AS - Teoria 2

Unit 2.1. Layered Architecture

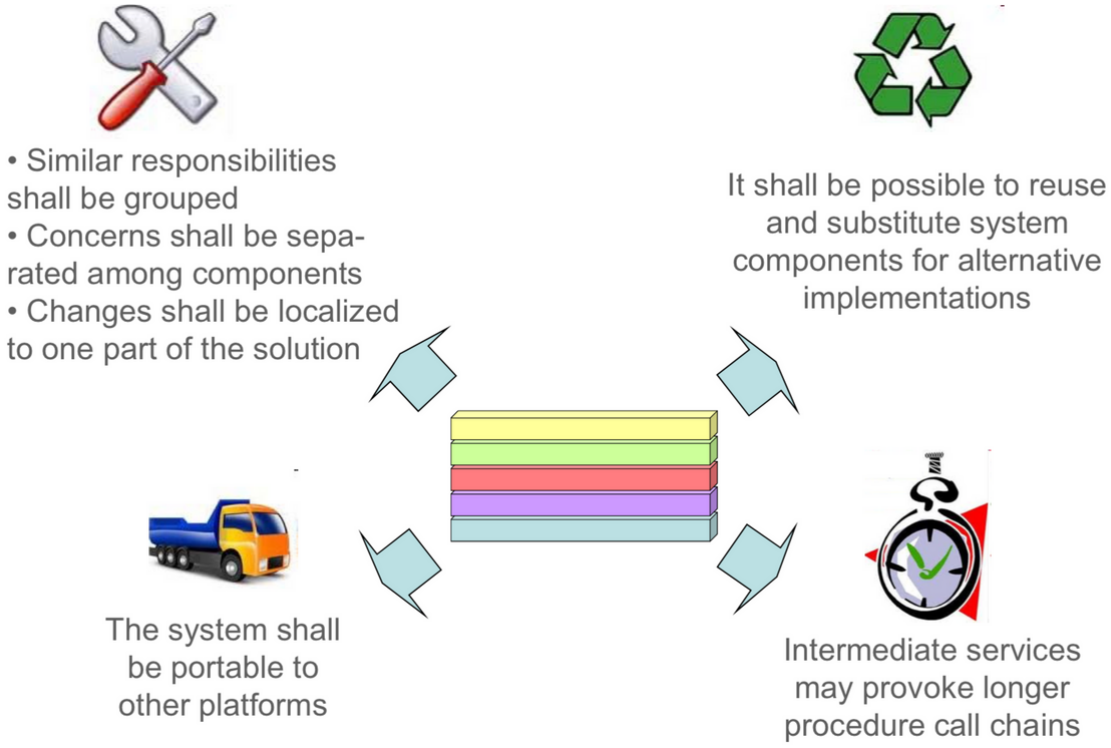
Context

A complex system that is composed of a large number of components across multiple levels of abstraction.

Problem

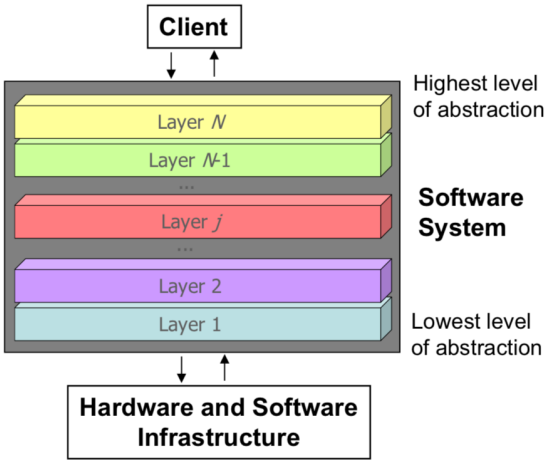
How do you structure a system to support such requirements as maintainability, reusability, portability, scalability, robustness, and security?

Forces to balance:



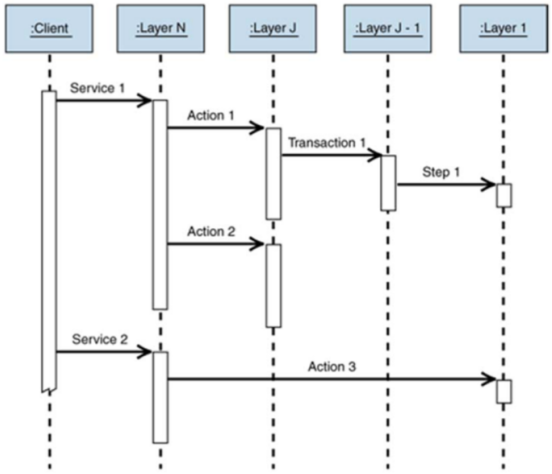
Solution

- The system is structured in an appropriate number of layers.
- All the layers are set vertically.
- All components in the same layer must work at the same level of abstraction.
- The services provided by layer **j** use the services provided by layer **j - 1**. Simultaneously, layer **j** services can depend on other services included in same layer.



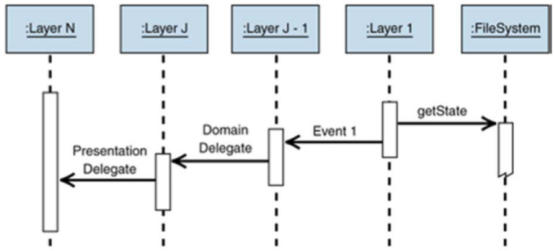
Top-down communication

An external entity interacts with the highest layer of the stack. The highest layer uses one or more services of the lower layers



Bottom-up communication

Layer 1 is monitoring the state of some external entity and notifies it to the higher layers



Architecture Definition Considerations

- Define the abstraction criteria and the number of layers
- Determine the functionalities in each layer:
  - The highest layer defines the services exposed to the outside.
- Specify the contracts for each layer:
  - The interface defines the services exposed to the immediately upper layer .
- Structure the layers individually
  - A component cannot be distributed into two or more layers.
- Apply design principles
  - Both inter- and intra- layer
- Design a strategy to manage errors
  - Handle the errors in the layer where they are detected or propagate to the immediately higher layer.

Pros and Cons

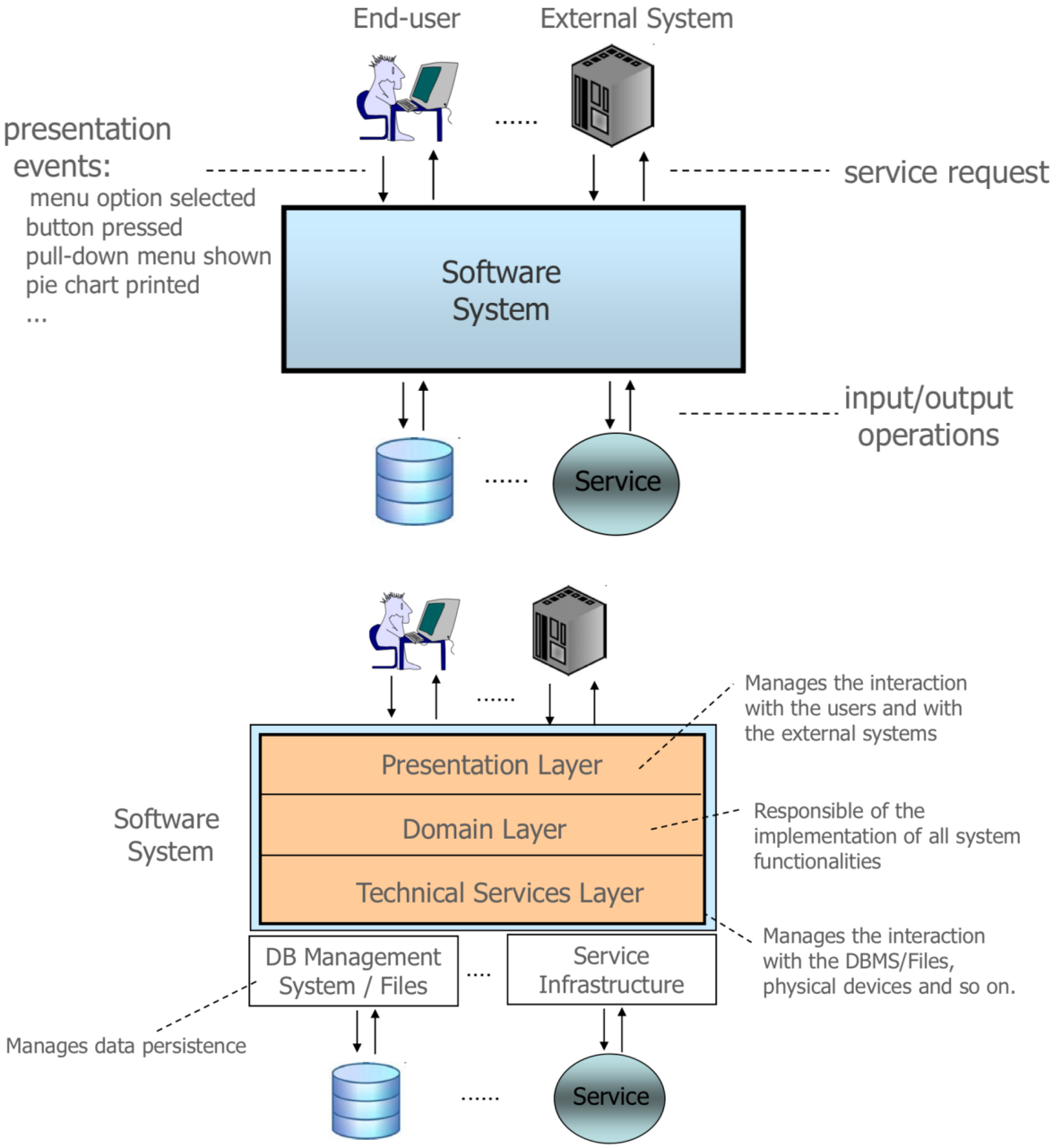
Pros:

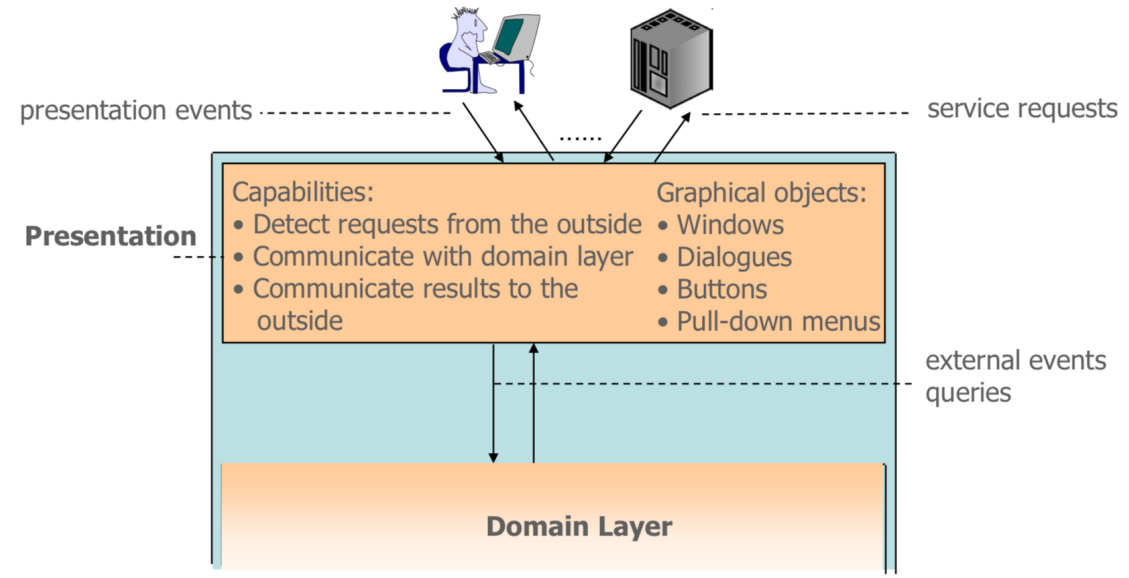
- Maintainable
- Reusable
- Portable
- Testable

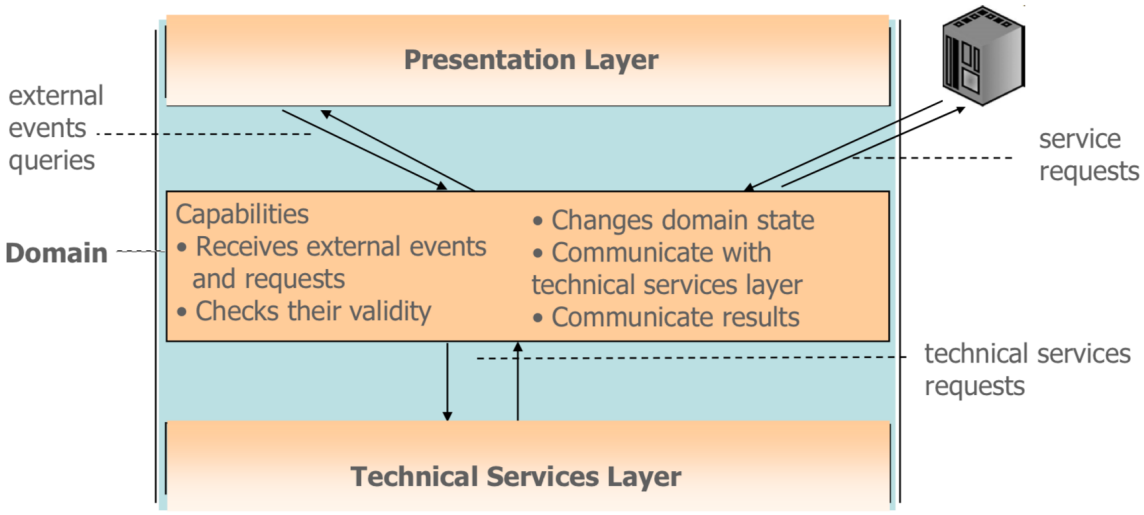
Cons:

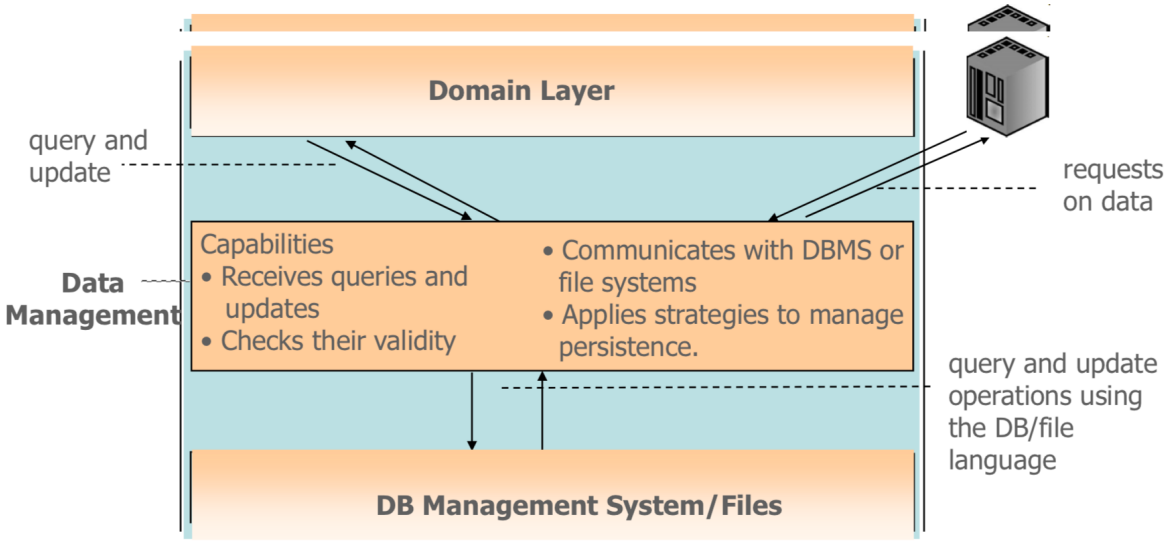
- Efficiency
- Unnecessary or redundant work
- Difficulty to establish the granularity and determine the number of layers.

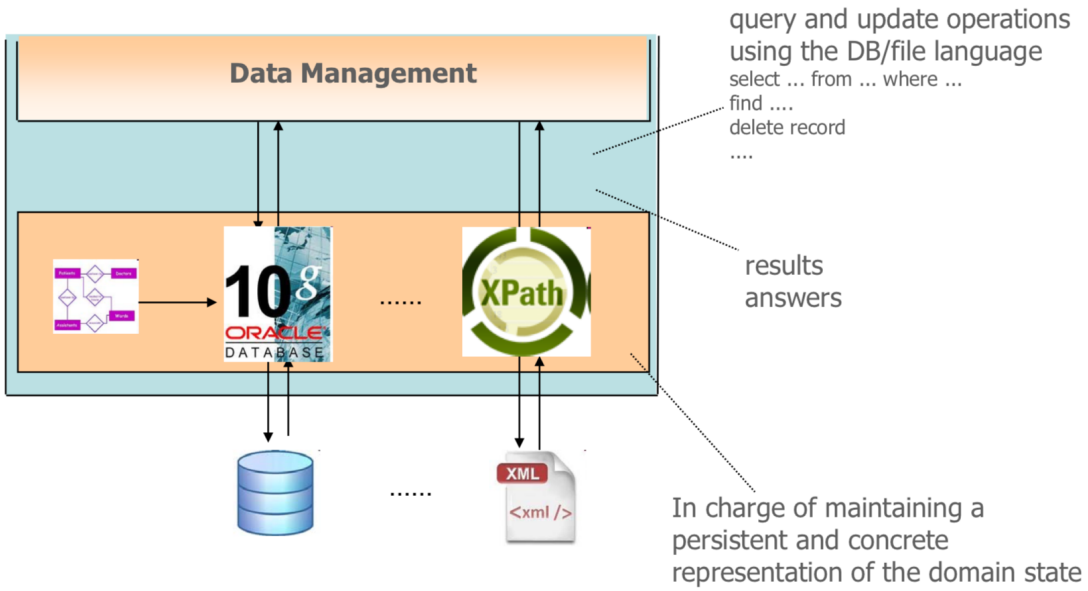
Application to Software Systems




presentation events
service requests
Capabilities:
• Detect requests from the outside
• Communicate with domain layer
• Communicate results to the outside
Graphical objects:
• Windows
• Dialogues
• Buttons
• Pull-down menus
Presentation
external events
queries
Domain Layer


Presentation Layer
external
events
queries
service
requests
Capabilities
• Receives external events and requests
• Checks their validity
• Changes domain state
• Communicate with technical services layer
• Communicate results
Domain
technical services
requests
Technical Services Layer





Layered architecture in UML

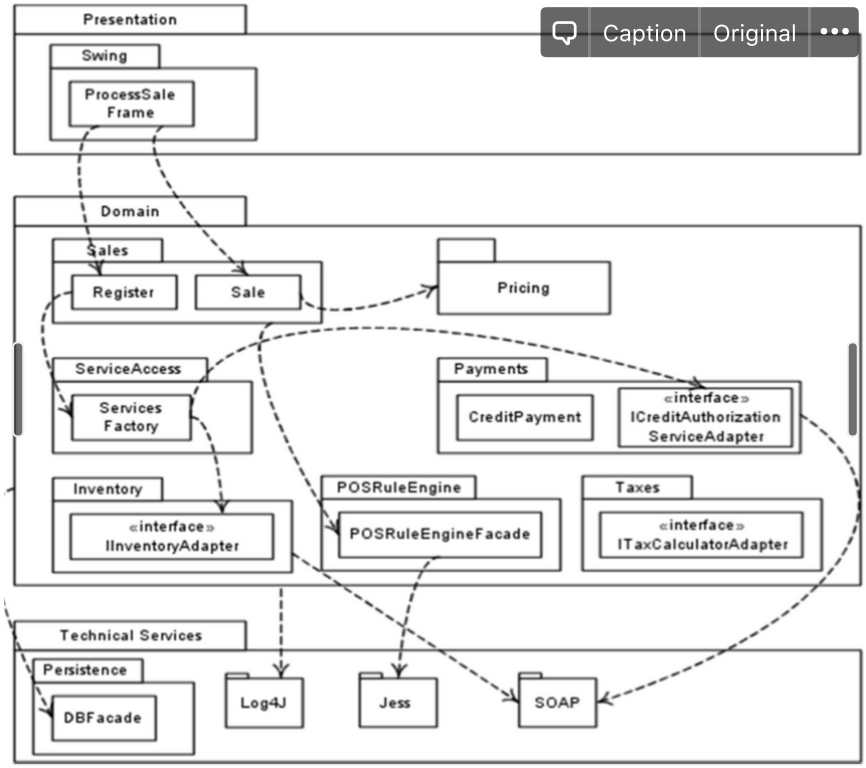
- UML package diagrams are often used to illustrate the layered architecture of a system. A layer can be modeled as a UML package.
- A UML package diagram provides a way to group elements as classes, other packages and so on.
- Packages may have dependency relationships to other packages. The UML dependency line is used for this.
- A dependency indicates a situation in which a change to the supplier element may require a change in the client element.

Layered Architecture Example

- NextGen is a point-of-sale system (POS) used to record sales and handle payments.
- It is typically used in a retail store. It includes hardware components as a computer and bar code scanner; and software to run the system.

- It interfaces to various service applications, such as a third-party tax calculator and inventory control. These systems must be relatively fault-tolerance; that is even if remote services are temporarily unavailable (such as the inventory system), they must still be capable of capturing sales and handling at least cash payments.
- A POS system increasingly must support multiple and varied client-side terminals and interfaces.

Layered architecture of POS system and UML Notation:



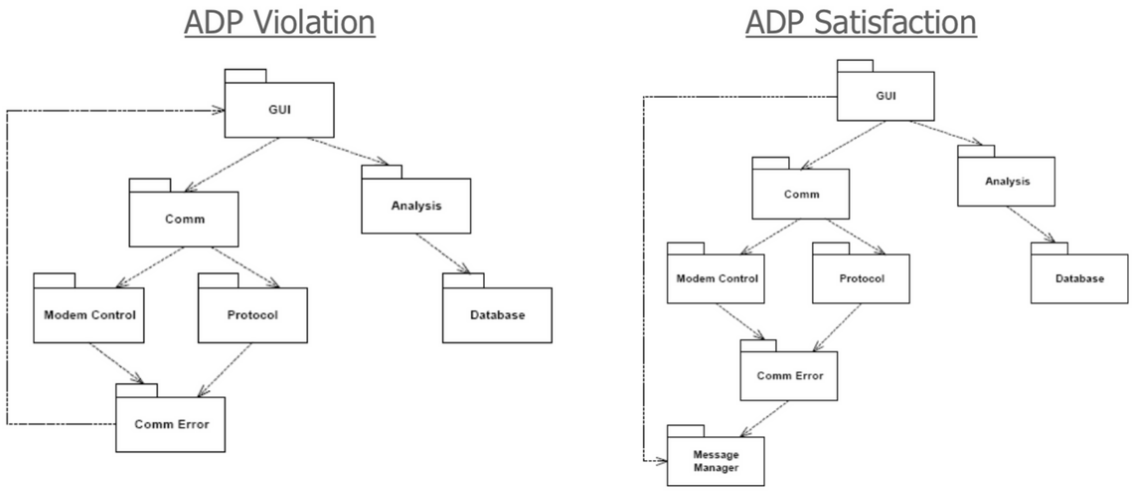
Layered Architecture Design Principles

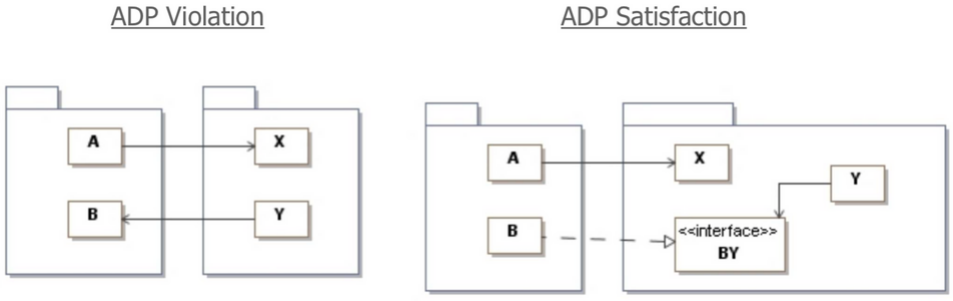
Package Cohesion Principles:

- The Release Reuse Equivalence Principle (**REP**): The granule of reuse is the granule of release (*Reusability*).
- The Common Closure Principle (**CCP**): Classes that change together, belong together (*Maintenance*).
- The Common Reuse Principle (**CRP**): Classes that are not reused together should not be grouped together (*Reusability*).

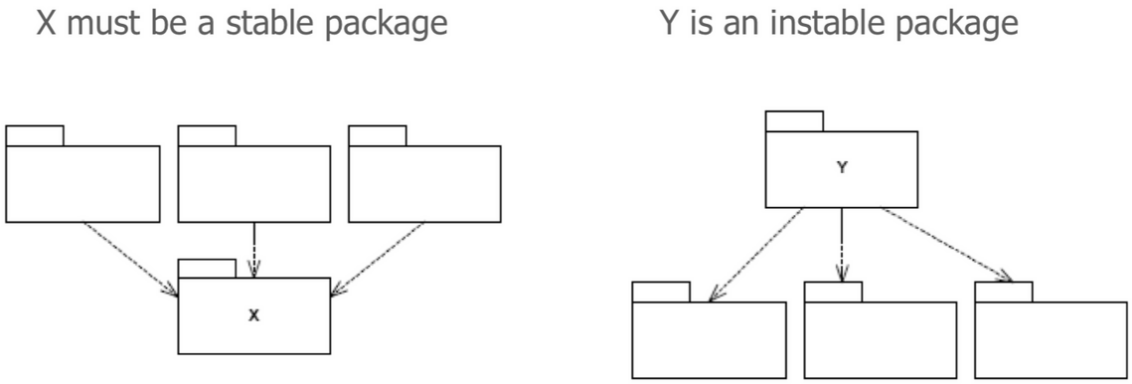
Package Coupling Principles:

- The Acyclic Dependencies Principle (**ADP**): The dependencies between packages must not form cycles.





- The Stable Dependencies Principle (**SDP**): Depend in the direction of stability.



- The Stable Abstractions Principle (**SAP**): Stable packages should be abstract packages.