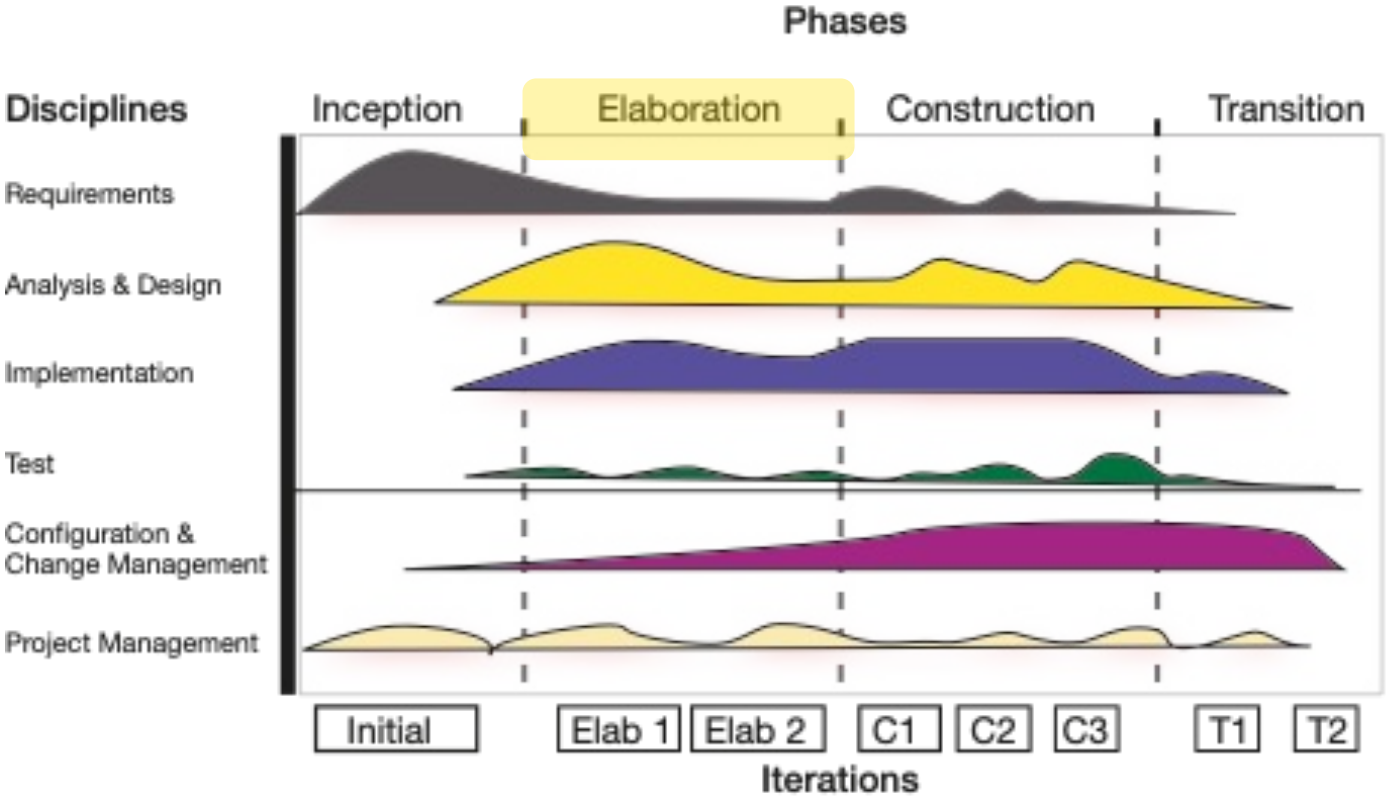
Some of the more important decision must be made early, even before we have a system!

They are often not directly testable as for instance code is.

In a project of some size, the decision impact several teams.

The activity must be fitted into the regular development process.

the Unified Process

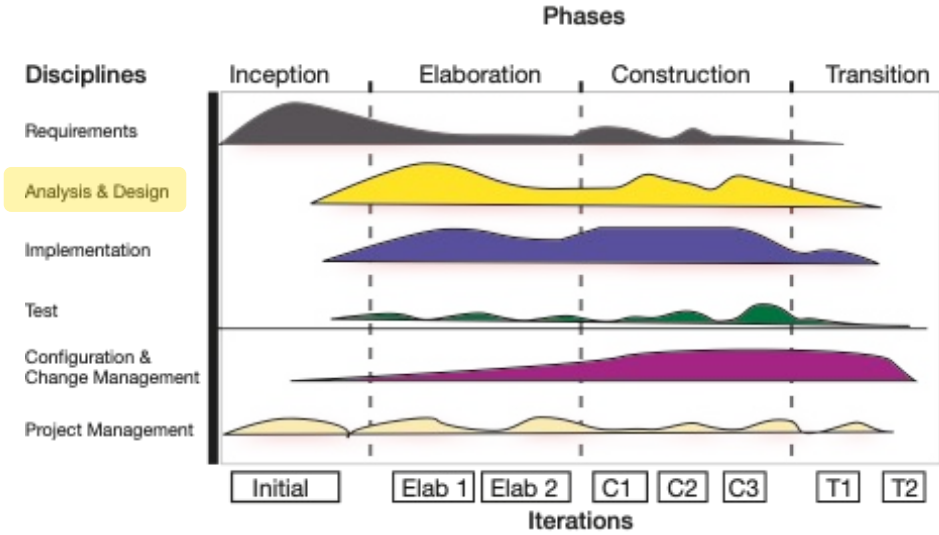


The Unified Process

Inception – define project & product scope, a vision and establish the business case

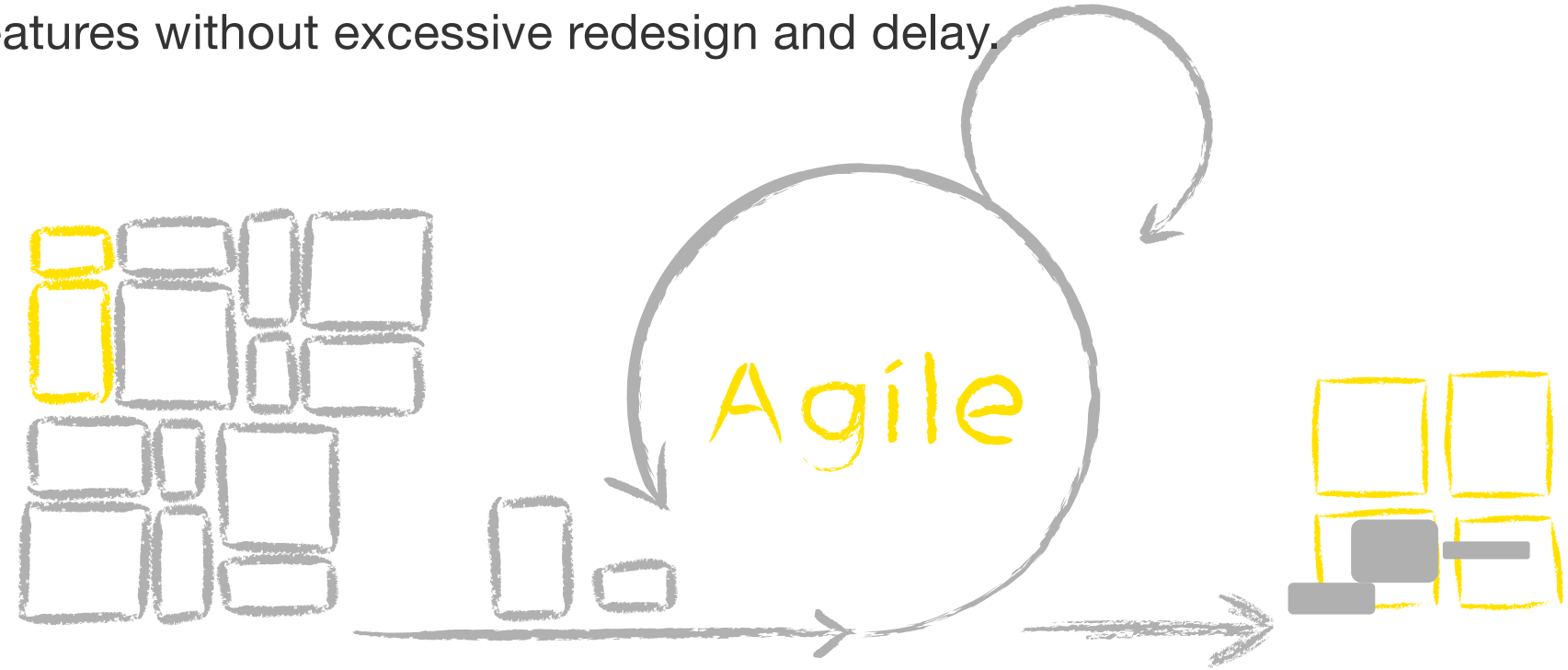
Elaboration – business and use case analysis. domain modeling, slowly shifting to design modeling, **architecture centric**

Construction – less analysis, more design, focus on achieving feature completeness according to construction plan



Iterative & Incremental?

Challening to work iteratively and incrementally with the existing code, components, and technical infrastructure needed to implement near-term features without excessive redesign and delay.

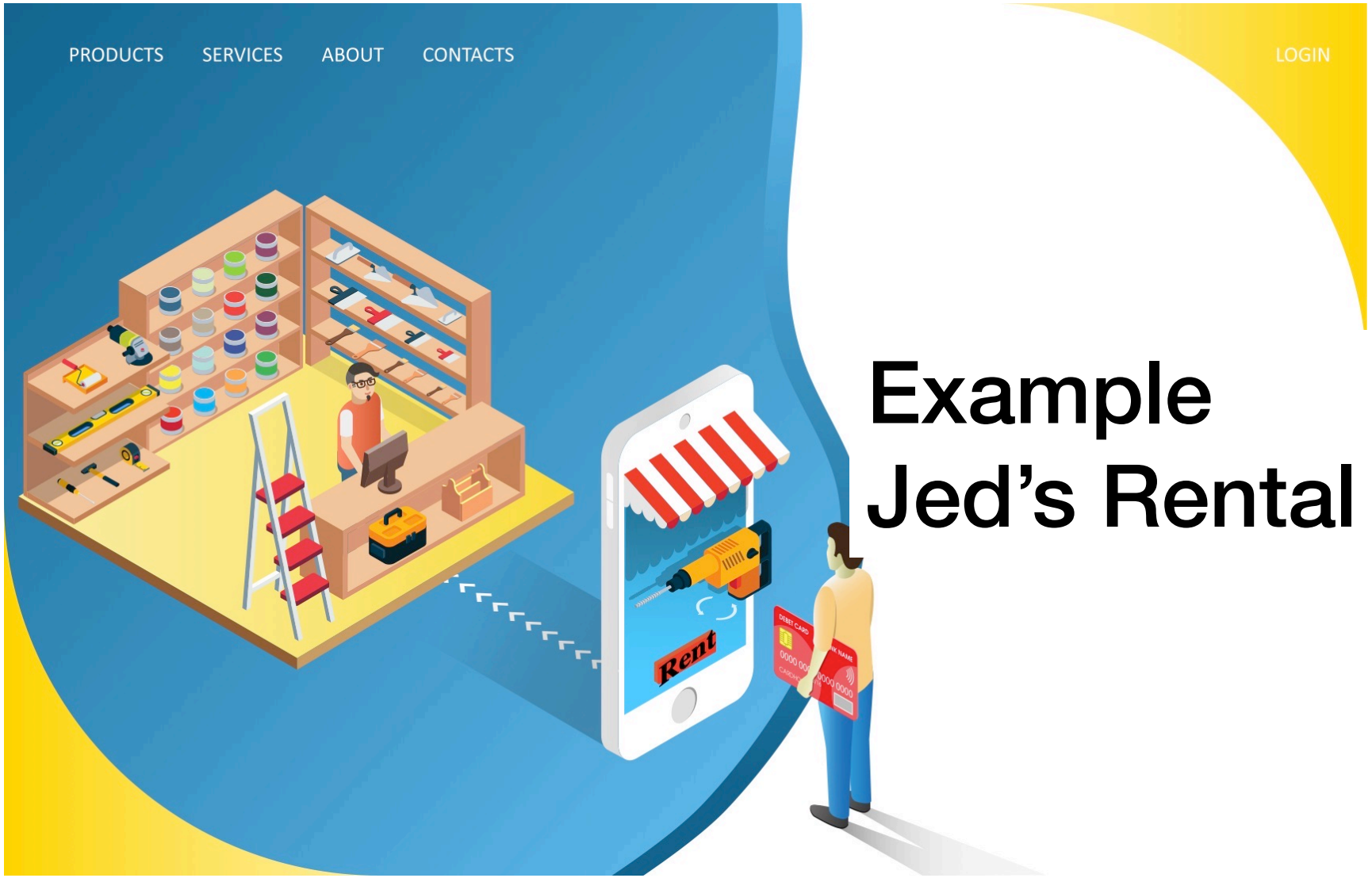


Architectural increments

When and What!

When you should make an architecture significant decision and what that decision is targeting is not always obvious!

Architecture design should be a structured decision activity that addresses its unique challenges.



Example Jed's Rental

Jed's – Architecture Increment

Elaboration - 1st iteration

Design a simple system with a client that connects to a server, reads data from a database, displays the data on the client side, manipulates the data, and sends it back to the server and updates the database!

Customers or Stakeholders

Jed

Judy

Functionality

Security

Performance

Usability

Cost

Lisa

Jed's – the Architecture

More examples of architecture significant decisions!

Functionality

Security

Performance

Usability

Cost

Architecture Significant Decisions

Functionality – how do I allocate the system's functionality?

Security – how do I design a system that is secure?

Performance – I need to design a system that meets these goals too

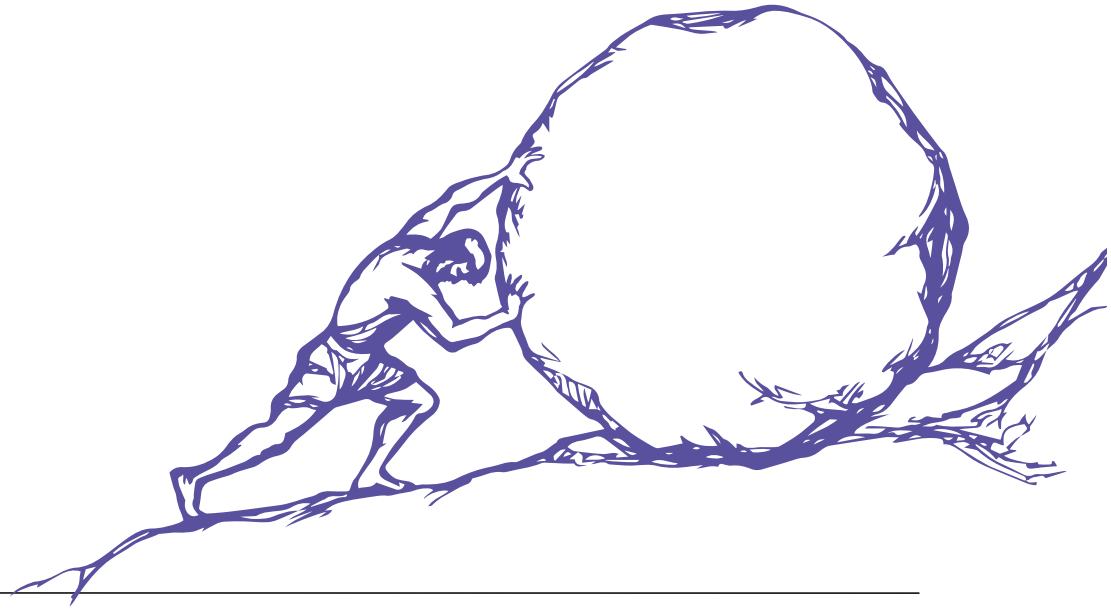
Usability – the system I design must be usable as well!

Cost – and of course we have some cost constraints too.

Software Architecture Design

Architecture is an abstraction level that enables discussions and reasoning among architects, designers, and developers about architecturally significant decisions throughout the software system's life-cycle.

Three principles!



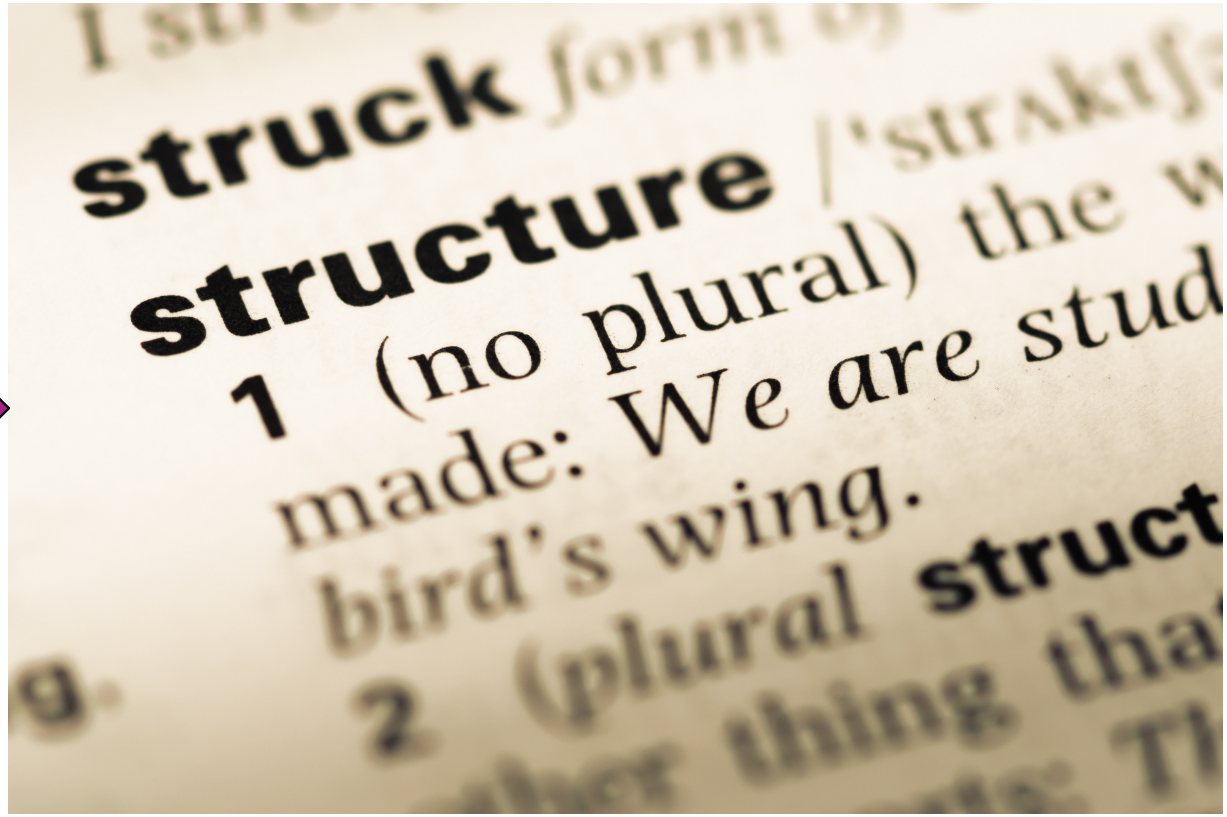
Principle No. 1

Quality as Behavior

Architecture Mechanisms

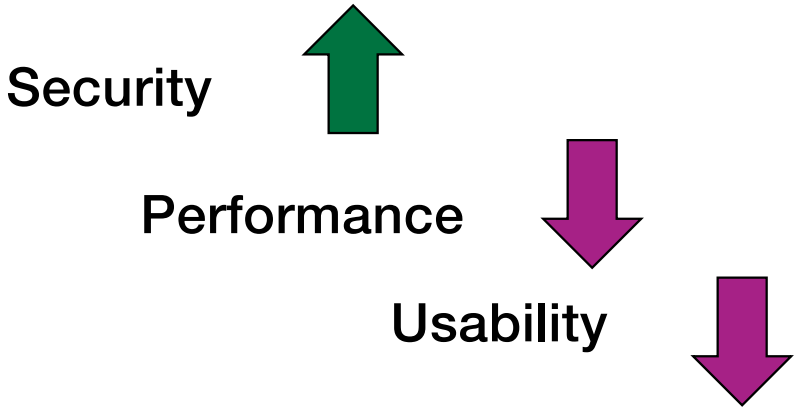
Principle No. 2

Quality as Structure



Principle No. 3

Your architecture decisions enable or inhibit quality

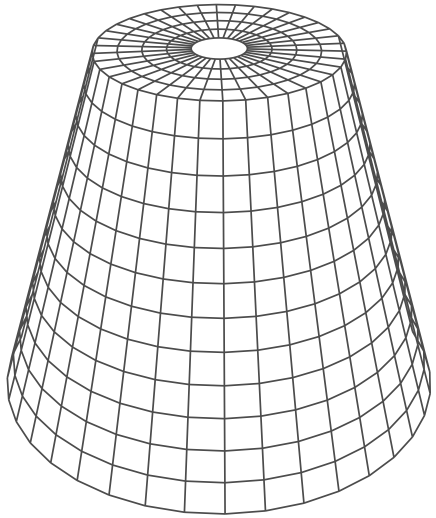


Partitioning & Decomposition

Abstraction, Modularity, Hierarchy & Encapsulation

From Black-box to White box

1. Partition the problem
2. Parts and Responsibilities
3. Divide n’ conquer



Perspectives

1. Layering or distribution
2. Functionality – interfaces
3. Separation of concerns

1st level Decomposition

untangles the complexity

tools for simplification

- abstraction
- modularity
- hierarchy
- encapsulation

Requirements

Responsibilities

Interfaces

Connections

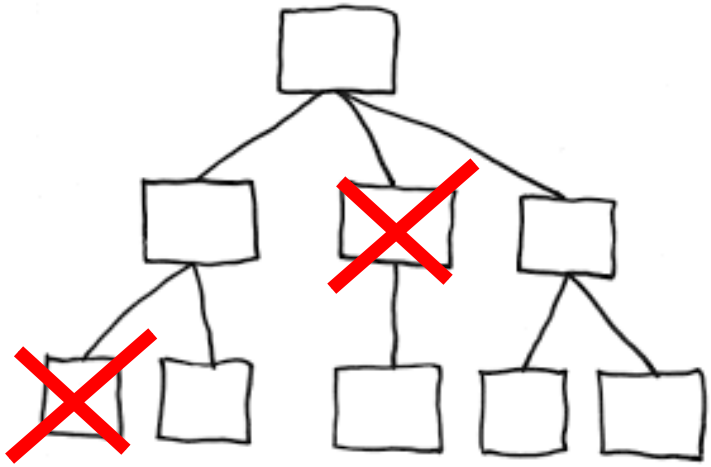
Remember the Magical number 7l

Decomposition

A system consists of subsystems, which contain subsystems, which can be further decomposed in subsystems …

This is true for all systems, biological as well as artificial

Architecture is and hierarchical abstraction

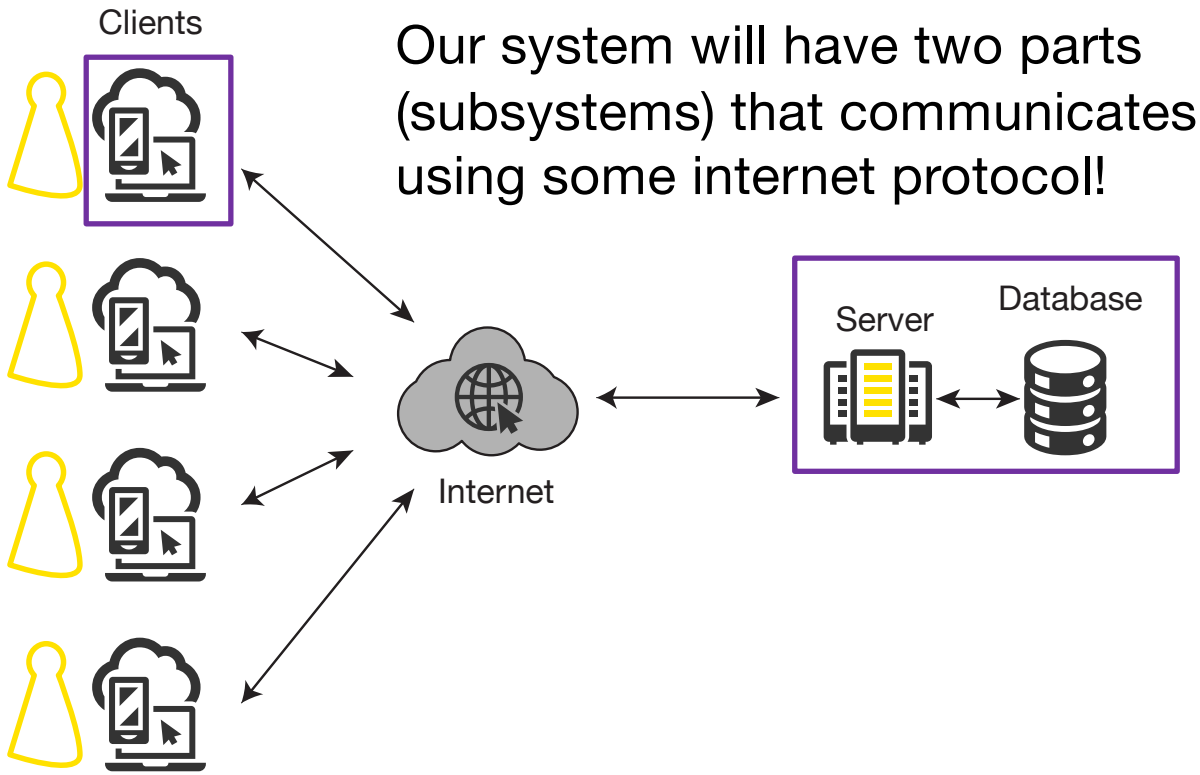


So, problem solving is

Finding the best “pieces” & join them, or?

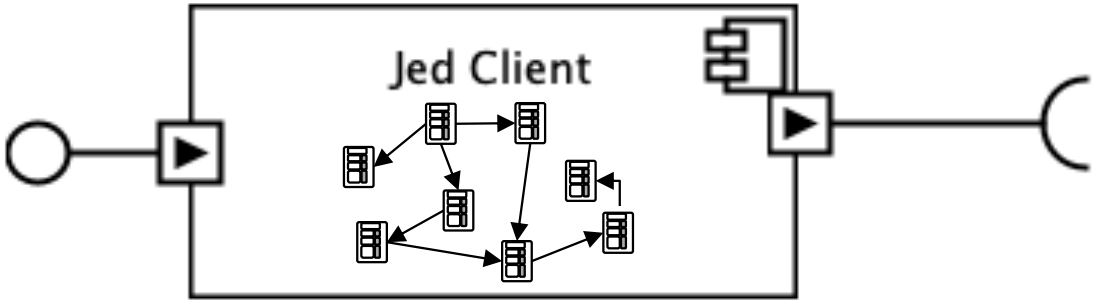
Will the parts fit & function together?

Physical level architecture

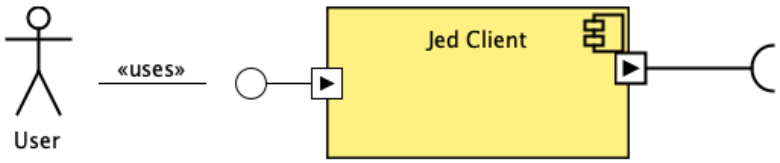


Logical level architecture

We start at the top, with a top-level abstraction – the component.
A component is a ”mini” system with an inner structure that implements the components responsibilities.



Component Diagram



UML Components and interfaces
Represents a logical package with responsibilities. Collaborating objects, defined by classes, provides behaviors to collaborators that require functions through interfaces

UML components are structured classifiers, which means that we can use other classifiers (components, classes) do describe the internal structures that ”realizes” the responsibilities.

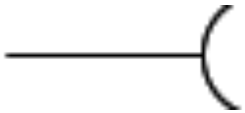
Modularity – Hierarchy – Encapsulation

Component Diagram

Elements

Component with port

Port



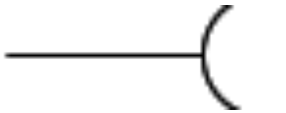
Interface <<requires>>

Interface <<provides>>

Component Diagram

Interfaces

In **UML**, component interfaces describe how components interact with each other. These interfaces are critical in modeling the interactions and dependencies between components in a system. A **component** represents a modular part of a system, while its **interfaces** define the points of interaction.



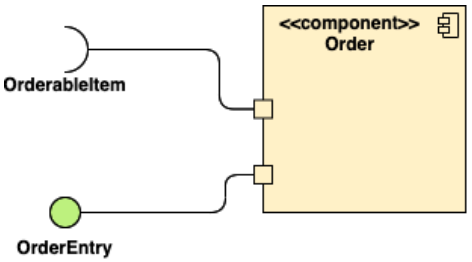
Interface <<requires>>

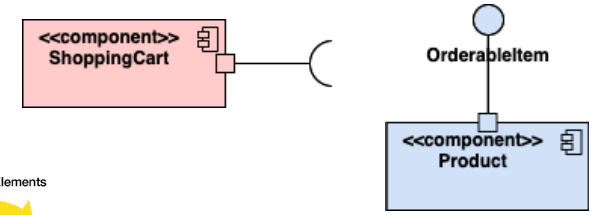
Interface <<provides>>

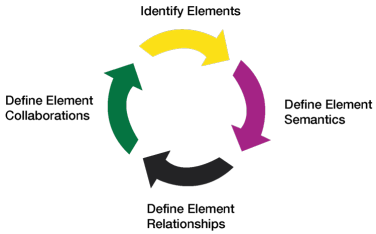
Component Diagram

Example – Functional decomposition Architecture level

First Step Basic Abstractions





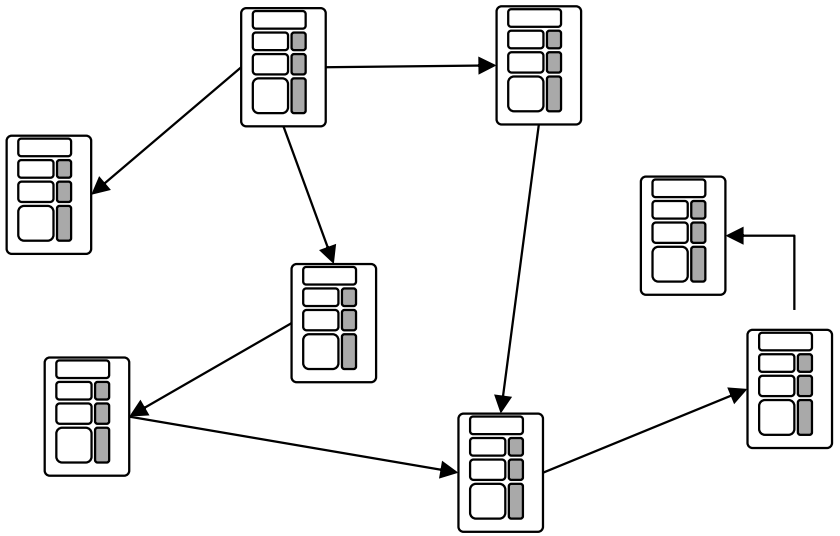


Add responsibilities

Add Interfaces

Describe responsibilities

Describe Interfaces



Responsibility driven design

Object abstractions

Responsibility Driven Design

Responsibility-driven design (RDD) focus on identifying and assigning responsibilities to different classes within a system.

- Brainstorming technique
- “Develop a system with classes”

Class – The name of the class being modeled. Represents a potential object in your system.

Responsibility – A list of the main responsibilities or functionalities of the class. Defines what the class is responsible for “doing”, primarily focusing on the actions (methods) it can perform and what the is “knowing”.

Collaborator – Other classes that the class needs to interact, that is, classes that a class interacts with to fulfill its responsibilities.

Responsibilities

- Knowing - What an object knows about itself or other objects. It could include attributes and data that the class of objects manages.
- Doing - This includes the actions or operations an object can perform. On its own or together with other objects. These responsibilities define the behavior of a class of objects.

A simple library system.

How you might define responsibilities:

Student objects:

- Knowing: Student name, student ID, list of borrowed books.
- Doing: Borrow a book, return a book, check borrowing history.

Library objects:

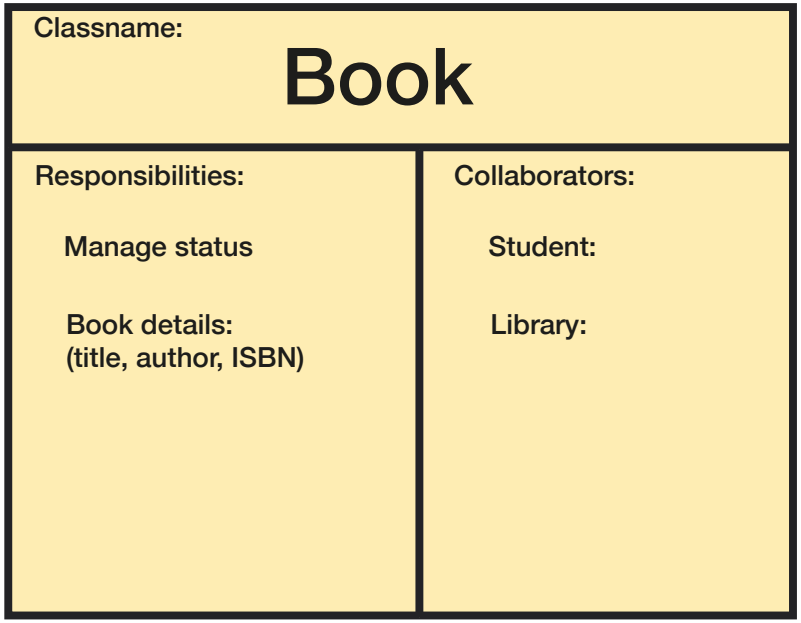
- Knowing: List of books, list of members.
- Doing: Add or remove a book, register a new student, manage lending.

Book objects:

- Knowing: Title, author, ISBN, current status [available, checked out].
- Doing: Update status, provide book details.

Creating a CRC card

The classname, at the top



The class's collaborators.

Help to carry out each responsibility, in the right column

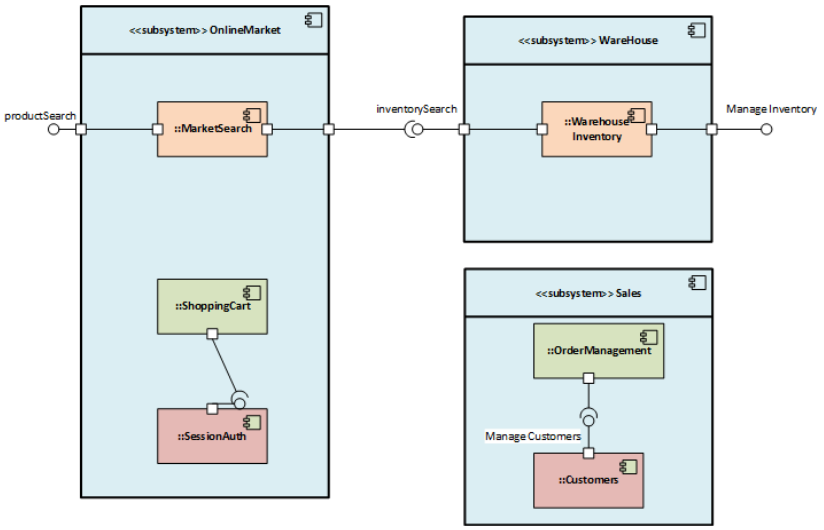
The class's responsibilities

Responsibilities

Yes, its Knowing and Doing

Allocate Responsibilities to the Abstractions

- Functionality – Interfaces
- Services



Black box to White-box

– Start at the highest of levels

– Refine responsibilities iteratively

Document your Components in
Component – Responsibilities – Collaborators cards

Documenting Interfaces

Use the Collaborators section or separate specification

Purpose – Briefly describe the role of the component and the purpose of its interfaces.

Provided Interfaces – List the services or operations the component provides.

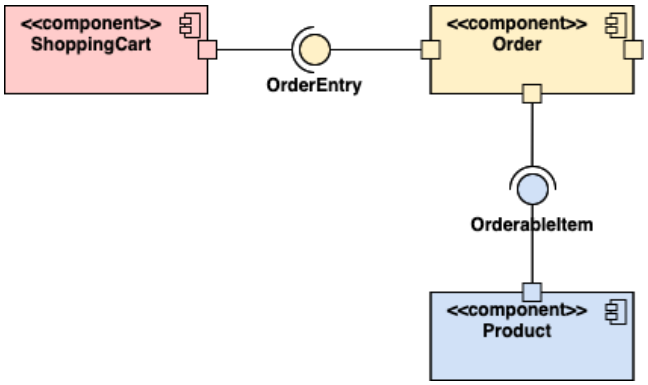
Required Interfaces – List the services or operations the component depends on.

Interactions – Document the connections between components through interfaces.

Component Diagram

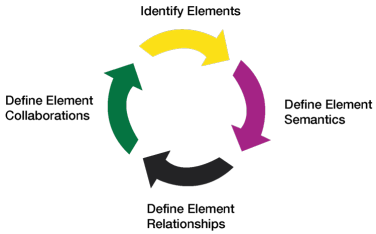
Example

Second Step Connect Abstractions



Connect interfaces

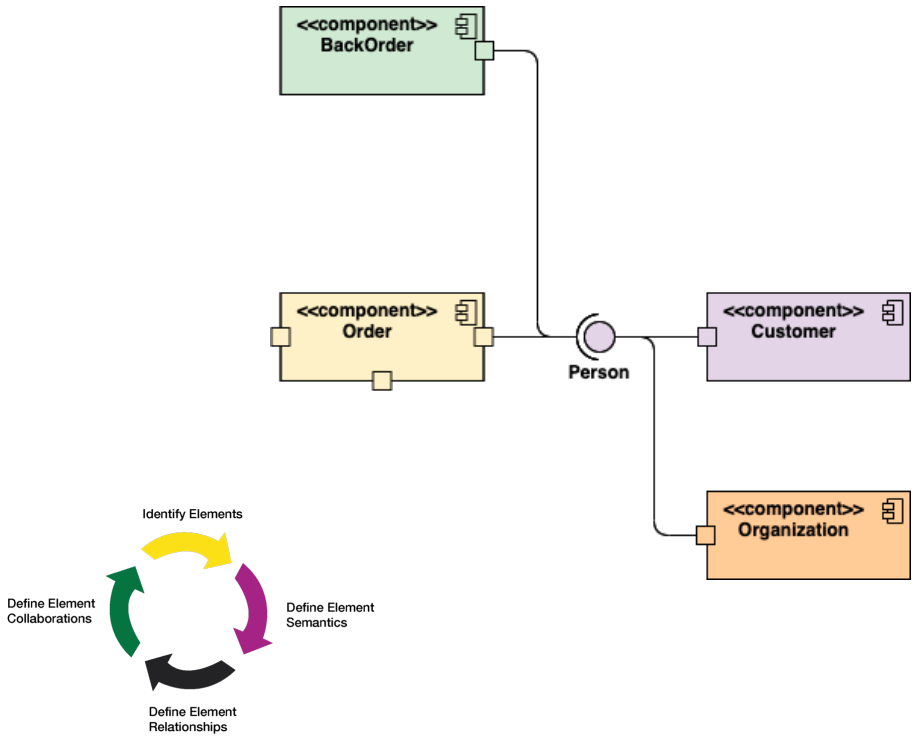
Integrity



Component Diagram

Example

Third Step Extend model



N-level decomposition

Add and refine components

Add and refine responsibilities

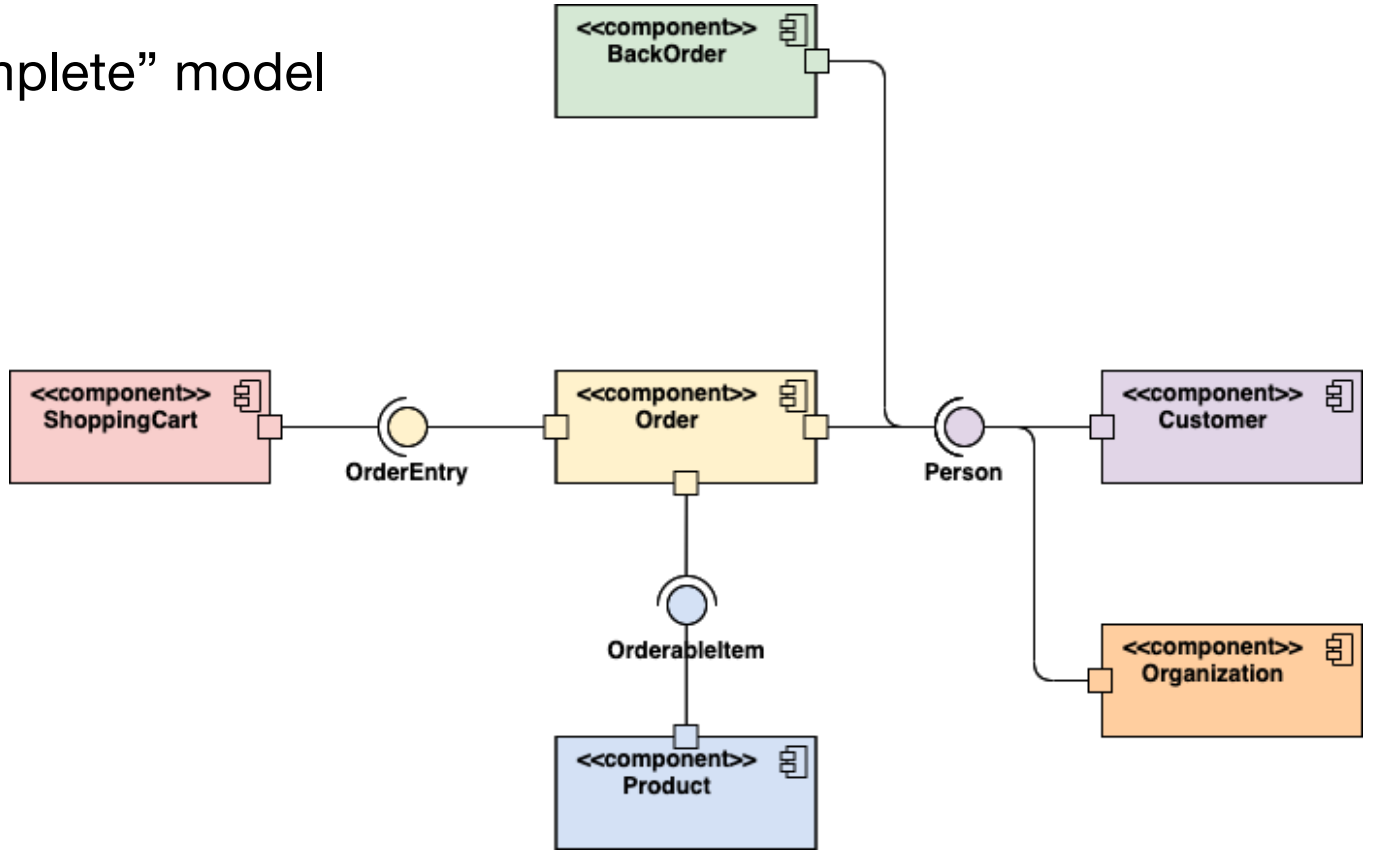
Add and refine interfaces

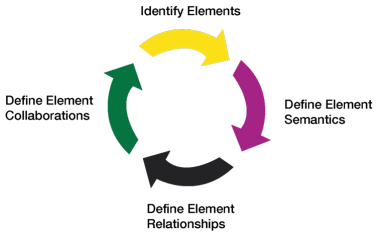
Add and refine connections

Component Diagram

Example

Fourth Step “Complete” model

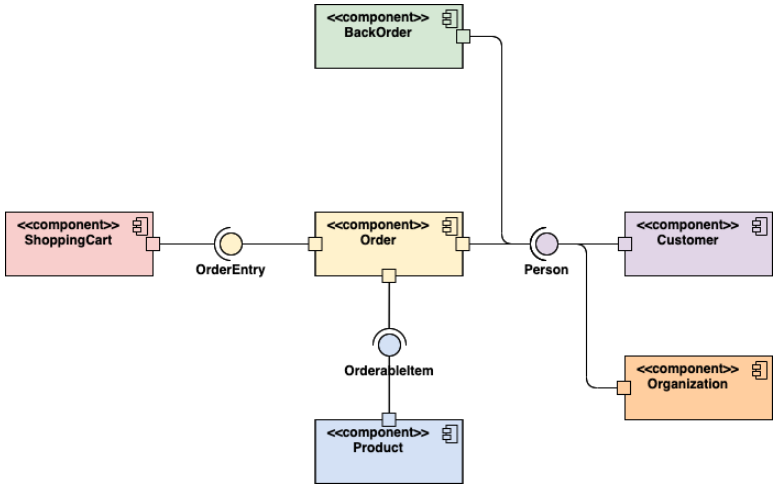




Component Diagrams

Can describe patterns – quality as structure!

Can describe mechanisms – quality as behavior!



Architecture design

From requirements to an OO model via design concepts and components

---

Linnæus University

A simple library system.

We talked about functional requirements & responsibilities:

Student objects:

- Knowing: Student name, student ID, list of borrowed books.
- Doing: Borrow a book, return a book, check borrowing history.

Library objects:

- Knowing: List of books, list of members.
- Doing: Add or remove a book, register a new student, manage lending.

Book objects:

- Knowing: Title, author, ISBN, current status [available, checked out].
- Doing: Update status, provide book details.

Library – more Requirements

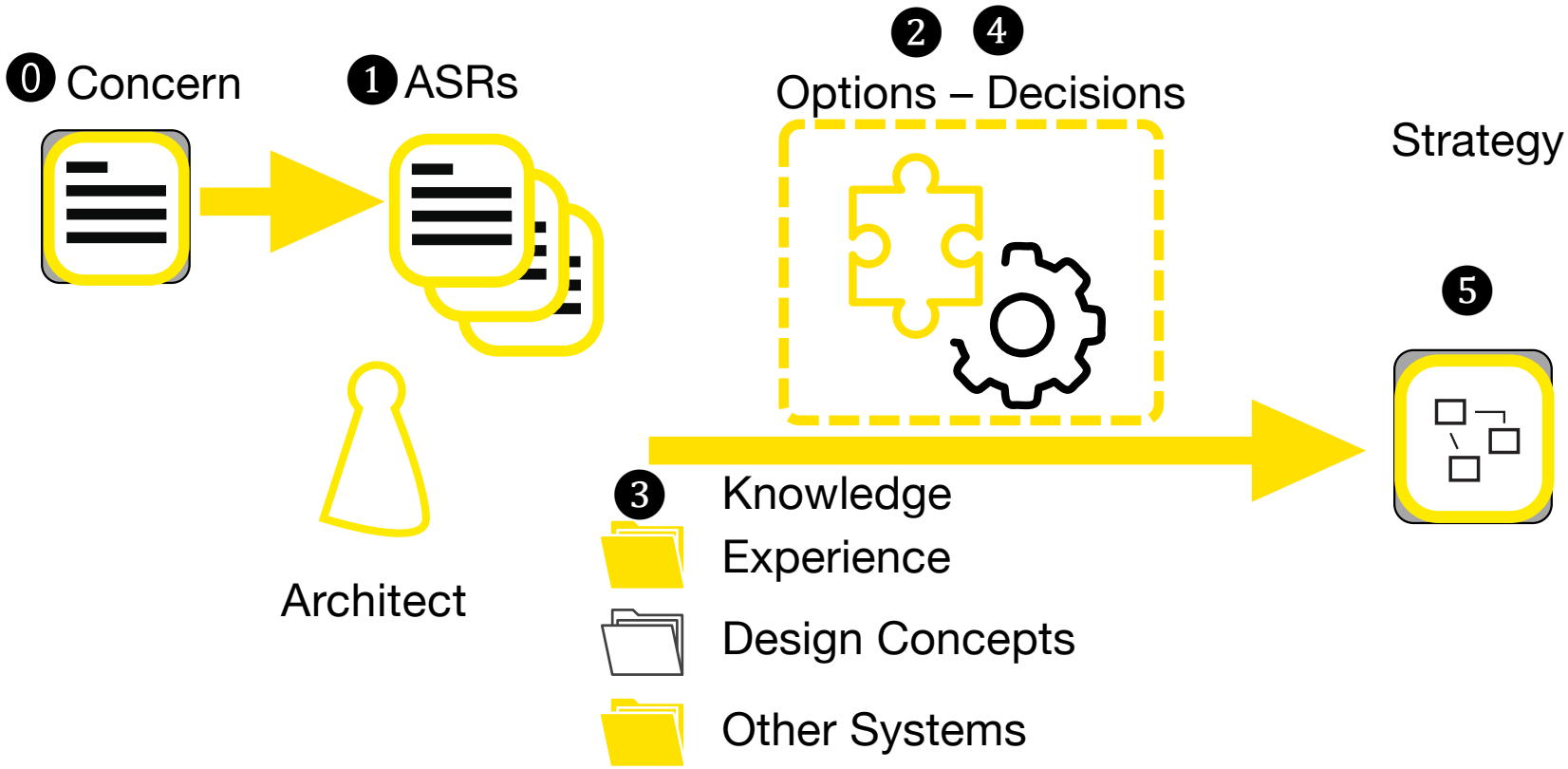
Make sure we get no prank reservations!

We would like to monitor important events in the system and store them somewhere else just in case.

I'm concerned with all the information we send to the system. It's important that it is not used for any other purpose!

What is this?

Architecture Design



Decomposition – a Concern

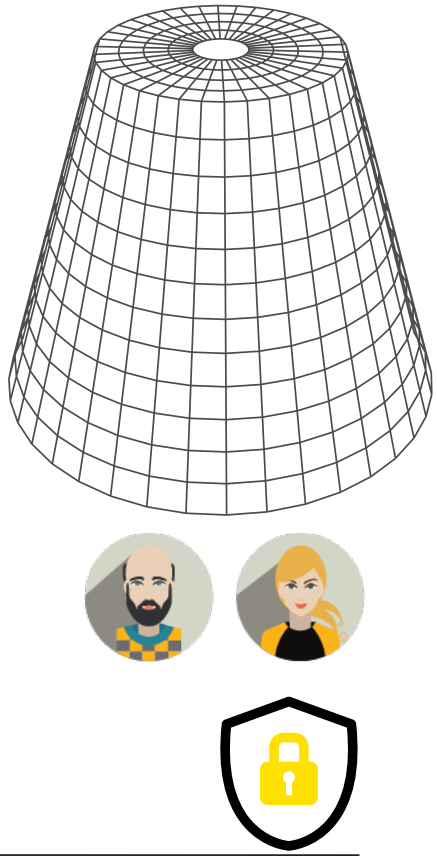
Design the Security Strategy

Understand design concepts to use

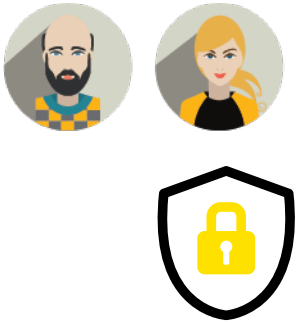
Allocate responsibilities to components

Specify <<provides>> & <<requires>> interfaces

Document the Components and Interfaces



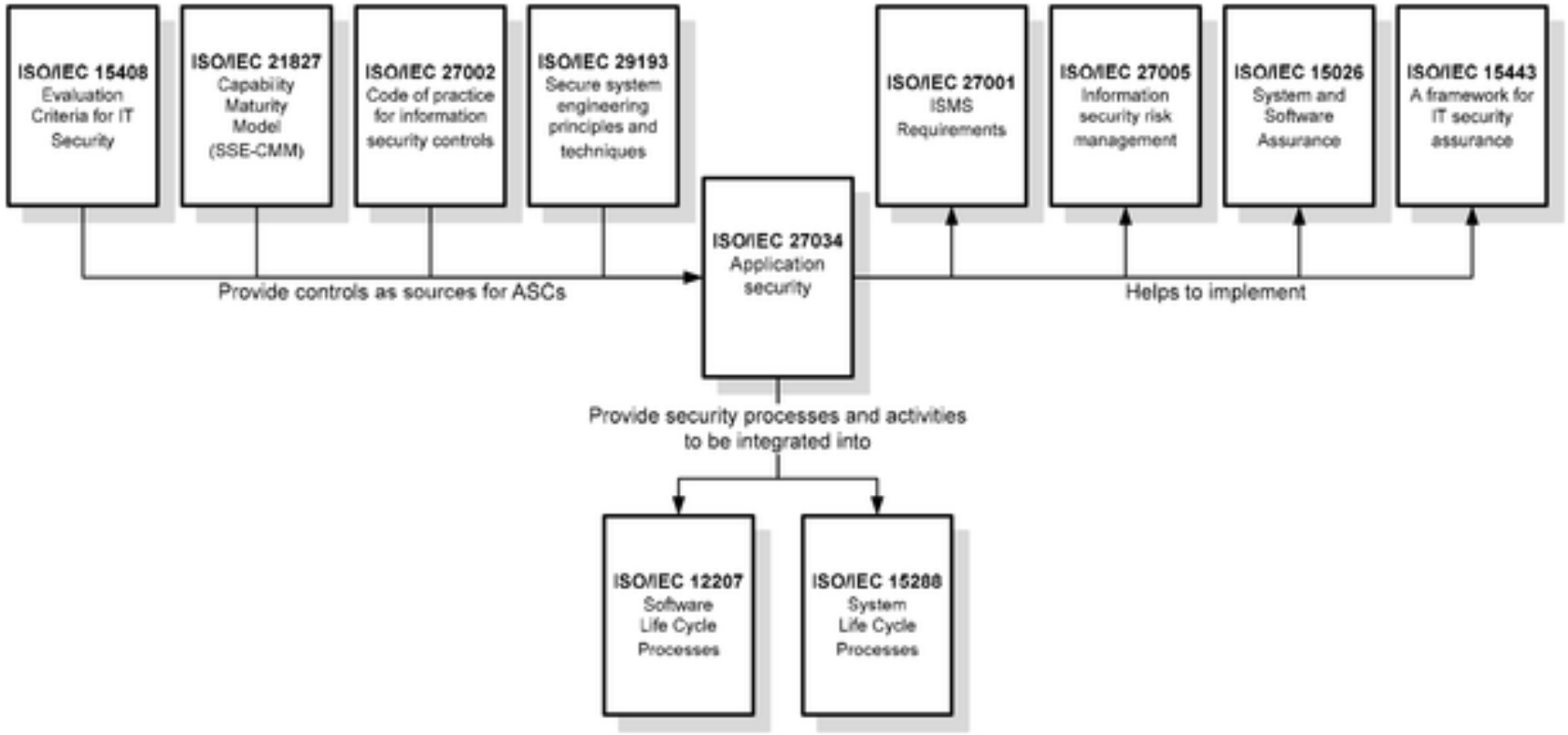
Architecture & Security!



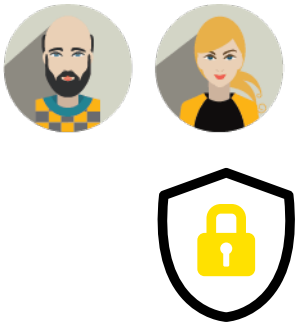
Three steps to prepare the architecture increment

- Understand the security requirements
- Select design concepts to use
- Design the security strategy and integrate

Security ISO/IEC 27034



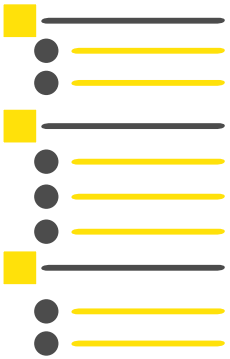
Some Security Objectives



Ensure that users and client applications are identified and that their identities are properly verified.
Ensure that users and client applications can only access data and services for which they have been properly authorized.
Detect attempted intrusions by unauthorized persons and client applications.
Ensure that unauthorized malicious programs (e.g., viruses) do not infect the application or component.
Ensure that communications and data are not intentionally corrupted.
Ensure that parties to interactions with the application or component cannot laterrepudiate those interactions.
Ensure that confidential communications and data are kept private.
Enable security personnel to audit the status and usage of the security mechanisms.
Ensure that applications and centers survive attack, possibly in degraded mode.
Ensure that centers and their components and personnel are protected against destruction, damage, theft, or surreptitious replacement (e.g., due to vandalism, sabotage, or terrorism).
Ensure that system maintenance does not unintentionally disrupt the security mechanisms of the application, component, or center.

Donald Firesmith: Engineering Security Requirements, in Journal of Object Technology, vol. 2, no. 1, January-February 2003, pages 53-68.

Security Requirements



- Identification Requirements
- Authentication Requirements
- Authorization Requirements
- Immunity Requirements
- Integrity Requirements
- Intrusion Detection Requirements
- Nonrepudiation Requirements
- Privacy Requirements
- Security Auditing Requirements
- Survivability Requirements
- Physical Protection Requirements
- System Maintenance Security Requirements

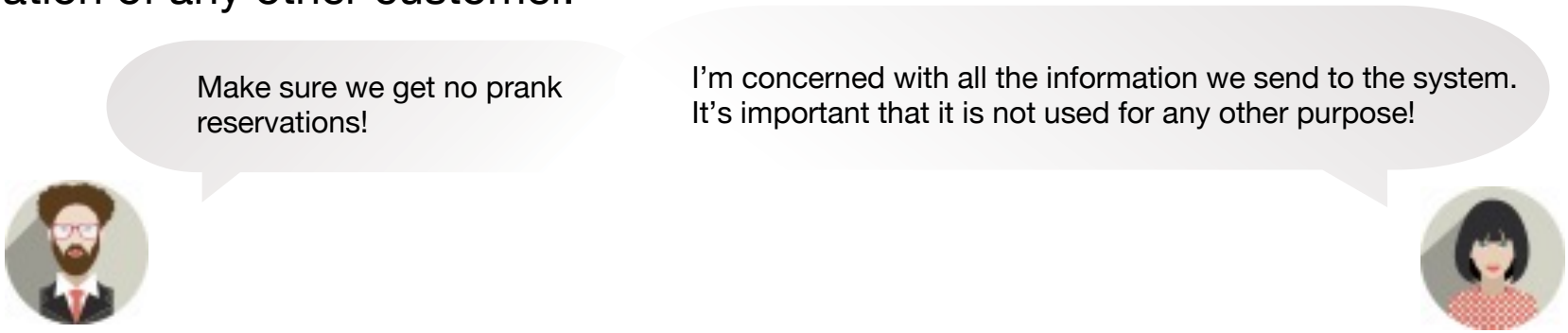
Library – Security Requirements

Security Policy – “The application shall allow each customer to obtain access to all of his or her own personal account information.”

Identification - “The application shall identify all of its human users beforeallowing them to use its capabilities.”

Authentication - “The application shall verify the identity of all of its users before allowing them to use its capabilities.”

Authorization - “The application shall **NOT** allow any customer to access any account information of any other customer.”

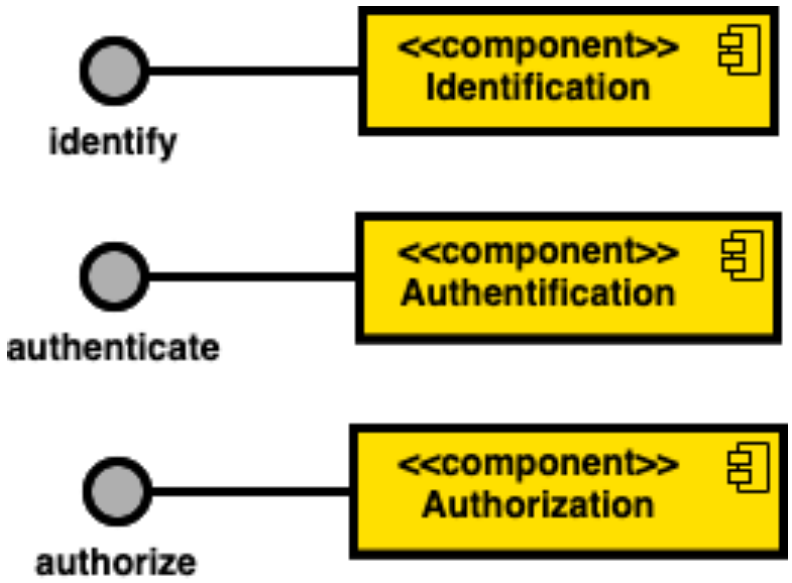


Architecture Mechanism

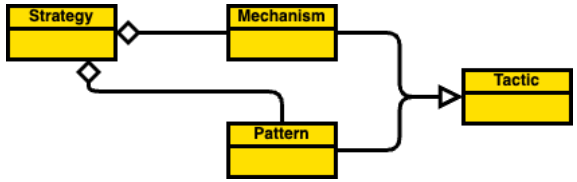
Secure access to functionality and data

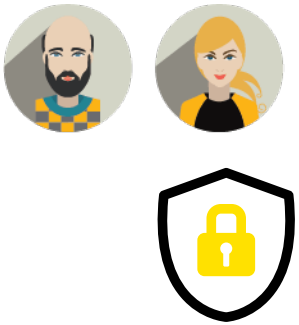
Security ASRs

- Identification Requirements
- Authentication Requirements
- Authorization Requirements



Next step Design Concepts





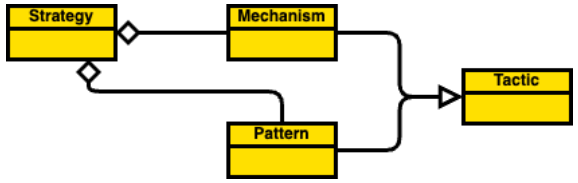
Options and alternatives ➡️ Discussion and Reasoning

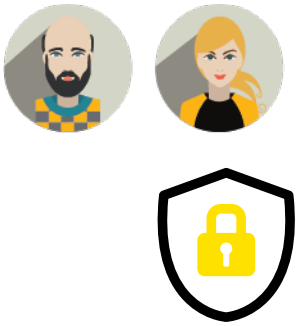
Examples of design concepts

- tactics, patterns, and mechanisms
- reference architectures,
- externally developed components and services

For each type, many options exist!

Concepts – Two questions





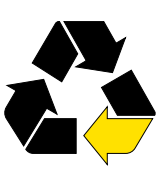
Should we do it ourselves?

Inhouse design or development

Should we use external components?

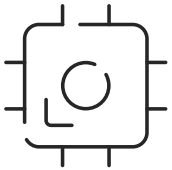
Inhouse adoption and development

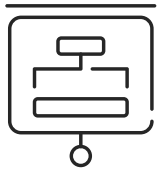
Do it inhouse or reuse?

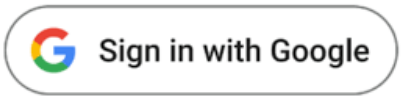




Service


Platform

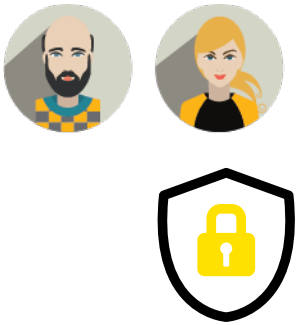

Framework

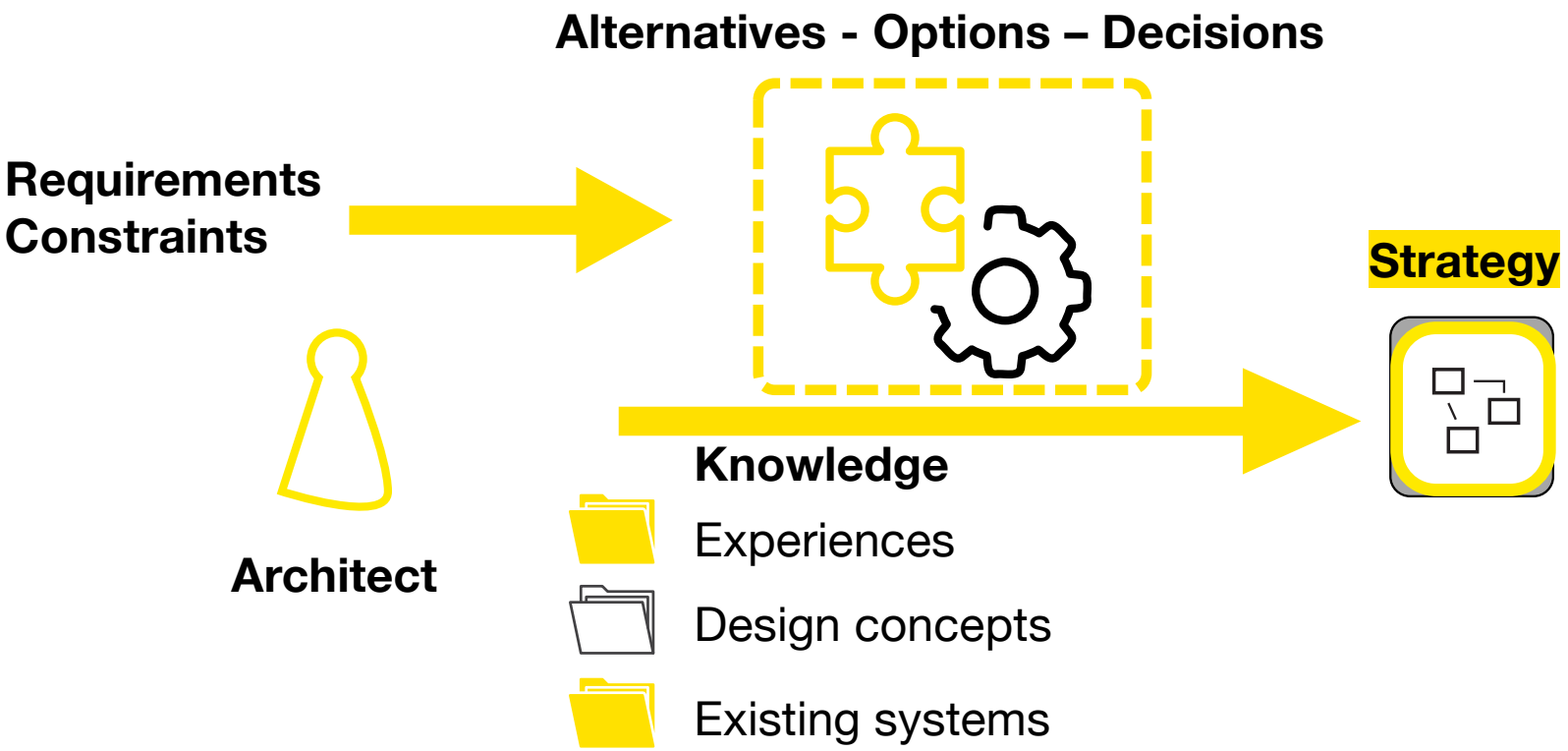



JAAS

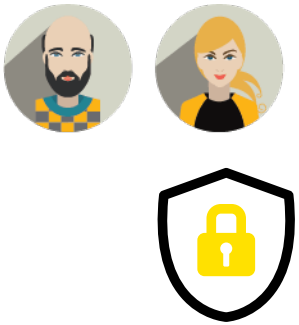
Internally or externally developed components?

Architecture Design

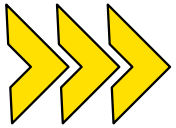




Architecture Strategy

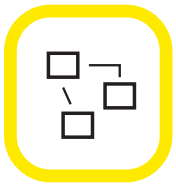


For a single quality



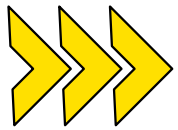
Design the security mechanism and integrate

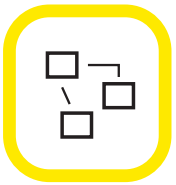
Strategy

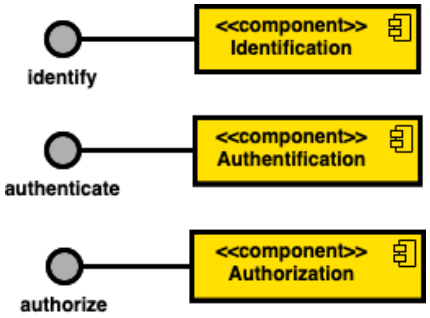


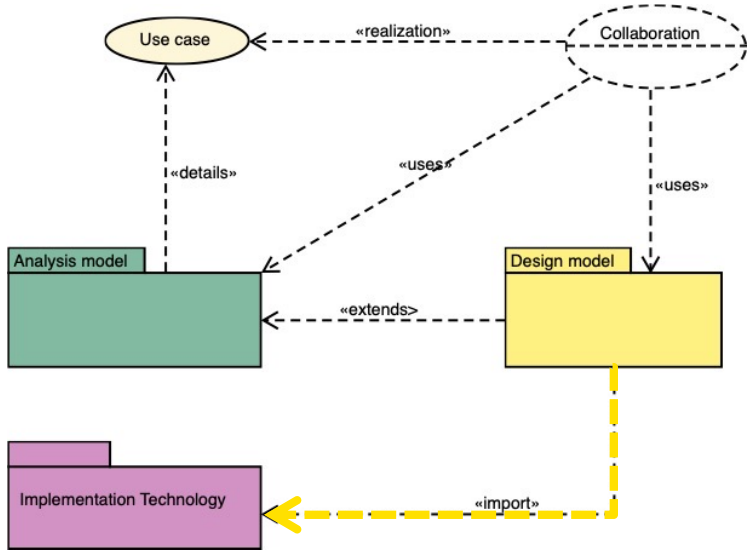
Behavior and structure that must be designed, implemented, and integrated in the system

Our Security Strategy









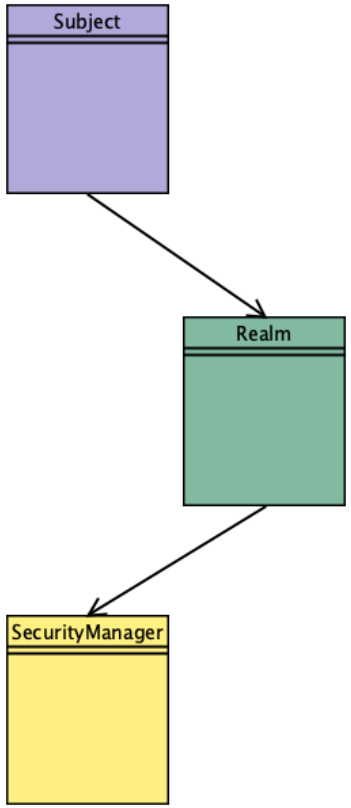
Externally developed component

Shiro - a framework that provides reusable classes

We use Shiro classes as implementation technology in our design model

Shiro Classes in our system

Implementation technology



Object-oriented design is a method of design encompassing the process of object-oriented decomposition and a notation for depicting both logical and physical as well as static and dynamic models of the system under design.

Object-oriented programming is a method of implementation in which programs are organized as cooperative collections of objects, each of which represents an instance of some class, and whose classes are all members of a hierarchy of classes united via inheritance relationships.

```
UsernamePasswordToken token = new UsernamePasswordToken( username: "jesper",  password: "1dv607");
try {
    currentUser.login(token);
    log.info("User successfully authenticated!");
} catch (AuthenticationException ae) {
    log.error("Authentication failed: " + ae.getMessage());
    System.exit( status: 1);
}
```

Shiro @work!

Today’s takeaways

Some design decisions have more impact than others

Architecture is design at a high level

- components,
- interfaces
- responsibilities

2DV604 – Software Architecture

Lnu.se