جامعة الملك سعود
King Saud University

College of Computer & Information Sciences

Software Design and Development

Chapter 6 and 7 (Sommerville)

Fall Semester 2021
1st Semester 1443 H

Topics

- What is SW design?
- SW design levels
- Architecture design
- High- level design
- Component-level design

The 4 Fundamental SW Process Activities

Software Specification (Requirements Engineering)

Customers and engineers **define** the software that is to be produced and the **constraints** on its operation.

Software Design & Development

The software is **designed** and **programmed**.

Software Validation

The software is **checked** to ensure that it is what the customer requires.

Software Evolution

The software is **modified** to reflect changing customer and market requirements

Software Design and Implementation

- Software design and implementation is the stage in the software engineering process at which an **executable software system is developed**.
- Software design and implementation activities are invariably **inter-leaved**.
- **Software design** is a creative activity in which you identify software components and their relationships, based on a customer's requirements.
- The design process moves from a "big picture" view of software to a more narrow view that defines the detail required to implement a system.
- **Implementation** is the process of realizing the design as a program.

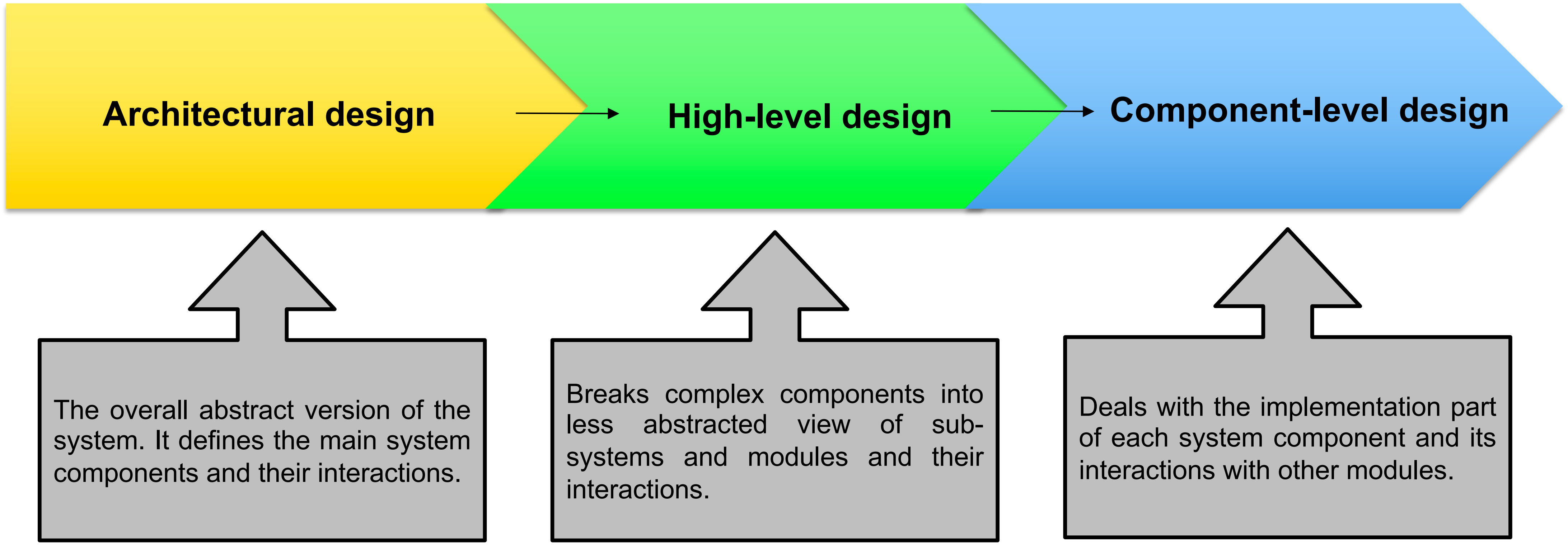
Build or Buy

- In a wide range of domains, it is now possible to buy **off-the-shelf systems (COTS)** that can be adapted and tailored to the users' requirements.

- For example, if you want to implement a medical records system, you can buy a package that is already used in hospitals. It can be cheaper and faster to use this approach rather than developing a system in a conventional programming language.

- When you develop an application in this way, the design process becomes concerned with how to use the configuration features of that system to deliver the system requirements.

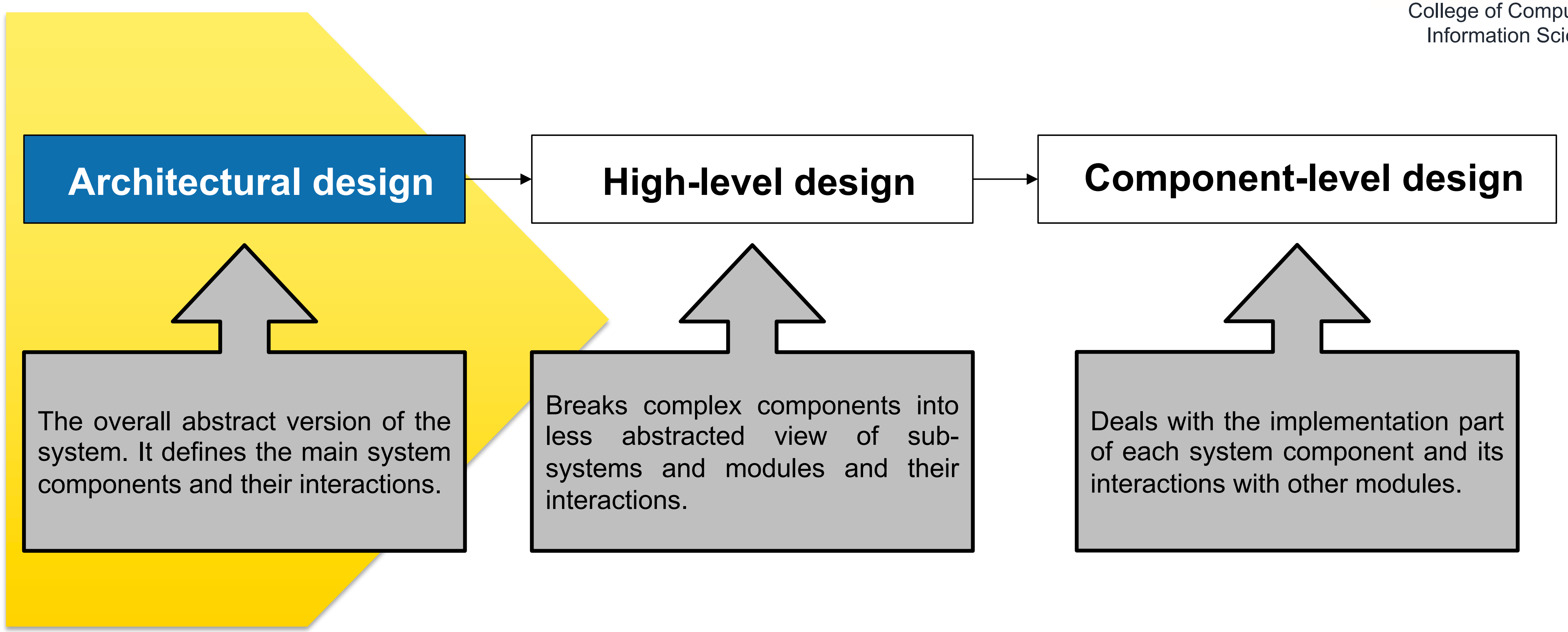
Software Design Levels



The overall abstract version of the system. It defines the main system components and their interactions.

Breaks complex components into less abstracted view of sub-systems and modules and their interactions.

Deals with the implementation part of each system component and its interactions with other modules.



Architectural design → High-level design → Component-level design

Architectural design: The overall abstract version of the system. It defines the main system components and their interactions.

High-level design: Breaks complex components into less abstracted view of sub-systems and modules and their interactions.

Component-level design: Deals with the implementation part of each system component and its interactions with other modules.

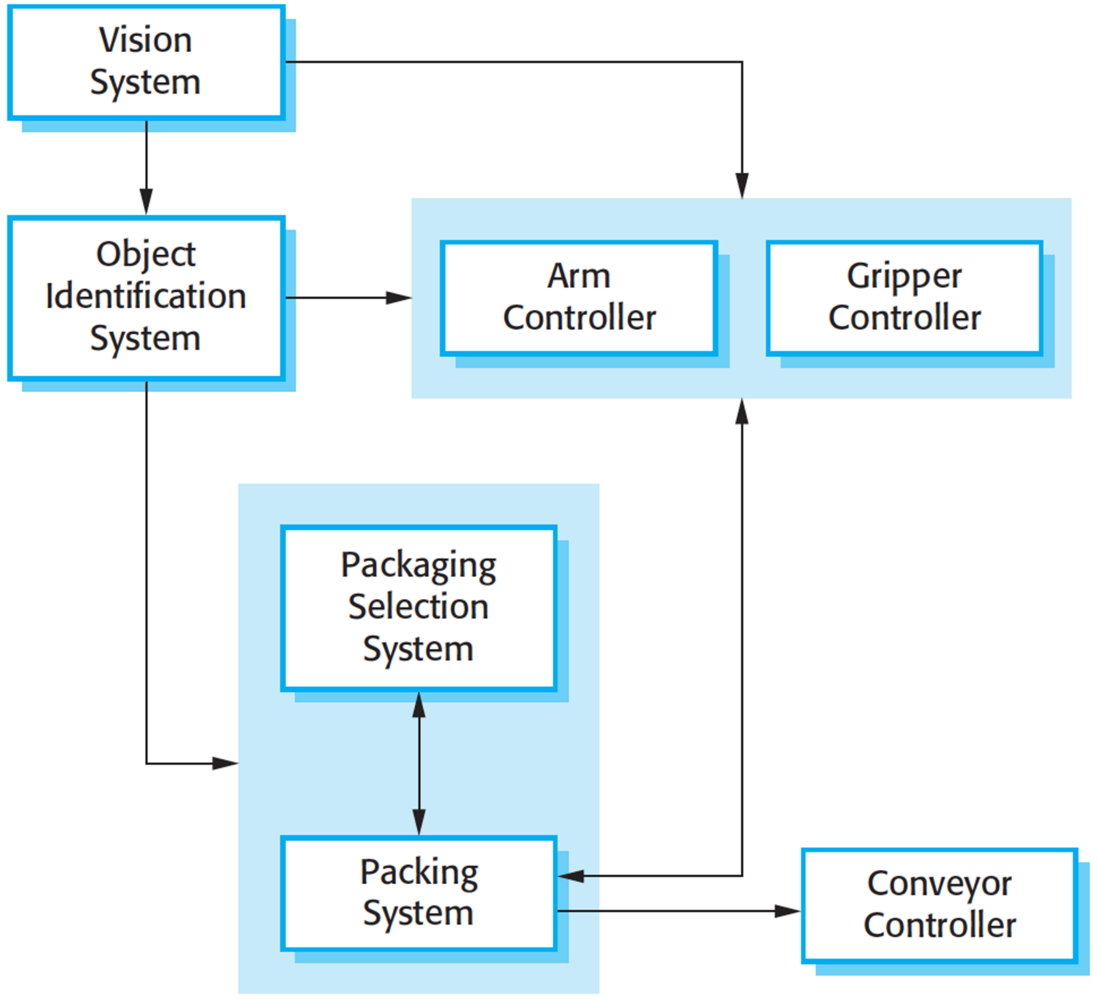
Architectural Design

- **Architectural design** is concerned with understanding how a software system **should be organized** and designing the **overall structure** of that system.
- Architectural design is the critical **link** between **design** and **requirements** engineering, as it identifies the **main structural components** in a system and their **relationships**.
- The **output** of the architectural design process is an **architectural model** that describes how the system is organized as a set of communicating components.

Architectural Design

This robotic system can pack different kinds of object.

- ➢ It uses a vision component to pick out objects on a conveyor.
- ➢ It identifies the type of object and selects the right kind of packaging.
- ➢ The system then moves objects from the delivery conveyor to be packaged.
- ➢ It places packaged objects on another conveyor.



Architecture of a packing robot control system

Agility and Architecture

- It is generally accepted that an early stage of agile processes is to design an overall systems architecture.
- While refactoring components in response to changes is usually relatively easy, **refactoring a system architecture is likely to be expensive.**

Advantages of Explicit Architecture

- Stakeholder communication
  - Architecture may be used as a focus of discussion by system stakeholders.
- System analysis
  - Means that analysis of whether the system can meet its non-functional requirements is possible.
- Large-scale reuse
  - The architecture may be reusable across a range of systems.
  - Product-line architectures may be developed.

Architectural Design Decisions

Is there a generic application architecture that can act as a template for the system that is being designed?

How will the system be distributed across a number of cores or processors?

What architectural patterns or styles might be used?

What will be the fundamental approach used to structure the system?

How will the structural components in the system be decomposed into subcomponents?

What architectural organization is best for delivering the non-functional requirements of the system?

Architecture and Non-Functional Requirements

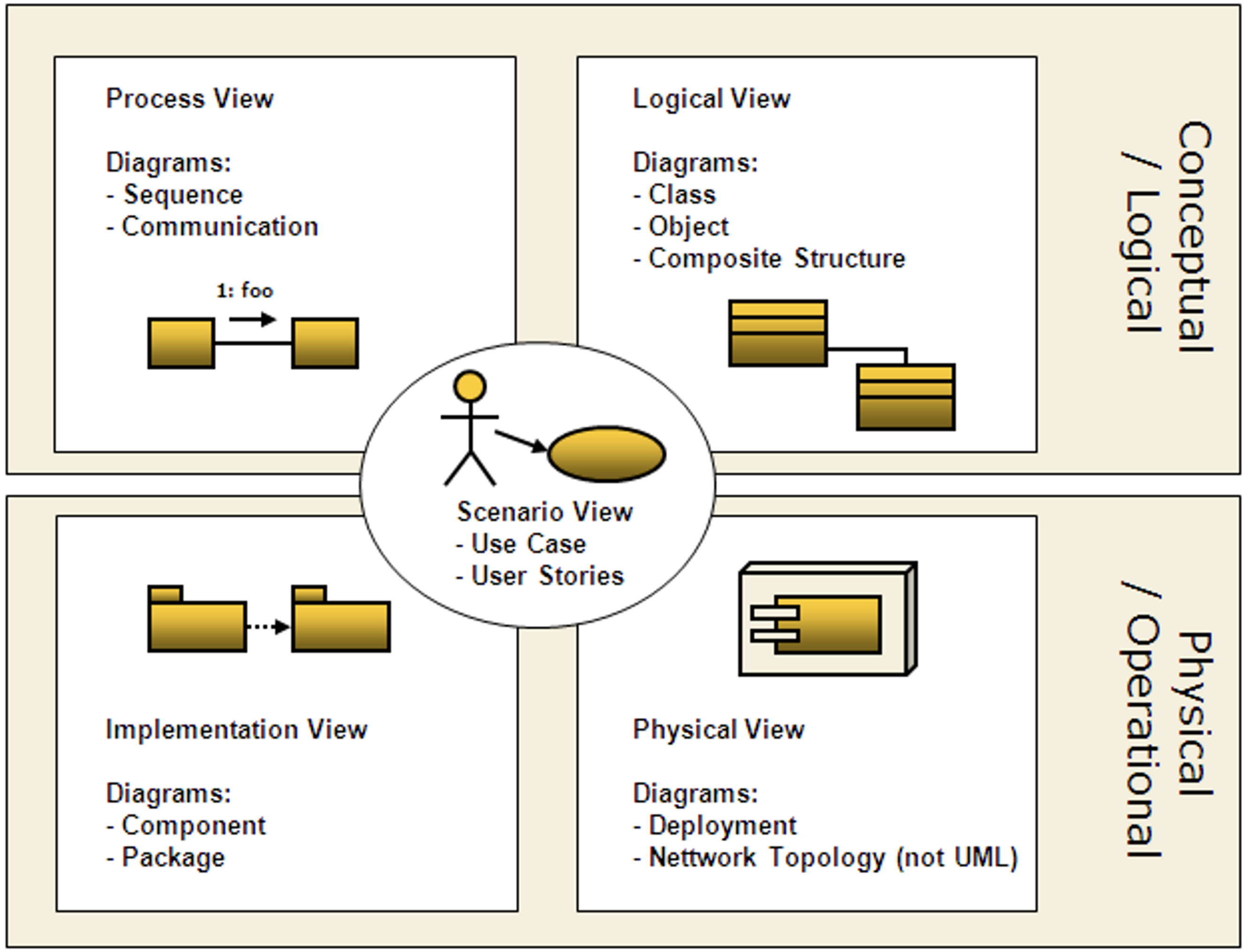
Non-Functional Requirement	To achieve it…
Performance	Localize critical operations and minimize communications. Use large rather than fine-grain components.
Security	Use a layered architecture with critical assets in the inner layers.
Safety	Localize safety-critical features in a small number of sub- systems
Availability	Include redundant components and mechanisms for fault tolerance.
Maintainability	Use fine-grain, replaceable components.

Architectural Views

- What views (perspectives) are useful when designing and documenting a system's architecture?
  - ➢ For both design and documentation, you usually need to present multiple views of the software architecture.

- Each architectural model shows one view of the system. For example,
  - ➢ How a system is decomposed into modules.
  - ➢ How the run-time processes interact.
  - ➢ The different ways in which system components are distributed across a network.

4+1 Architectural Views



- Process View
- Logical View
- Implementation View
- Physical View
- Scenario View

Why 4 + 1 instead of just 5?
Because all other views use the scenarios view to guide them.

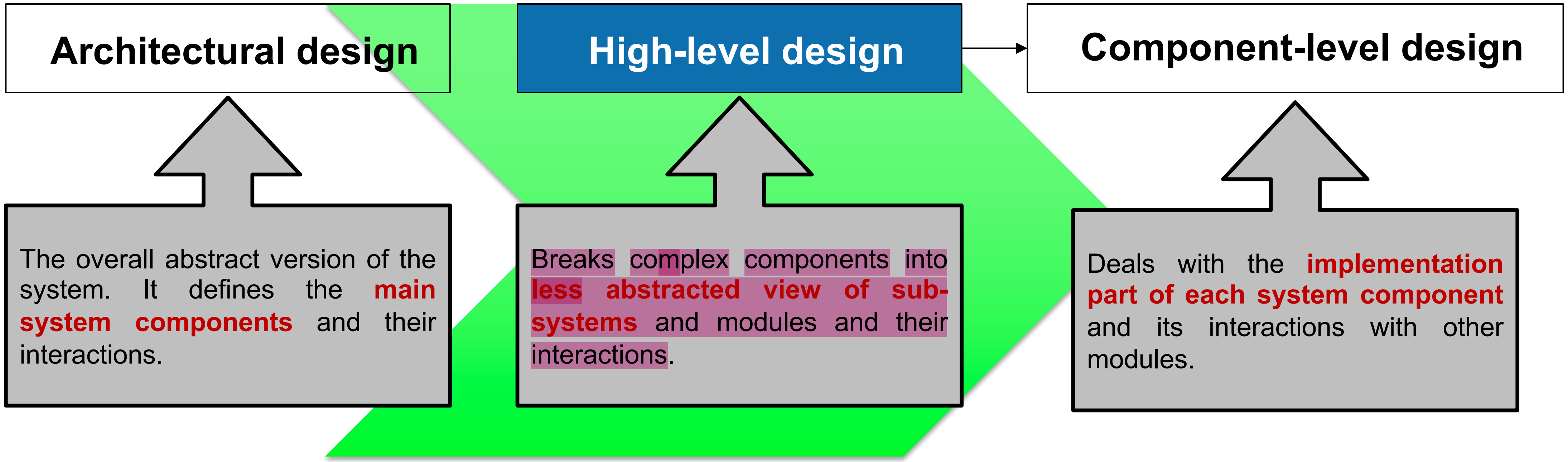
https://wiki.cantara.no/display/dev/4+plus+1+View+Model

4 + 1 View Model of Software Architecture

View	Description	Focus	Relevant to	(Possible) UML
Logical view	Shows the key abstractions in the system as classes	Functional requirements	End-user, analyst, designer.	Class diagram
Process view	Shows how, at run-time, the system is composed of interacting processes.	Non-functional requirements	Developer, integrator.	Activity diagram
Development view	Shows how the software is decomposed for development.	Software module organization (Software management reuse & tool constraints)	Programmer and project manager	Component diagram
Physical view	Shows the system hardware and how software components are distributed across the processors in the system	Non-functional requirements (related to hardware)	System engineer, operator, system administrator, system installer	Deployment diagram
Use cases or scenarios	Shows the various usage scenarios.	Functional & non-functional requirements	All	Use case diagram

Architectural Views for Agile Methods

- Detailed design documentation is mostly unused. It is, therefore, a waste of time and money to develop it.
- For most systems, it is not worth developing a detailed architectural description from these four perspectives.
- You should develop the views that are useful for communication and not worry about whether or not your architectural documentation is complete.
- However, an exception to this is when you are developing critical systems, when you need to make a detailed dependability analysis of the system.



Architectural design

The overall abstract version of the system. It defines the **main system components** and their interactions.

High-level design

Breaks complex components into **less abstracted view of sub-systems** and modules and their interactions.

Component-level design

Deals with the **implementation part of each system component** and its interactions with other modules.

Object-Oriented Design Process

- Object-oriented design processes involve developing a number of different system models.
  - ➢ They require a lot of effort for development and maintenance and, for **small systems**, this may not be cost-effective.

  - ➢ However, for **large systems** developed by different groups design models are an important communication mechanism.

Process Stages in OO Design

- There are a variety of different object-oriented design processes that depend on the organization.

- Common activities in these processes include:
  - ➢ Define the context and interaction with other systems.
  - ➢ Design the system architecture.
  - ➢ Identify the principal system objects.
  - ➢ Develop design models.
  - ➢ Specify object interfaces.

System Context and Interactions

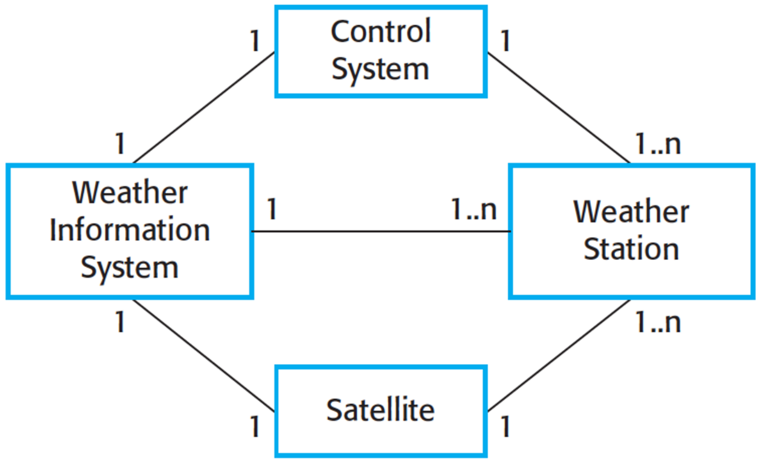
King Saud University
College of Computer and Information Sciences

- Understanding the relationships between the **software that is being designed and its external environment.**
  - Essential for deciding how to provide the required system functionality and how to structure the system to communicate with its environment.

- System context lets you establish the **system boundaries**. Setting the system boundaries helps you decide what features are implemented in the system being designed and what features are not.

- A **system context model** is a structural model that demonstrates the other systems in the environment of the system being developed.

- An **interaction model** is a dynamic model that shows how the system interacts with its environment as it is used.



System context model for the weather station

For system interaction models, read Section 7.1.1

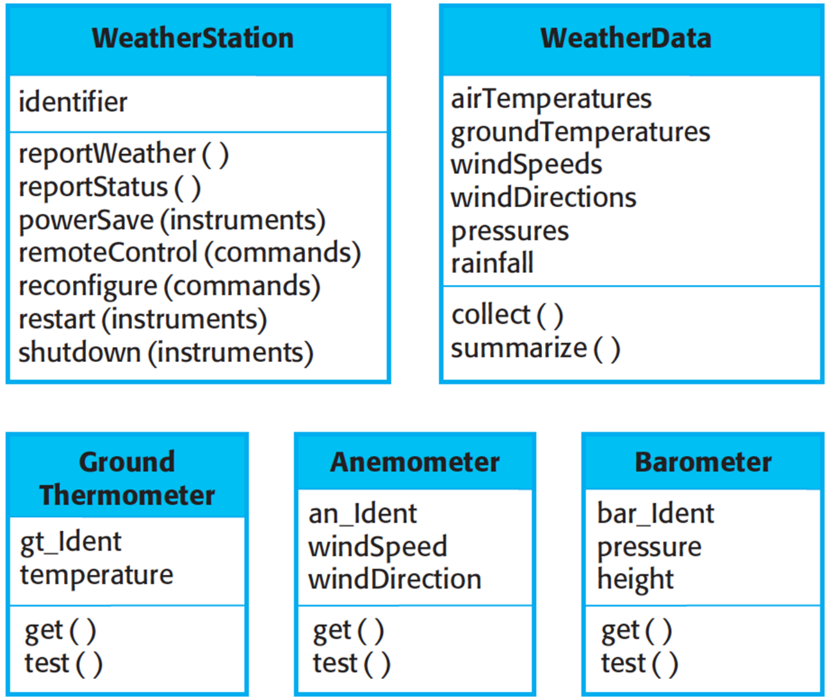
Identify the Principal System Objects

College of Computer and Information Sciences

- Identifying object classes is often a **difficult part** of object-oriented design.
  - There is no 'magic formula' for object identification.
  - It **relies on the skill, experience and domain knowledge** of system designers.
  - Object identification is an **iterative process**. You are unlikely to get it right first time.
- Approaches to identification:
  - Use a grammatical approach based on a natural language description of the system.
  - Base the identification on tangible things in the application domain.
  - Use a behavioral approach and identify objects based on what participates in what behavior.
  - Use a scenario-based analysis. The objects, attributes and methods in each scenario are identified.

Weather Station Object Classes

- Object class identification in the weather station system may be based on the tangible hardware and data in the system:
  - **Ground thermometer, Anemometer, Barometer**
    - Application domain objects that are 'hardware' objects related to the instruments in the system.
  - **Weather station**
    - The basic interface of the weather station to its environment. It therefore reflects the interactions identified in the use-case model.
  - **Weather data**
    - Encapsulates the summarized data from the instruments.



Weather station objects

Design Models

- Design models show the objects and object classes and relationships between these entities.
- Two kinds of design model:
  - **Structural models** describe the static structure of the system in terms of classes and relationships.
  - **Dynamic models** describe the dynamic interactions between objects.

- Examples:
  - **Subsystem models** that show logical groupings of objects into subsystems.
  - **Sequence models** that show the sequence of object interactions.
  - **State machine models** that show how individual objects change their state in response to events.
  - **Other models** include use-case models, aggregation models, generalization models, etc.

More in Section 7.1.4

Architectural design → **High-level design** → **Component-level design**

Architectural design: The overall abstract version of the system. It defines the **main system components** and their interactions.

High-level design: Breaks complex components into **less abstracted view of sub-systems** and modules and their interactions.

Component-level design: Deals with the **implementation part of each system component** and its interactions with other modules.

Software Design

Component-level design

Deals with the **implementation part of each system component** and its interactions with other modules.

- Finish adding all class diagram details (attributes & methods).
- Decide data types and method return types.
- Decide all method details using one or more of the following:
  - Algorithm.
  - Flowchart.
  - Pseudo code.
  - Activity diagram.
  - Swimlane diagram.
  - State diagram.
  - Decision table.
  - …

Each component needs to be ready for implementation after this step.

Thank you!