XJTLU Xi'an Jiaotong-Liverpool University 西交利物浦大学

CPT 203

Week 8

Design Concepts

智能工程学院
西 交 利 物 浦 大 学

Learning Objectives

1.Design in the Software Engineering (SE) context

2. Quality control in the design process

3. Design concepts

4. Elements in design model

1.1 Design in the SE context - The Basics

Software Design emcompasses the set of

- **principles**
- **concepts**
- **practices**

that lead to the development of a high-quality system or product.

The goal of design is to produce a model or representation that exhibits:

- **Firmness (no bugs)**
- **Commodity (useful or valuable)**
- **Delight (pleasurable experience)**

1.2a Design in the SE context - From Requirement to Design

Software design is:

- The last software engineering action within the modelling activity and sets the stage for construction (code generation and testing).
- A process to transform user requirements into some suitable form, which helps the programmer in software coding and implementation.

Requirement -> Design -> Coding

- In practice requirements and design are **interrelated**. In particular, working on the design often clarifies the requirements.

Design example 1 （more details in Week 9 software design)

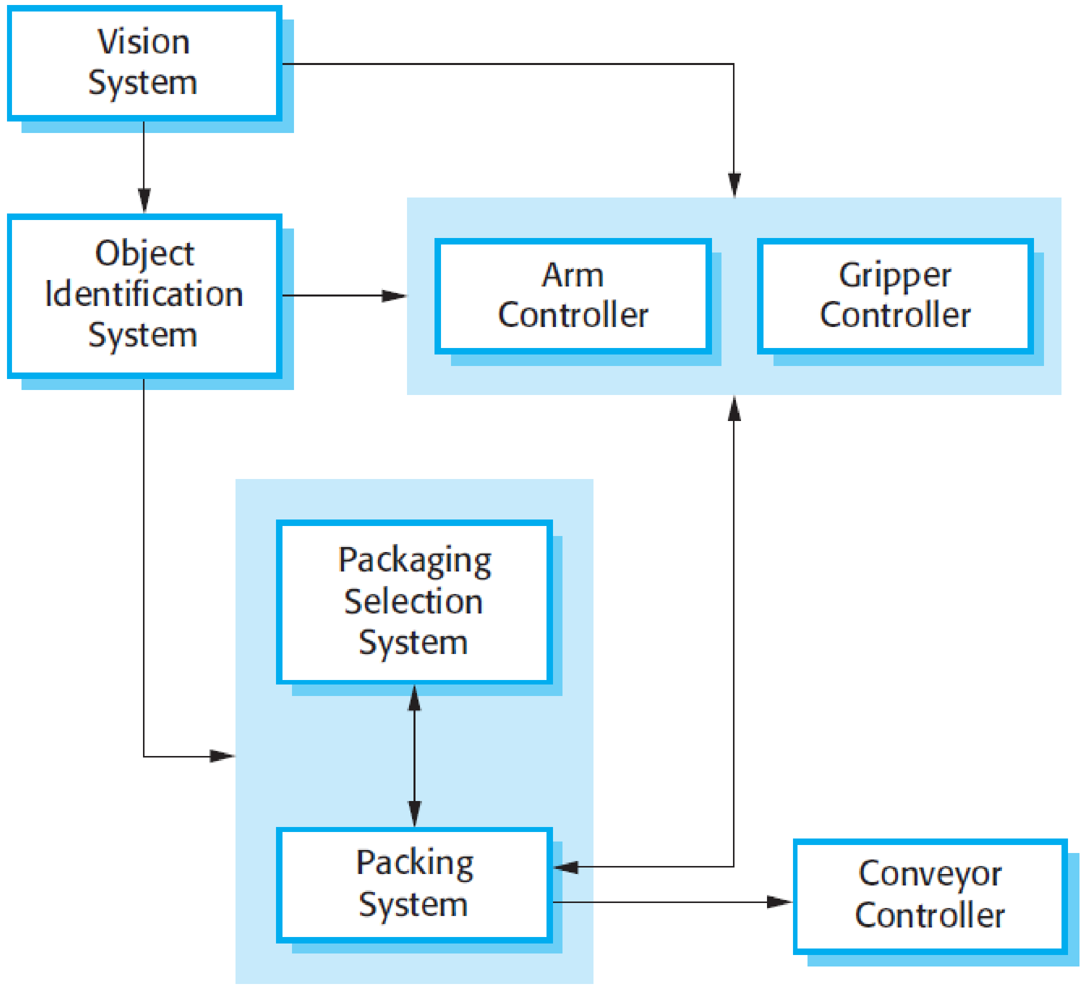
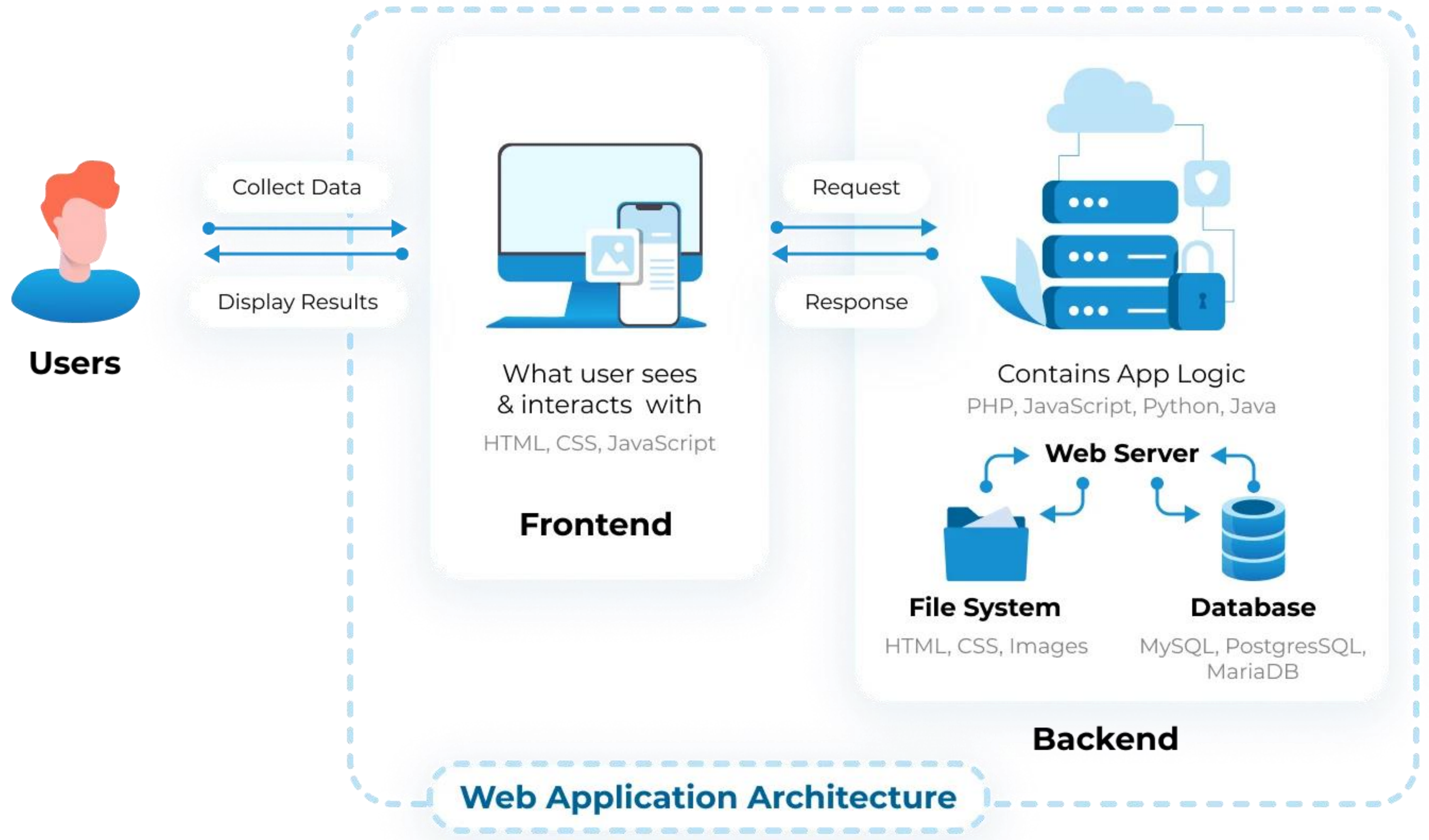


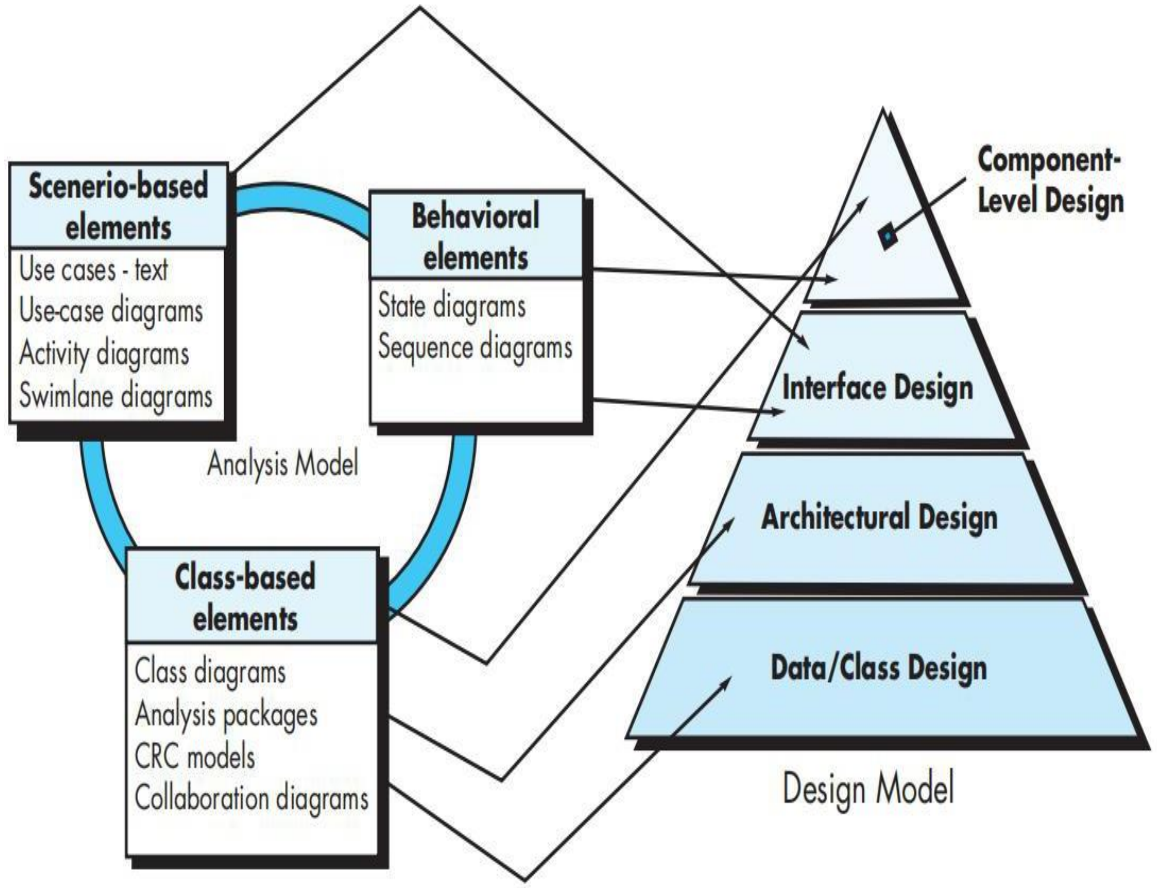
Figure 6.1 The architecture of a packing robot control system

Design example 2



https://litslink.com/blog/web-application-architecture

1.2b Design in the SE context - From Requirement to Design

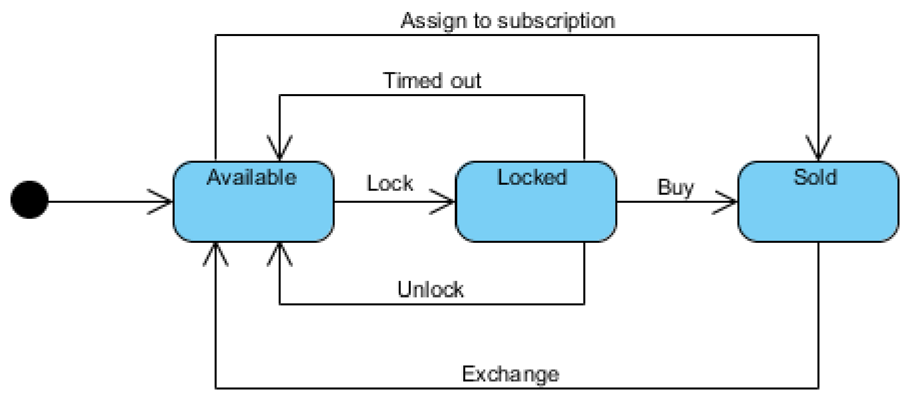


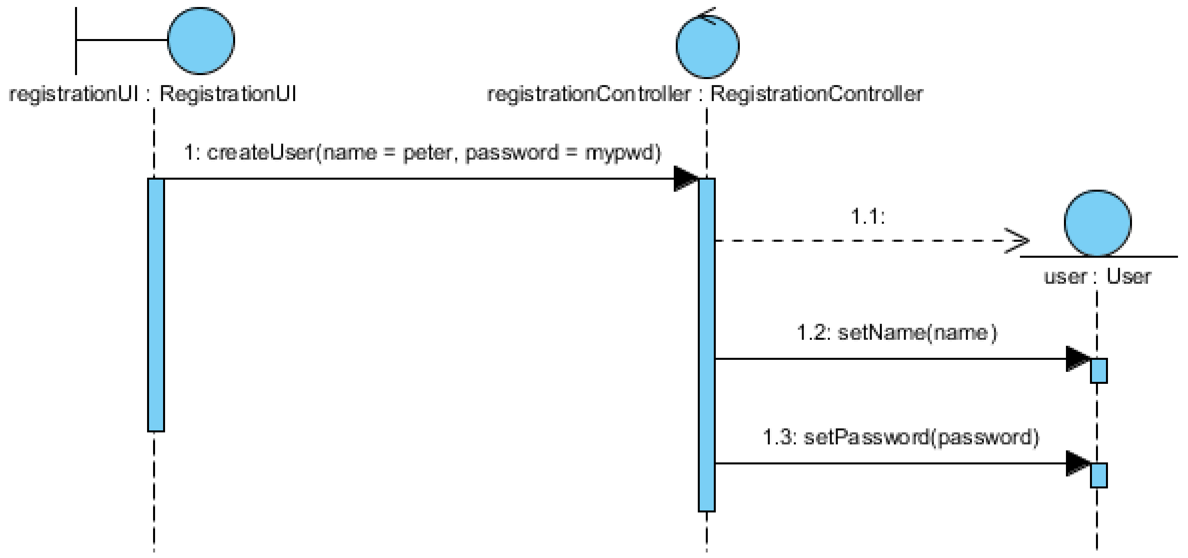
Component-level design: transforms structural elements of the software architecture into a procedural description of software components.
Interface design: describes how the software communicates with systems that interoperate with it, and with humans who use it
Architectural design: defines the relationship between major structural elements of the software, the architectural styles and patterns that can be used to achieve the requirements defined for the system, and the constraints that affect the way in which architecture can be implemented.
Data design/class design: transforms class models into design class realizations and the requisite data structure required to implement it.

2.1 Quality control in the design process - The Goals

- Three goals of a design in good quality:
  - The design should **implement all of the explicit requirements** contained in the requirements model, and it must accommodate all of the implicit requirements desired by stakeholders.
  - The design should be a **readable, understandable guide** for those who generate code and for those who test and subsequently support the software.
  - The design should provide **a complete picture of the software**, addressing the data, functional, and behavioral domains from an implementation perspective

Example: how about UML diagrams?




registrationUI : RegistrationUI
registrationController : RegistrationController
1: createUser(name = peter, password = mypwd)
1.1:
user : User
1.2: setName(name)
1.3: setPassword(password)

Source: https://www.visual-paradigm.com/tutorials/how-to-draw-uml-sequence-diagram.jsp

A UML diagram is a way to visualize systems and software using Unified Modeling Language (UML). Software engineers create UML diagrams to understand the designs, code architecture, and proposed implementation of complex software systems. UML diagrams are also used to model workflows and business processes.

2.2 Quality control in the design process – Why it matters

The importance of software design can be stated with a single word – quality.

- **Defect Prevention:** Implementing quality control early in the design phase helps identify and address potential issues before they go into later stages of development.
- **Cost Efficiency:** Fixing defects during the design phase is generally much less expensive than addressing them after coding or, worse, after deployment.
- **Improved User Satisfaction:** A well-designed software product that undergoes quality control is more likely to meet user needs and expectations.
- **AND:** Enhanced Reliability, Compliance and Standards Adherence, Documentation and Traceability and so on

2.3 Quality control in the design process – What to be Considered

Five quality attributes (FURPS):

- **Functionality** is assessed by evaluating the feature set and capabilities of the program, the generality of the functions that are delivered, and the security of the overall system.
- **Usability** is assessed by considering human factors, overall aesthetics, consistency, and documentation.
- **Reliability** is evaluated by measuring the frequency and severity of failure, the accuracy of output results, the mean-time-to-failure (MTTF), the ability to recover from failure, and the predictability of the program.
- **Performance** is measured using processing speed, response time, resource consumption, throughput, and efficiency.
- **Supportability** combines extensibility, adaptability, and serviceability. These three attributes represent a more common term, maintainability—and in addition, testability, compatibility, configurability, the ease with which a system can be installed, and the ease with which problems can be localized.

2.4a Quality control in the design process - Guidelines

Eight technical criteria for good design:

1.A design should exhibit an architecture that

(1) has been created using recognizable architectural styles or patterns,

(2) is composed of components that exhibit good design characteristics, and

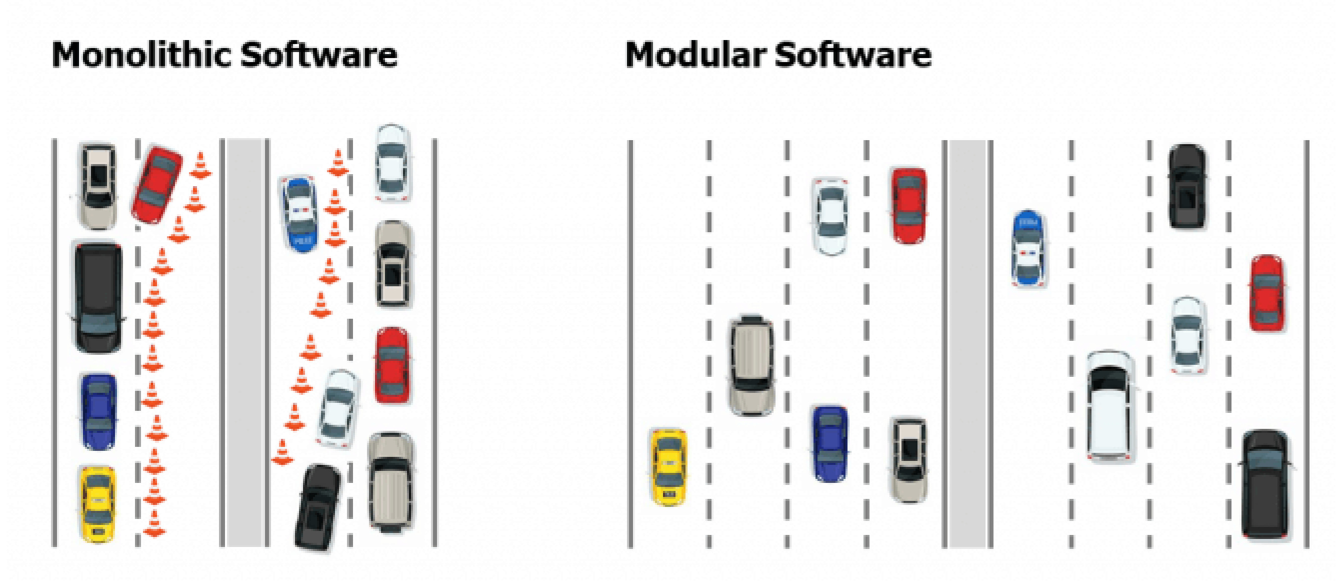
(3) can be implemented in an evolutionary fashion, thereby facilitating implementation and testing.

2.A design should be modular; that is, the software should be logically partitioned into elements or subsystems.

3.A design should contain distinct representations of data, architecture, interfaces, and components.

4.A design should lead to data structures that are appropriate for the classes to be implemented and are drawn from recognizable data patterns.

Example of modular design

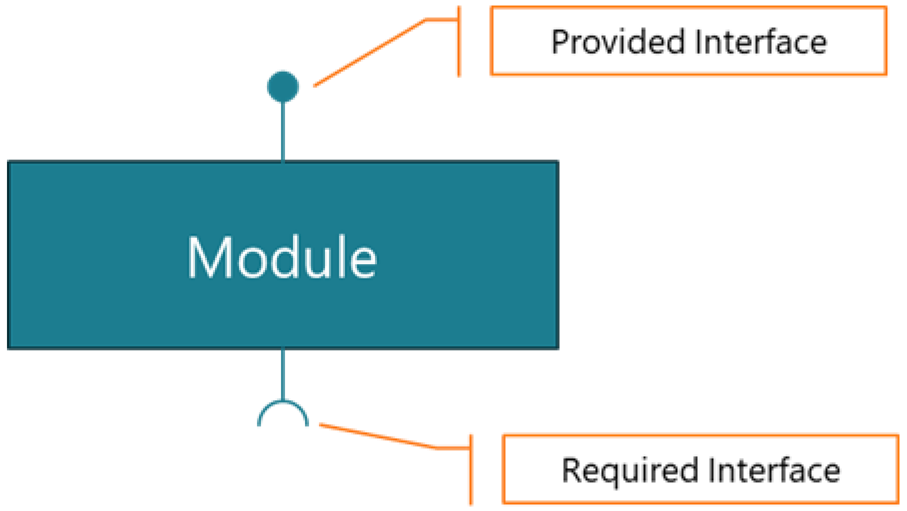


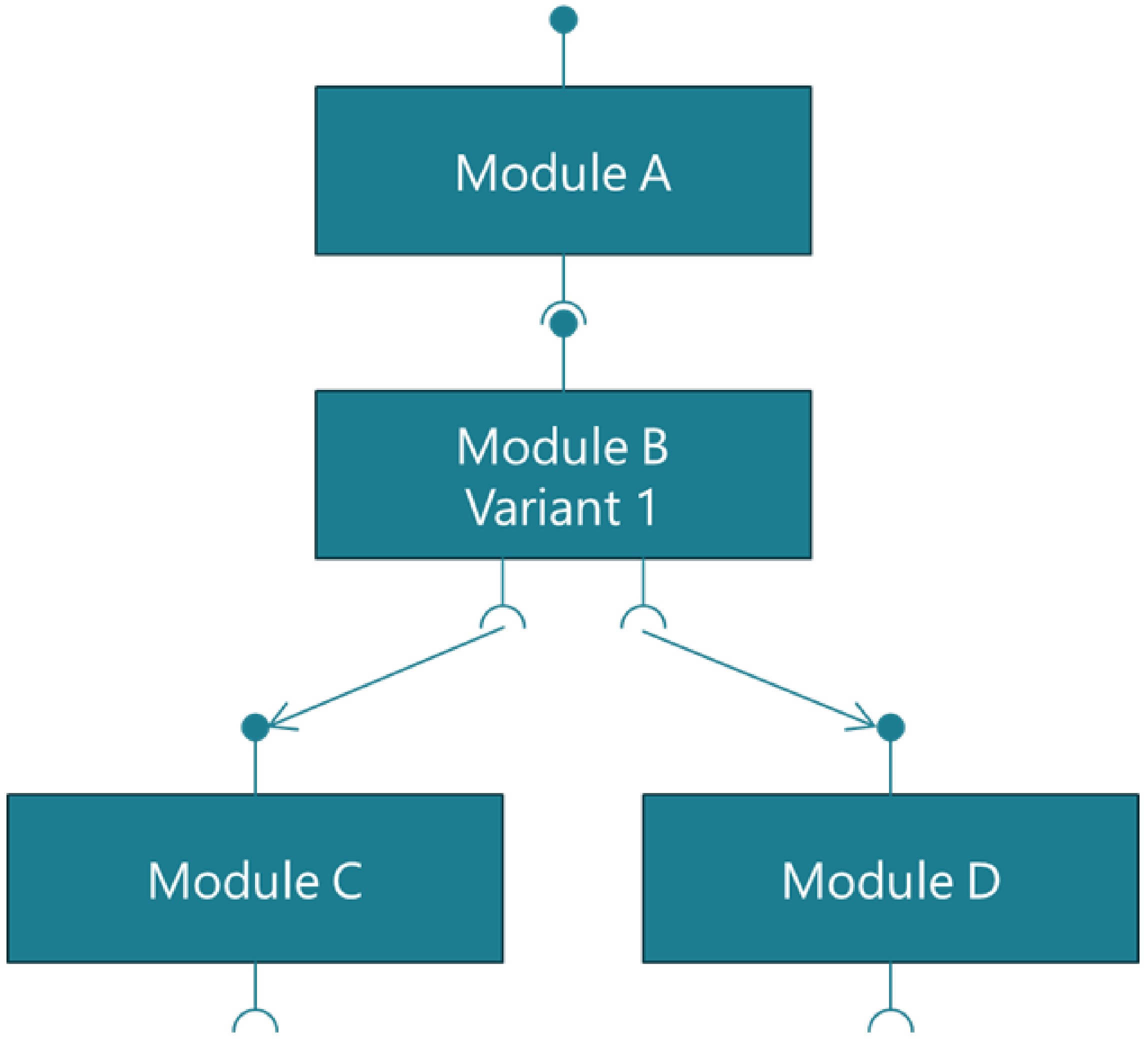
A module is a piece of code that can be independently created and maintained to be (re)used in different systems.

Source: https://www.modularmanagement.com/blog/software-modularity

https://www.ibm.com/think/topics/monolithic-architecture

Example of modular design





https://www.modularmanagement.com/blog/software-modularity

2.4b Quality control in the design process - Criteria

5. A design should lead to components that exhibit independent functional characteristics.
6. A design should lead to interfaces that reduce the complexity of connections between components and with the external environment.
7. A design should be derived using a repeatable method that is driven by information obtained during software requirements analysis.
8. A design should be represented using a notation that effectively communicates its meaning.

3.1 Design Concepts - What is it

The idea/principles the foundational ideas and principles that **guide the creation and organization of software systems**

It encompasses the guiding principles and methodologies used to ensure the software meets **functional requirements, maintains high quality, and adapts to future changes**.

It informs **decisions** related to architecture, component interactions, and user experience, ultimately shaping the system's effectiveness and usability.

3.2 Design concepts - Key Ones

Xi'an Jiaotong-Liverpool University 西交利物浦大学

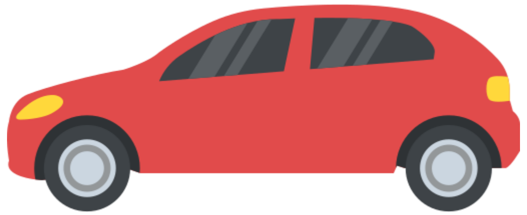
General design concepts:

- Abstraction
- Modularity
- Functional Independence
  - Coupling
  - Cohesion
- Object-oriented design

3.3 Design concepts - Abstraction

- Abstraction is a design concept that focuses on simplifying complex systems by highlighting essential features while hiding unnecessary details.
- The higher the level, the less the detail.
- At the highest level of abstraction, a concept / approach / solution is stated in broad terms using the language of the problem environment.
- At lower levels of abstraction, a more detailed description of the concept/approach/solution is provided
- From high to low: A car -> a Xiaomi Su7 -> a Xiaomi Su7 Ultra

Example of abstraction


A car

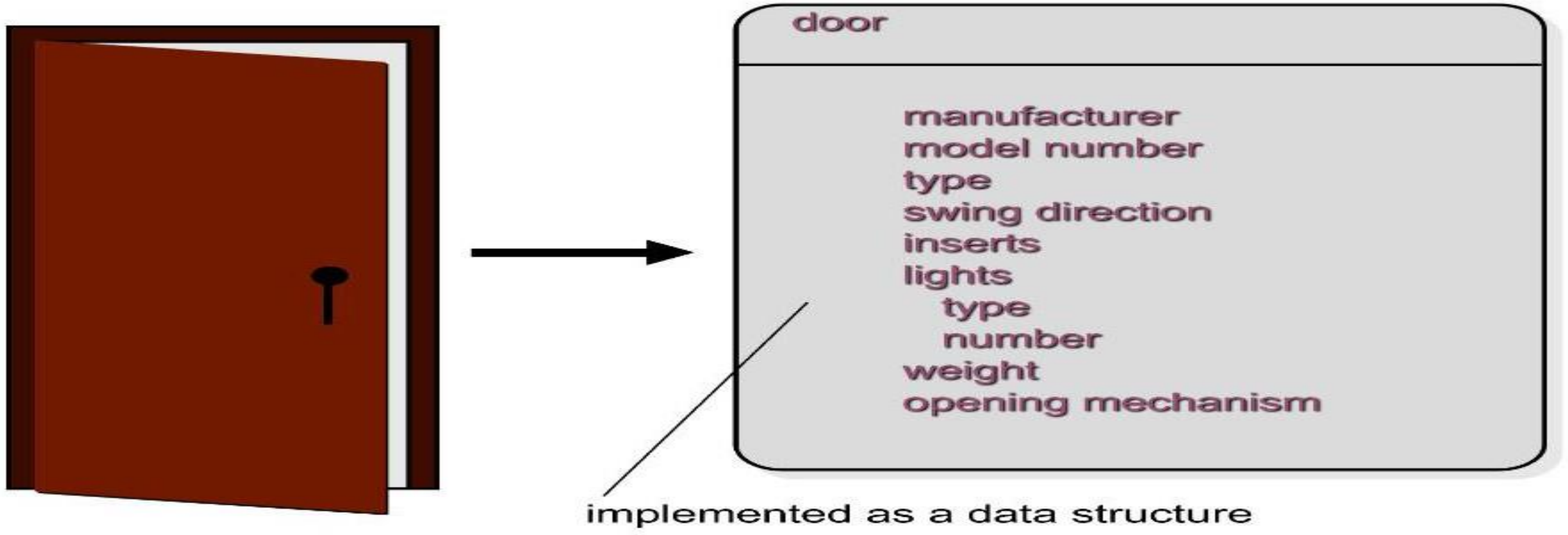
Xiaomi SU7

Xiaomi SU7 Ultra

Xiaomi SU7 Ultra 2025

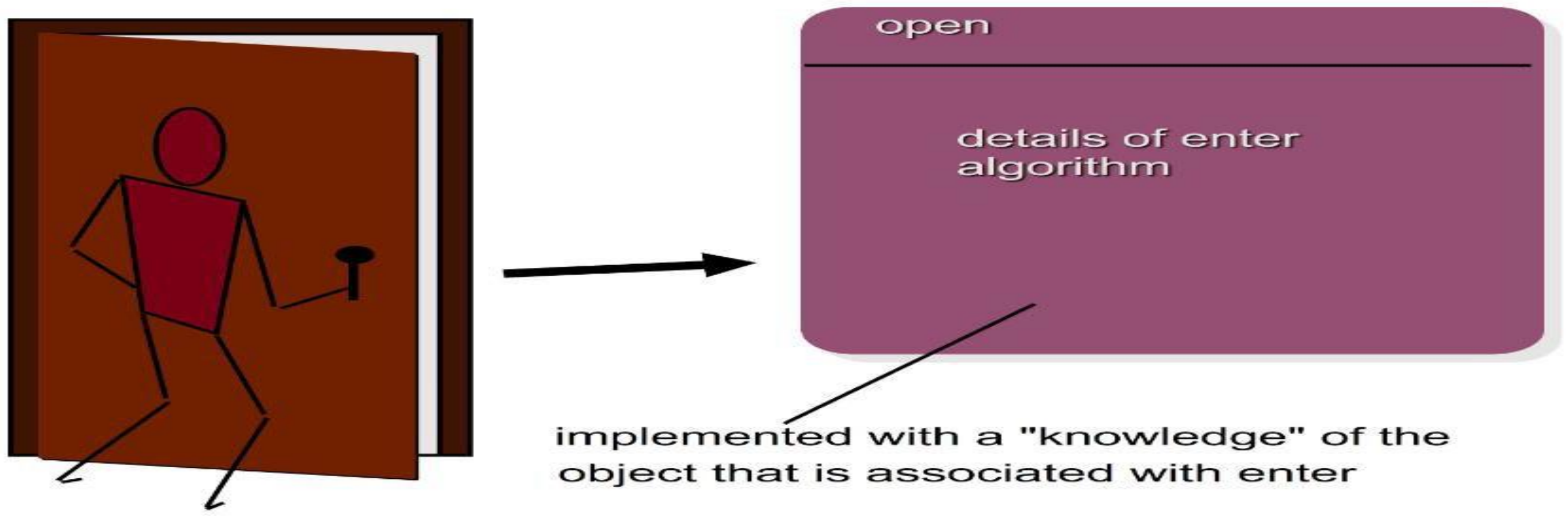
3.3.1 Design concepts - Data Abstraction

- **A data abstraction** is a named collection of data that describes a data object.



3.3.2 Design concepts – Procedural Abstraction

A procedural abstraction refers to a sequence of instructions that have a specific and limited function. The name of a procedural abstraction implies these functions, but specific details are omitted.


open
details of enter algorithm
implemented with a "knowledge" of the object that is associated with enter

open()–> walk to the door, reach the knob, pull the door, step away

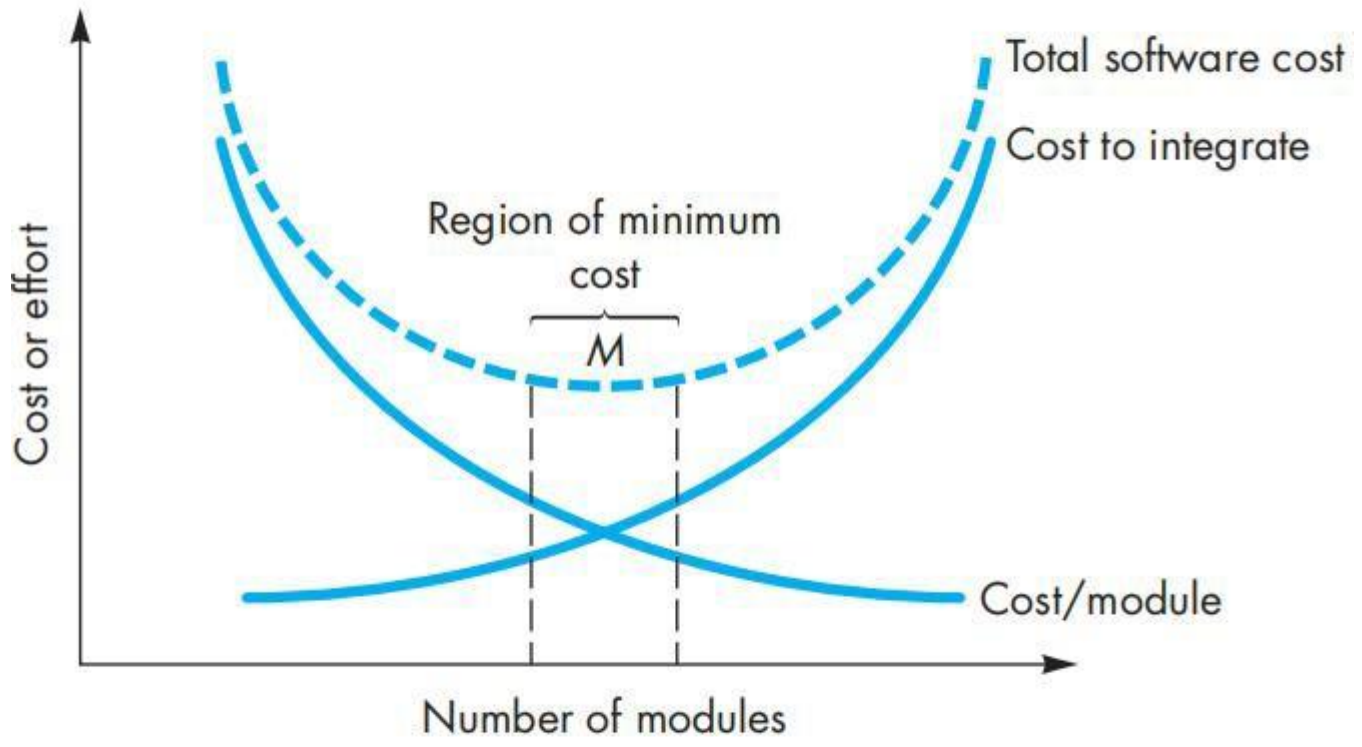
3.4a Design concepts - Modularity

- Modular design helps us to better organize complex system.
- It basically clusters similar or relative functions together, sets up boundaries and provides interfaces for communication
  - mobile phone as an complex system that is modularized
  - modularity increases manufacture efficiency and save time
- Modularity allows the possibility of the development of system parts to be carried out independently of each other, therefore reducing development time.
- However, too many modules in a system increase the complexity of modules integration.

3.4b Design concepts - Modularity

Xi'an Jiaotong-Liverpool University 西交利物浦大学

Given the same set of requirements, more modules means smaller individual size. However, as the number of modules grows, the effort (cost) associated with integrating the modules also grows. These characteristics lead to a total cost or effort curve shown in the figure.



3.5 Design concepts - Functional independence

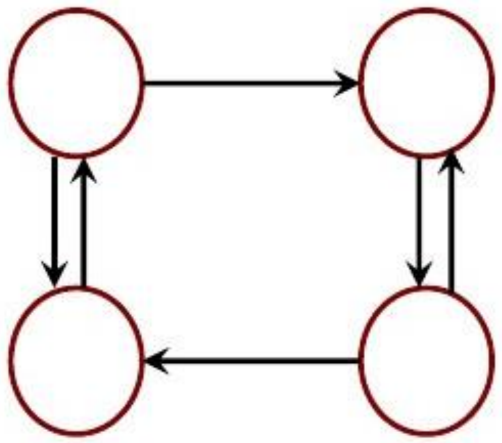
Functional Independence refers to the degree to which a software module or component can operate independently of other modules or components within a system. It is characterized by two key attributes:

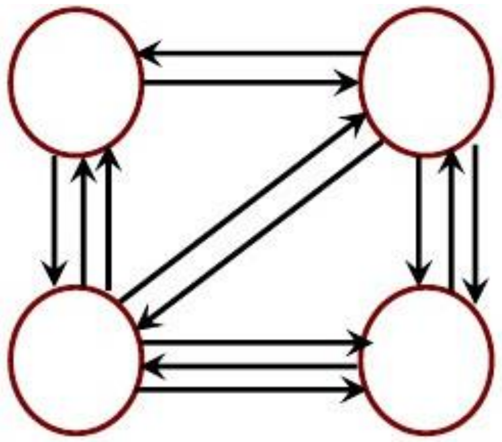
- **Low Coupling:** The module has minimal dependencies on other modules, allowing it to function without needing to interact heavily with external components. This means that changes in one module are less likely to impact others
- **High Cohesion:** The module is focused on a single, well-defined task or responsibility, ensuring that its internal elements work together closely and contribute to a common purpose.

3.5.1 Design concepts - Coupling

- Coupling refers to **the degree of interdependence between different modules or components**.
- To solve and modify a module separately, we would like the modules to be **loosely coupled**
- Low coupling means that modules are largely independent of each other and often interact through well-defined interfaces.

Coupling illustrations


Loosely coupled:
some dependencies


Highly coupled:
many dependencies

Linkedin

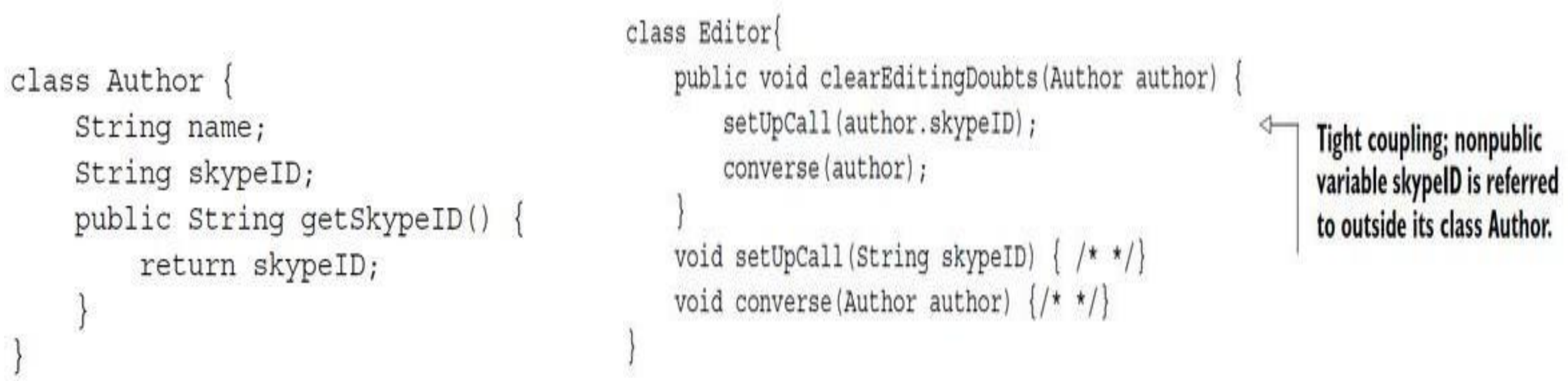
Loose vs Tight Coupling

https://www.youtube.com/watch?v=uWseUdUqM5U

1 to 4 min

3.5.1 Design concepts - Coupling

- **Tight coupling (a bad example)**



```
class Author {
    String name;
    String skypeID;
    public String getSkypeID() {
        return skypeID;
    }
}
```

```
class Editor{
    public void clearEditingDoubts(Author author) {
        setUpCall(author.skypeID);
        converse(author);
    }
    void setUpCall(String skypeID) { /* */}
    void converse(Author author) {/* */}
}
```

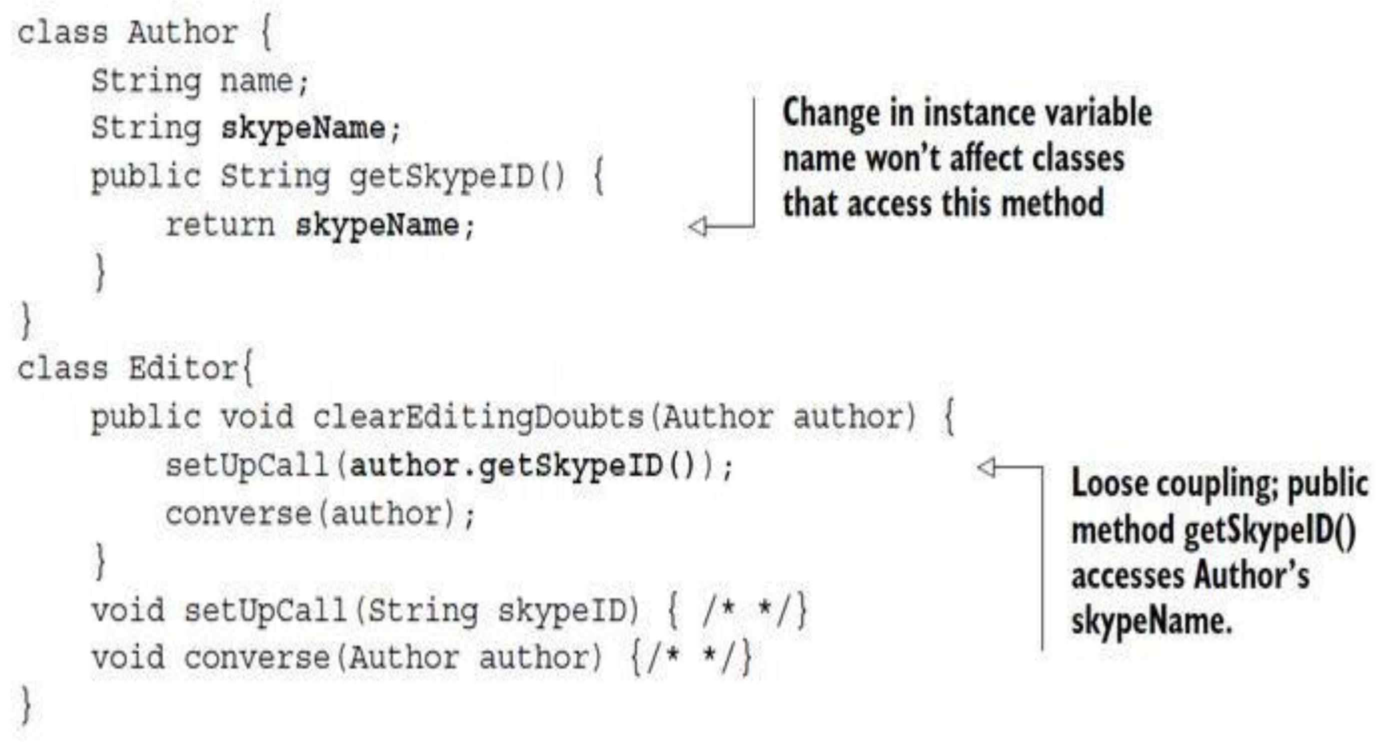
Tight coupling; nonpublic variable skypeID is referred to outside its class Author.

What happens, if a programmer changes the name of the variable **skypeID** in class Author to **skypeName**?

3.5.1 Design concepts - Coupling

Xi'an Jiaotong-Liverpool University 西交利物浦大学

Solution:



```
class Author {
    String name;
    String skypeName;
    public String getSkypeID() {
        return skypeName;
    }
}
class Editor{
    public void clearEditingDoubts(Author author) {
        setUpCall(author.getSkypeID());
        converse(author);
    }
    void setUpCall(String skypeID) { /* */}
    void converse(Author author) {/* */}
}
```

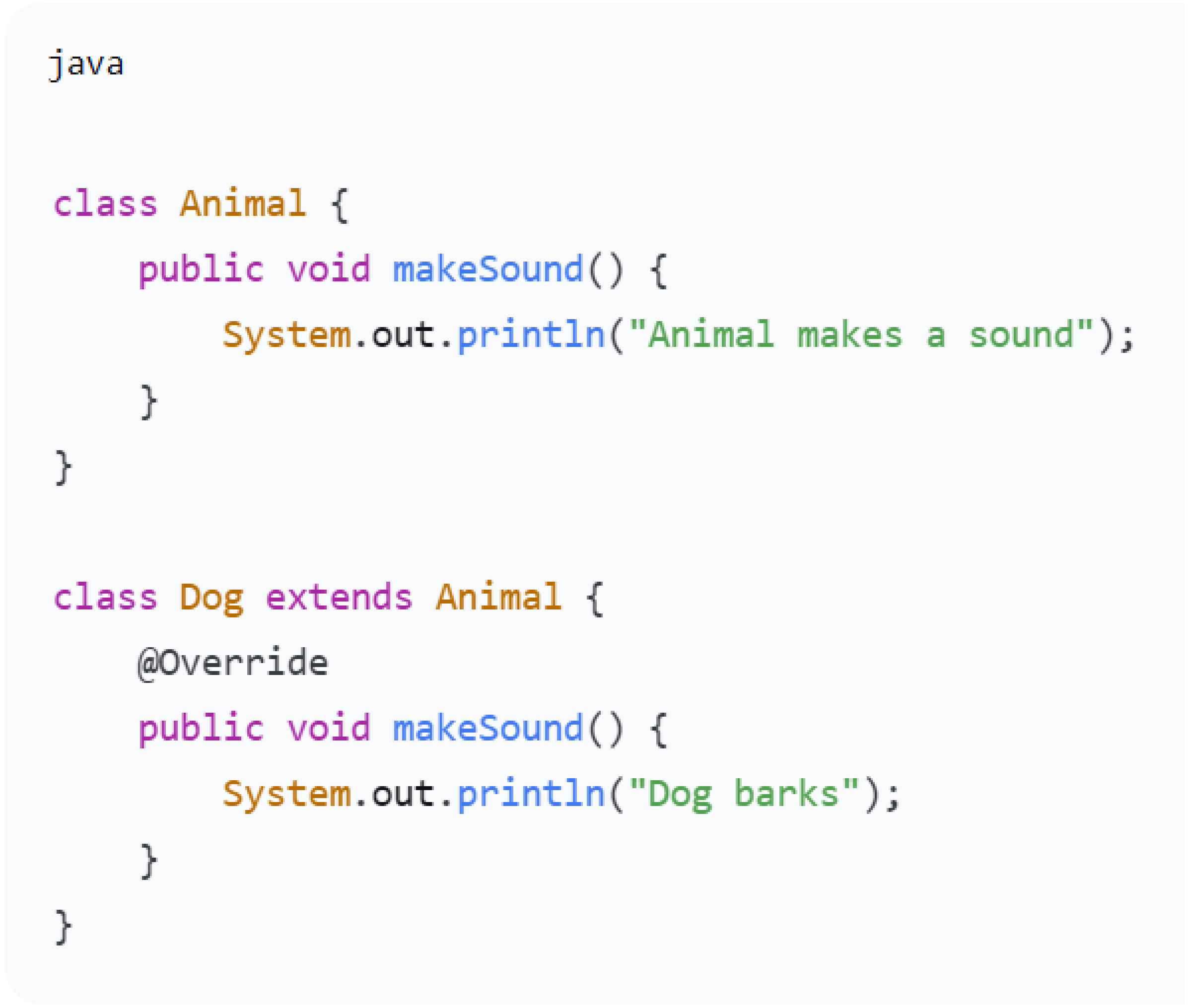
Change in instance variable name won't affect classes that access this method

Loose coupling; public method getSkypeID() accesses Author's skypeName.

Use the public method getSkypeID() in class Editor (changes in bold)

Xi'an Jiaotong-Liverpool University 西交利物浦大学

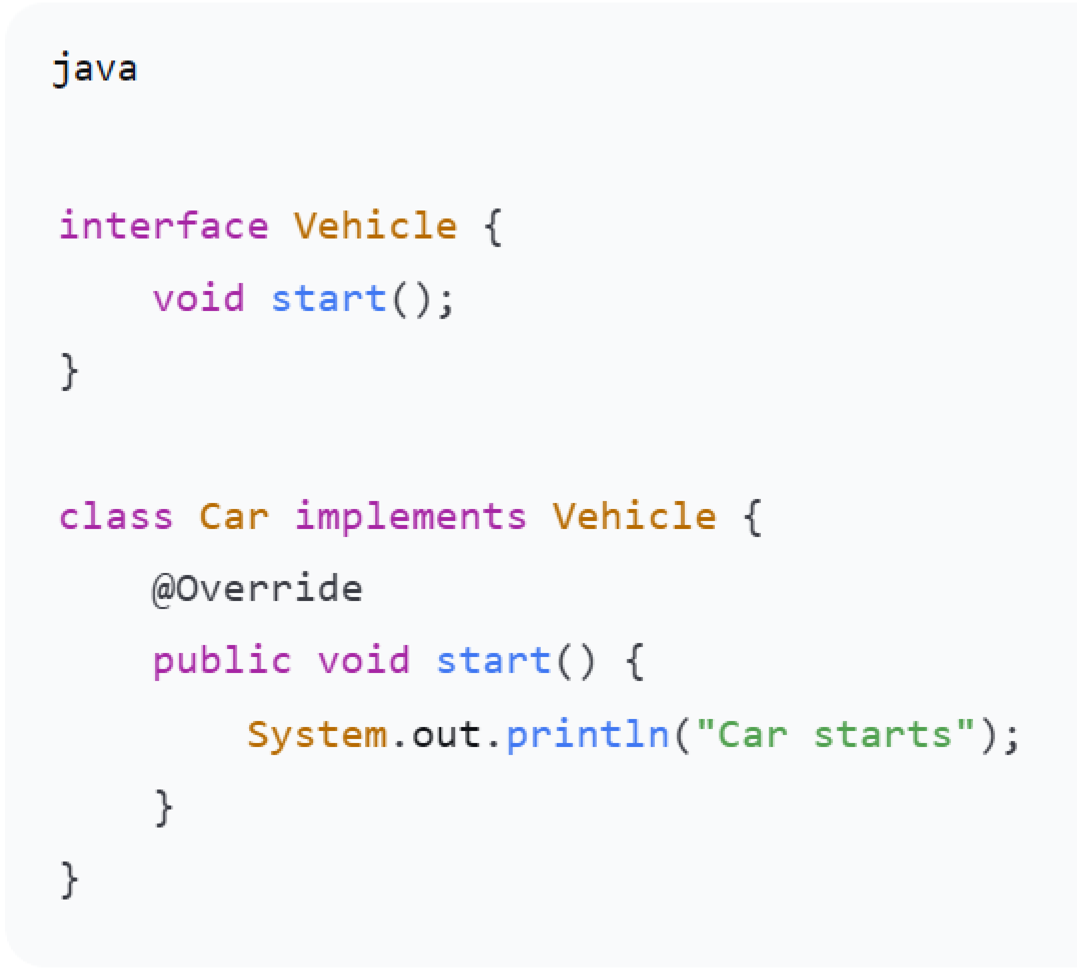
Examples using @Override



```java
class Animal {
    public void makeSound() {
        System.out.println("Animal makes a sound");
    }
}

class Dog extends Animal {
    @Override
    public void makeSound() {
        System.out.println("Dog barks");
    }
}
```

Overriding a superclass method



```java
interface Vehicle {
    void start();
}

class Car implements Vehicle {
    @Override
    public void start() {
        System.out.println("Car starts");
    }
}
```

Implementing an interface method

What happens, if we forget to write @Override:

```
class Dog extends Animal {
    public void makeSoud() {  // No error! This creates a NEW method, doesn't override
        System.out.println("This is a bug - won't be called polymorphically");
    }
}
```

Additional reading: decoupling using Java interface

``Decoupling in programming means *reducing the interdependence* between the different components or modules of a software system.

It is all about creating components that are independent and can be used elsewhere with minimal changes, and that can be tested and maintained independently''

- Java interface for decoupling in java -additional materials

- https://www.ellej.dev/blog/decoupling-using-interfaces-and-dependency-injection-in-java/

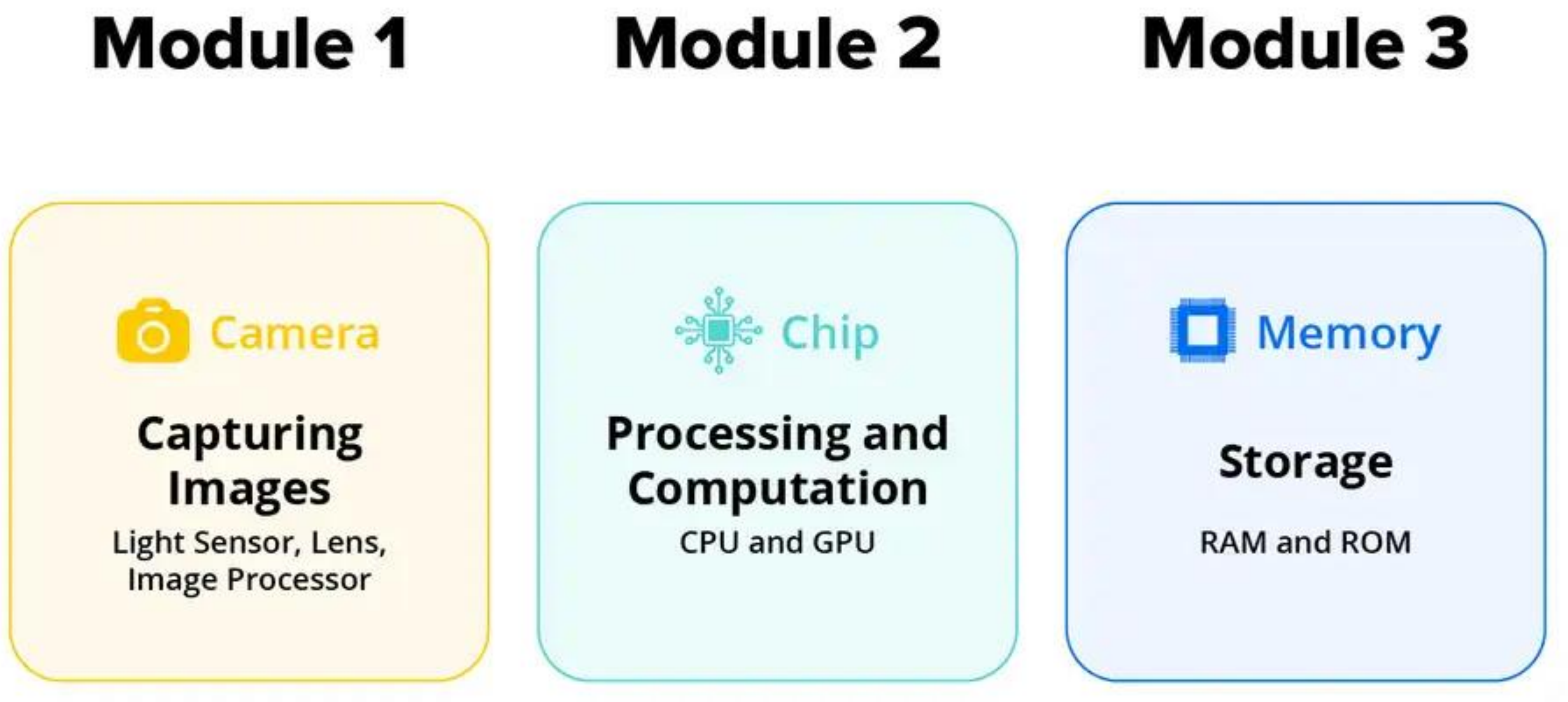
- https://dev.to/ivangavlik/decoupling-using-example-in-java-4cgb

3.5.2 Design concepts - Cohesion

- Cohesion refers to **how closely related and focused** the responsibilities of a single module or component are.
- High cohesion within a module **also** contributes to functional independence by ensuring that the module is focused and self-contained. It usually means that the elements within a module work together to **perform a single task or function**.

- **Types:**
  - Method Cohesion
  - Class Cohesion
  - Module Cohesion
  - Component Cohesion
  - …

Example of High Cohesion

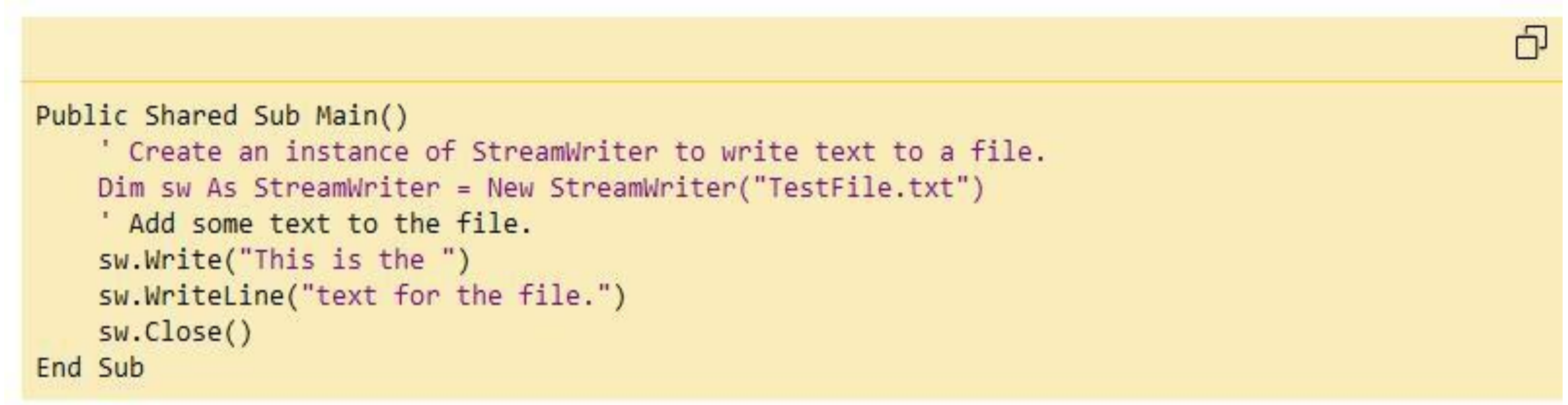


a module is performing a single task

Source : https://www.haptik.ai/tech/why-product-development-and-design-needs-cohesion-coupling

3.5.2 Design concepts - Cohesion

Method cohesion: A method that implements a clearly defined function, and all statements in the method contribute to implementing this function.



```
Public Shared Sub Main()
    ' Create an instance of StreamWriter to write text to a file.
    Dim sw As StreamWriter = New StreamWriter("TestFile.txt")
    ' Add some text to the file.
    sw.Write("This is the ")
    sw.WriteLine("text for the file.")
    sw.Close()
End Sub
```

- The above method performs a sequence of tasks such as opening a file, write to a file, and close a file.
- How about adding a function **sw.Read()**?

3.5.2 Design concepts - Cohesion

Xi'an Jiaotong-Liverpool University 西交利物浦大学

Class cohesion

- A high class cohesion is the class that implements a single concept or abstraction with all elements contributing toward supporting this concept.
- Highly cohesive classes are usually more easily understood and more stable.

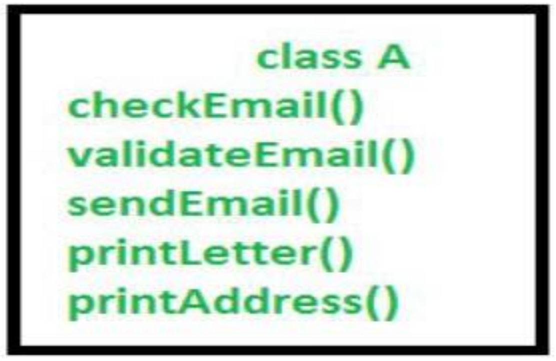


Fig: Low cohesion

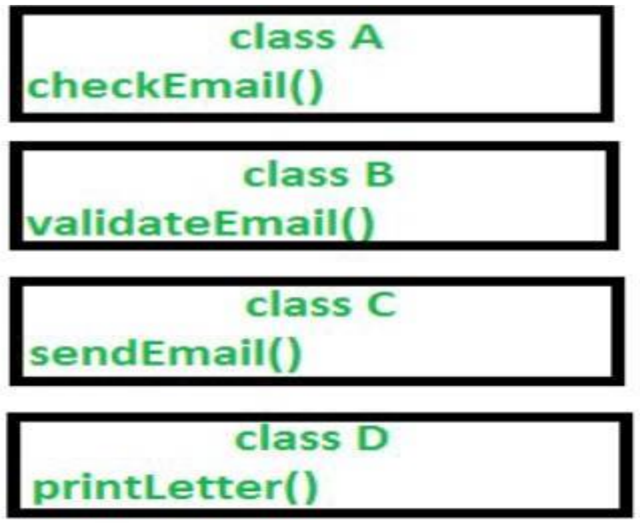
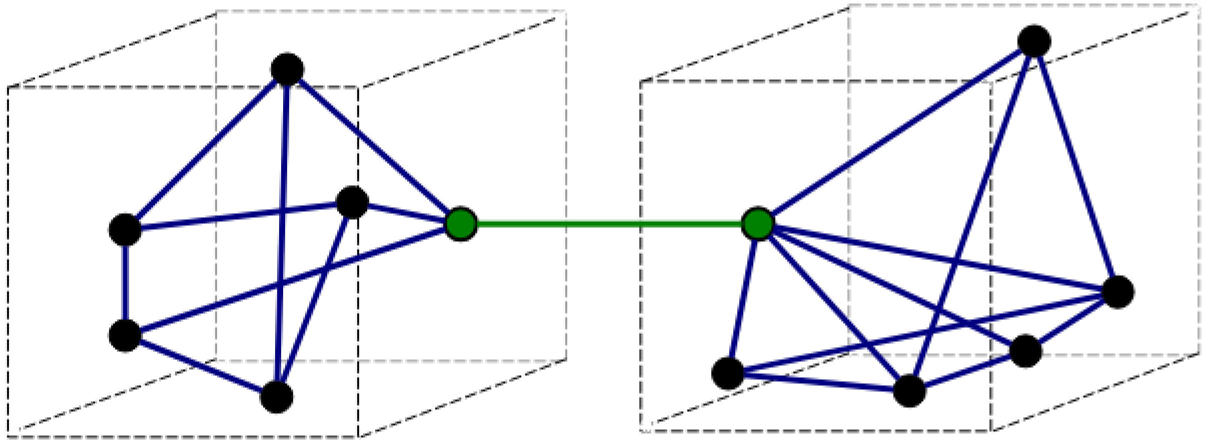


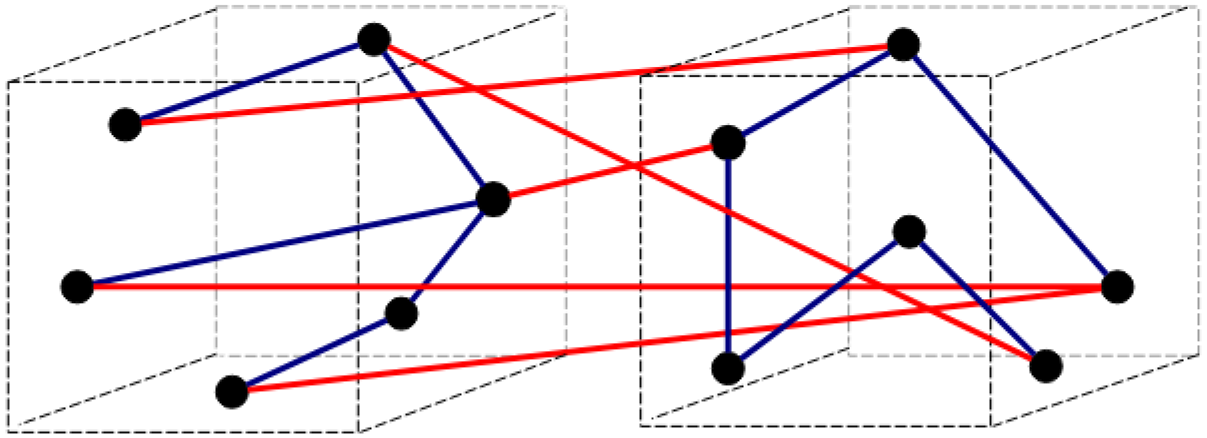
Fig: High cohesion

https://www.geeksforgeeks.org/java/cohesion-in-java/

Example : which one is a better design? Why?



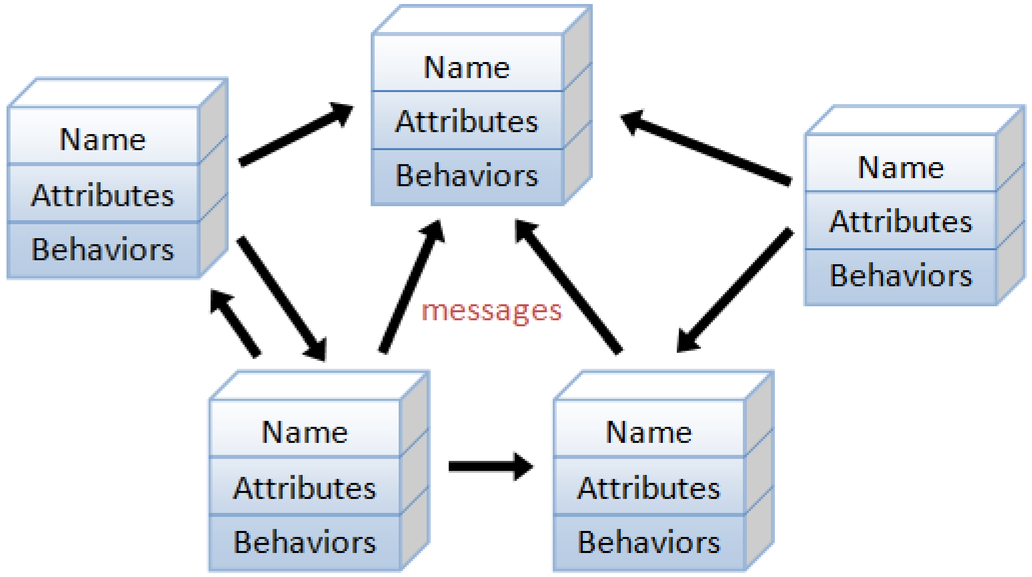
a)



b)

3.6 Design concepts – Object-Oriented Design

- Object-Oriented Design is a software design approach that **organizes a system as a collection of interacting objects**, each representing an instance of a class within a specific domain.
- **These objects encapsulate both data (attributes) and behavior (methods)**, promoting modularity, reusability, and scalability.



An object-oriented program consists of many well-encapsulated objects and interacting with each other by sending messages

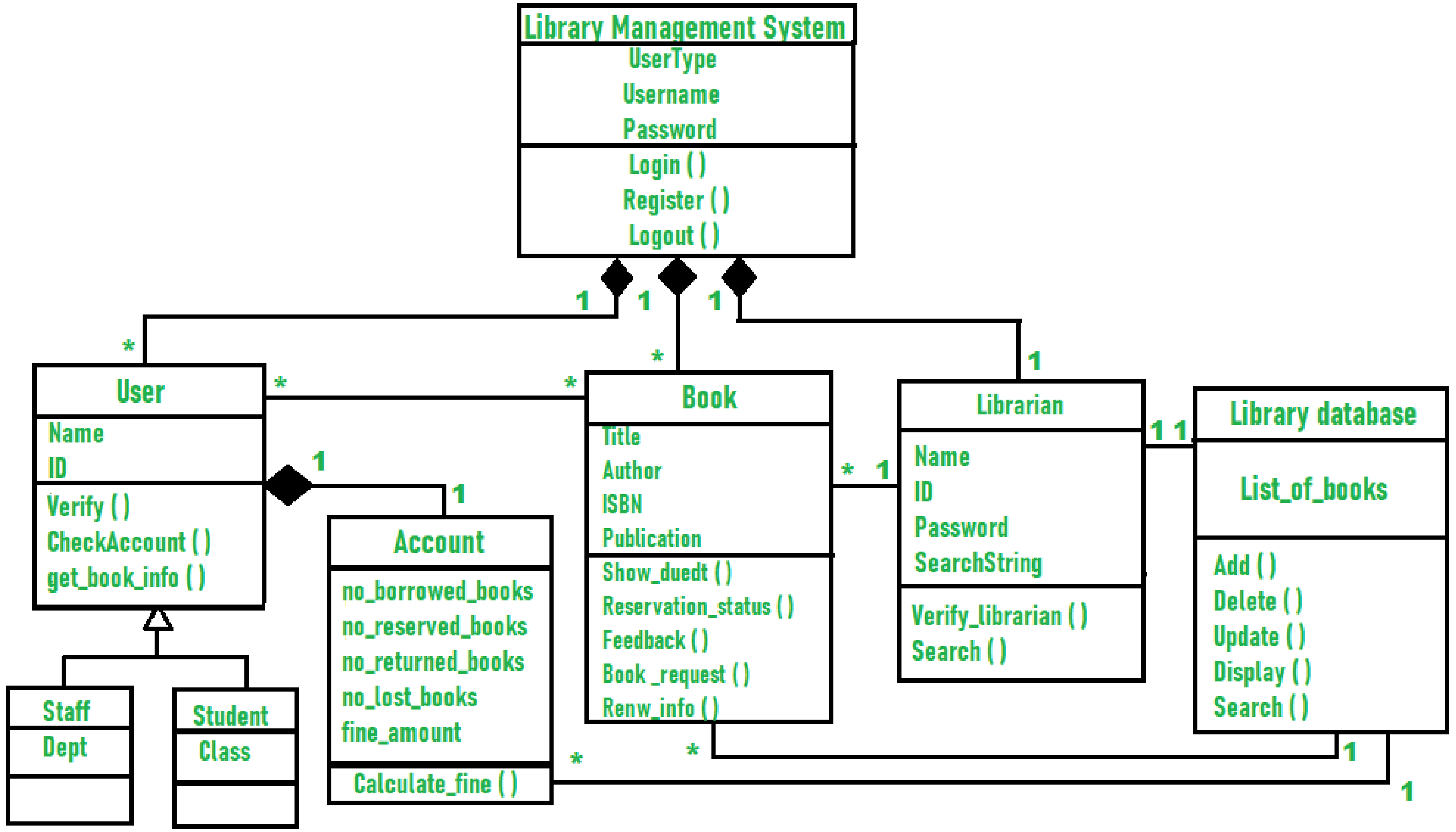
https://www3.ntu.edu.sg/home/ehchua/programming/java/J3a_OOPBasics.html

3.6 Design concepts – Object-Oriented Design

- A case: We want to create a system for managing books in a library. This system should:
    1. Allow users to search for and borrow books.
    2. Track which books are borrowed and which are available.
    3. Allow administrators to add new books to the library.

- We do:
    1. Identify Classes (i.e., Book, User, Library)
    2. Define Relationships (e.g., a library *contains* many Book objects, etc)
    3. **Define Attributes and Methods** (e.g., A Book object should have attributes like Title, author, etc; it should also have methods including borrow(), returnBook(), etc)

Example



CLASS DIAGRAM FOR LIBRARY MANAGEMENT SYSTEM

https://www.geeksforgeeks.org/software-engineering/class-diagram-for-library-management-system/

3.6 Design concepts

Object-Oriented Design Principles

- **Encapsulation:** Each class encapsulates its data and methods, allowing controlled access to its attributes (e.g., Book controls access to isAvailable).
- **Inheritance:** In a more complex system, we might have classes like DigitalBook and PhysicalBook that inherit from Book.
- **Polymorphism:** Different types of books or users might override methods for specific behaviors, if necessary.

Explanations with examples

Encapsulation

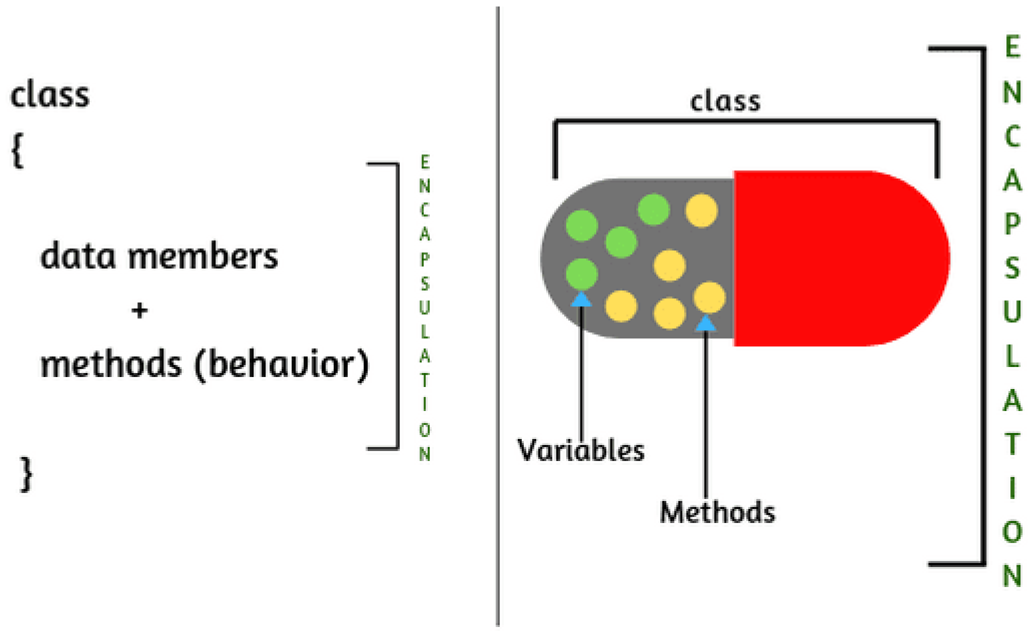
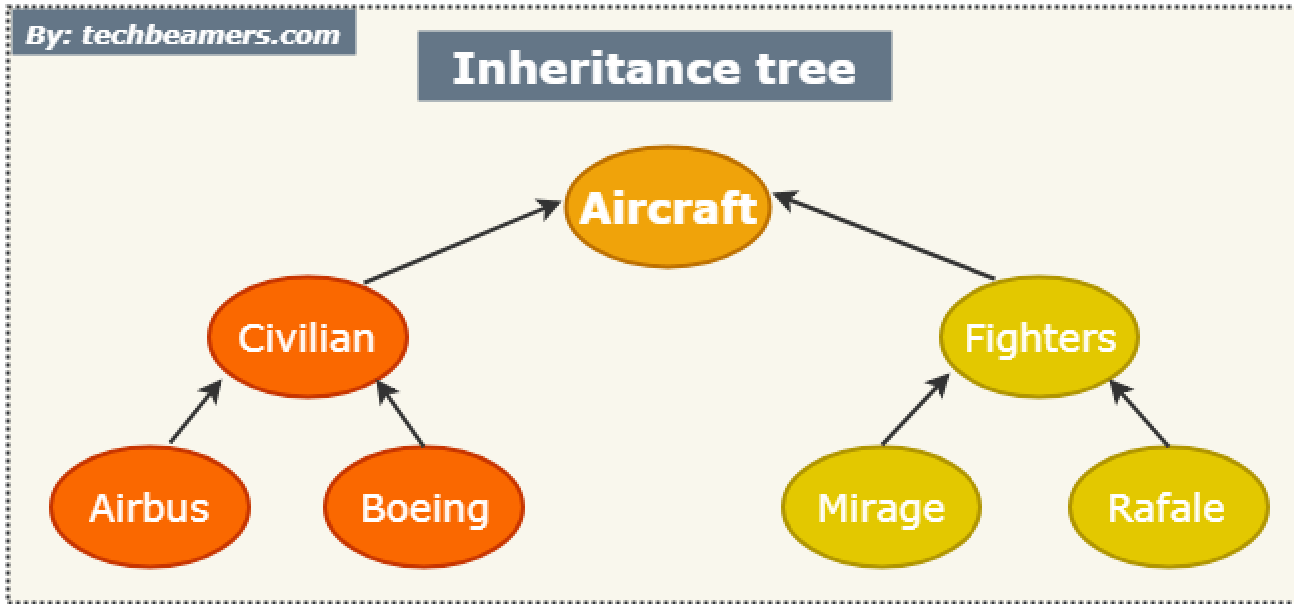


Fig: Encapsulation

Inheritance

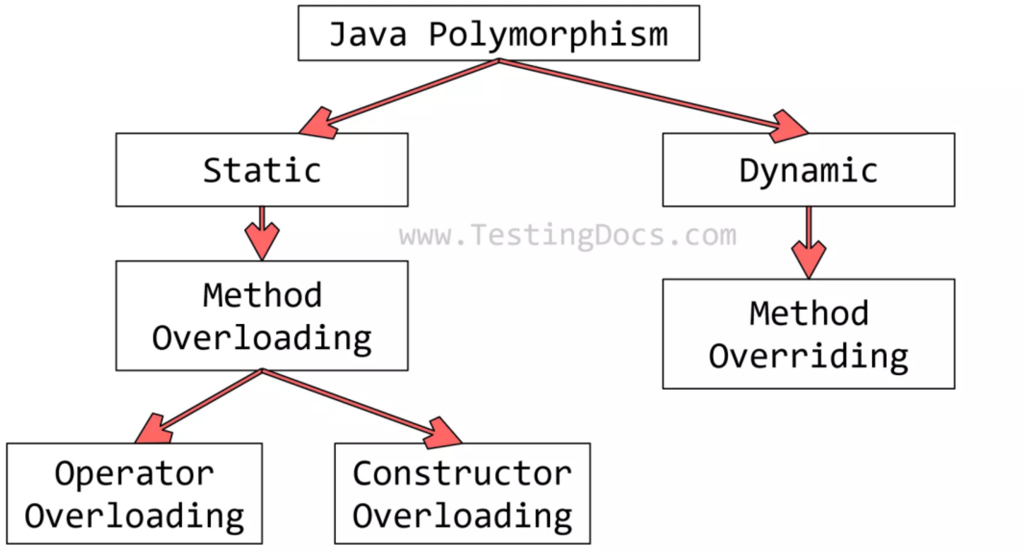


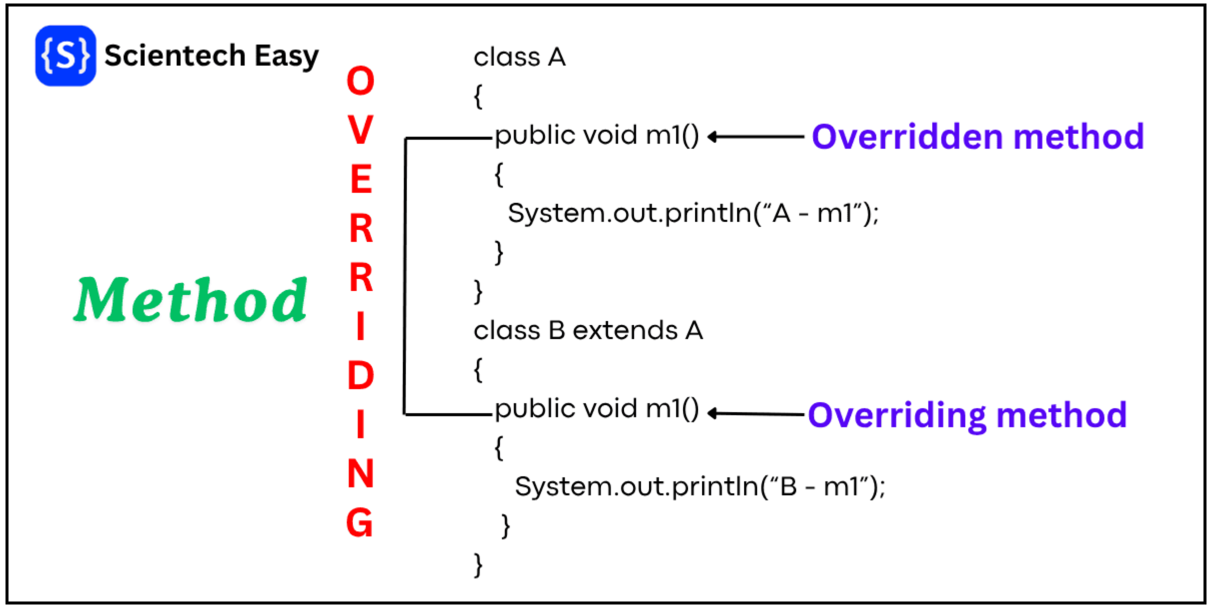
a newly built class extracts features (methods and variables/fields) from an already existing class.

https://www.scientecheasy.com/2020/07/encapsulation-in-java.html/
https://techbeamers.com/java-inheritance/
https://www.geeksforgeeks.org/java/understanding-encapsulation-inheritance-polymorphism-abstraction-in-oops/

Additional explanations with examples

Polymorphism English meaning – ‘poly’ means ‘many’ and ‘morph’ means ‘forms’. polymorphism means many forms.





```
class A
{
  public void m1()
  {
    System.out.println("A - m1");
  }
}
class B extends A
{
  public void m1()
  {
    System.out.println("B - m1");
  }
}
```

https://techvidvan.com/tutorials/java-polymorphism/

scientecheasy.com/2020/07/method-overriding-java.html/

4.1 Elements in Design Model

We can see the **Design Model** as a detailed framework, a set of approaches, that guide the development of a software system from analysis to implementation. It include the following elements (aspects):

- Data design elements
- Architectural Design Elements
- Interface Design Elements
- Component-Level Design Elements
- Deployment-Level Design Elements

4.2 Elements in design model-Data Design

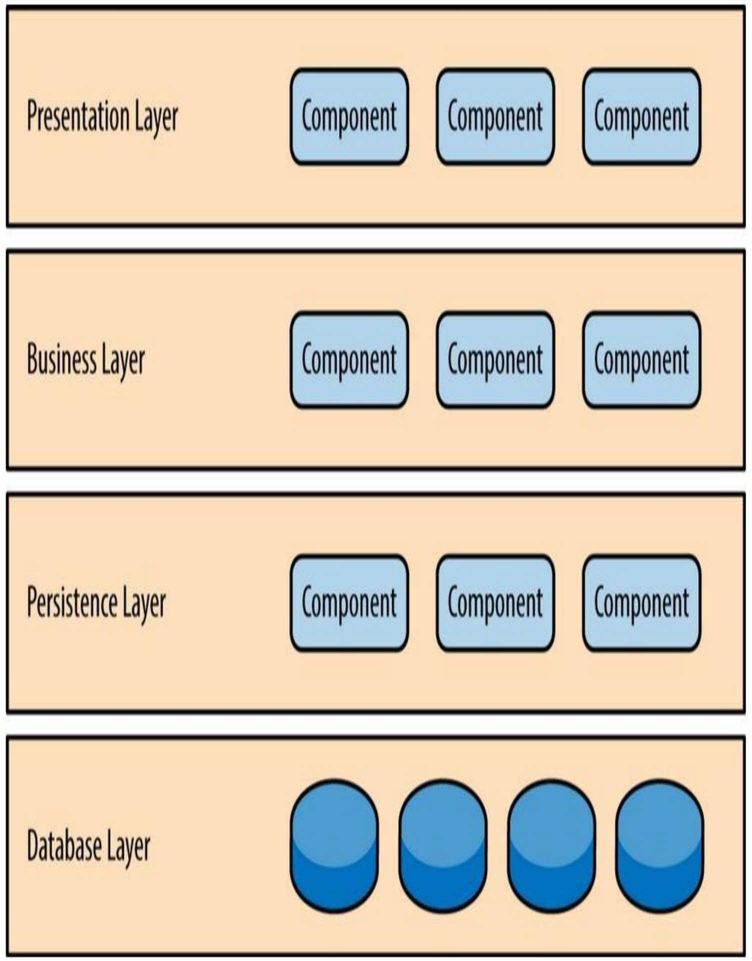
Like other software engineering activities, **data design** (sometimes referred to as data architecting) creates a model of data and/or information that is represented at a high level of abstraction (the customer/user's view of data).

It is then refined progressively into more implementation-specific representations that can be processed by the computer-based system, based on the following level (e.g., e-commerce system):

- **Business level**: Define high-level goals (increase sales, optimize inventory, enhance satisfaction) and identify required data (product, customer, order, inventory).
- **Application level**: Design data usage for core functionality (product catalog, order processing, inventory management, recommendation engine).
- **Program-component level**: Implement data structures (Product, Customer, Order, Inventory, Recommendation Engine - all as **Var**)

4.3 Elements in design model – Architectural Design

- The architectural design for software is the equivalent to the floor plan of a house. The floor plan depicts the overall layout of the rooms; their size, shape, and relationship to one another; and the doors and windows that allow movement into and out of the rooms.
- The floor plan gives us an overall view of the house. Architectural design elements give us an overall view of the software.
- **Three sources:**
  - Information about the application domain for the software to be built;
  - Specific requirements model elements such as use cases or analysis classes, their relationships and collaborations for the problem at hand; and
  - The availability of architectural styles and patterns



Xi'an Jiaotong-Liverpool University 西交利物浦大学

Introduction to architecture patterns

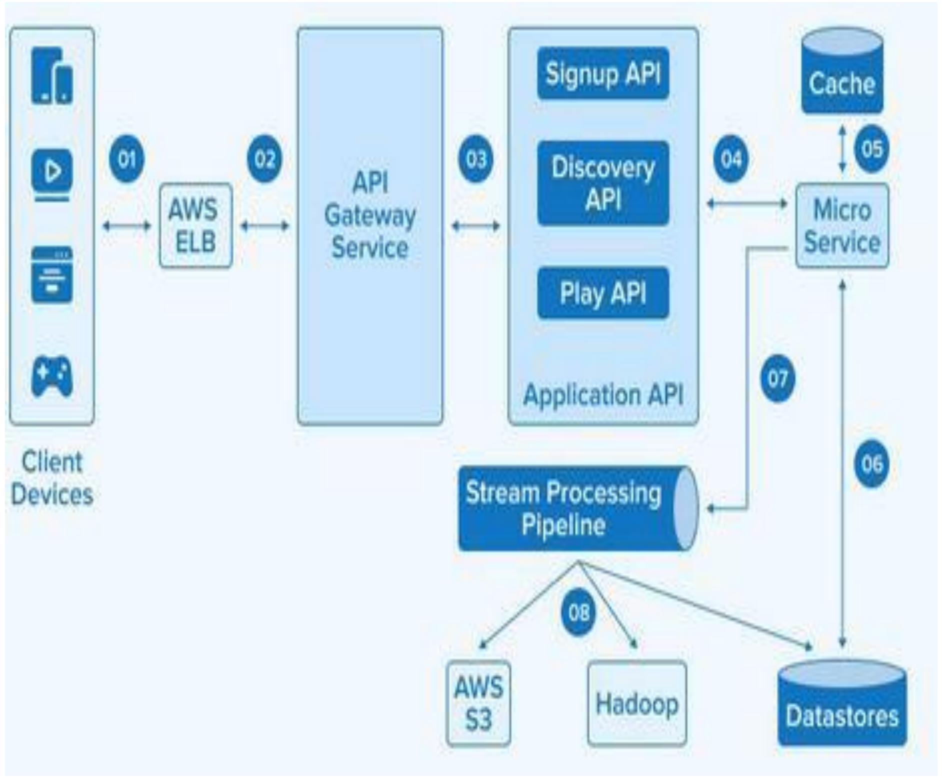
- https://www.youtube.com/watch?v=f6zXyq4VPP8

- 5min

4.4 Elements in design model – Interface Design

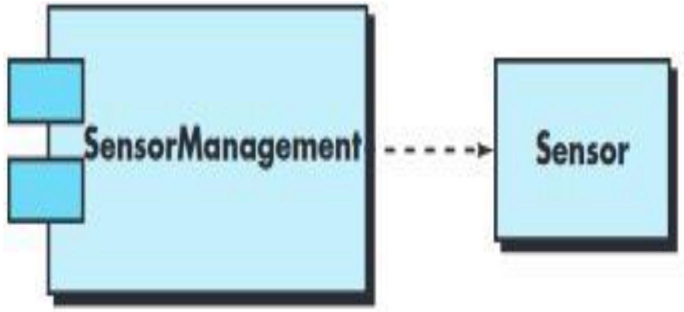
Xi'an Jiaotong-Liverpool University 西交利物浦大学

- The interface design for software is similar to a set of detailed drawings (and specifications) for the doors, windows, and external utilities of a house.
- The interface design elements for software depict information flows into and out of a system and how it is communicated among the components defined as part of the architecture.
- **Three important elements:**
  - The user interface (UI),
  - External interfaces to other systems, devices, networks, or other producers or consumers of information, and
  - Internal interfaces between various design components.



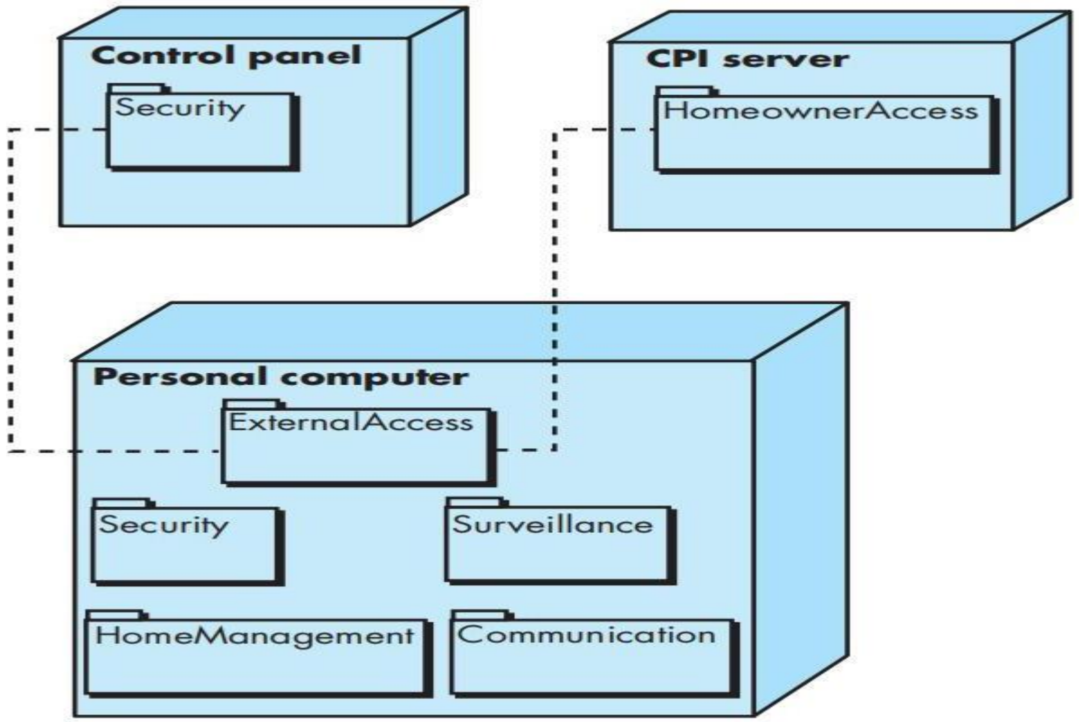
4.5 Elements in design model – Component-Level

- The component-level design for software is the equivalent to a set of detailed drawings (and specifications) for each room in a house.
- The component-level design for software fully describes the internal detail of each software component (e.g., an object).
- In software engineering, a component is represented in UML diagrammatic form.
- The design details of a component can be modeled at many different levels of abstraction
  - e.g., Flowchart or box diagram -> detailed procedural flow for a component.
  - Or, the figure on the right



4.6 Elements in design model -Deployment-level

- Deployment-level design elements indicate how software functionality and sub-systems will be allocated within the physical computing environment that will support the software



SUMMARY

Design in the Software Engineering
Quality control in the design process
Design concepts
Elements in design model