CP317A Software Engineering

High-level design, part-2 – week 3-2

Shaun Gao, Ph.D., P.Eng.

Agenda

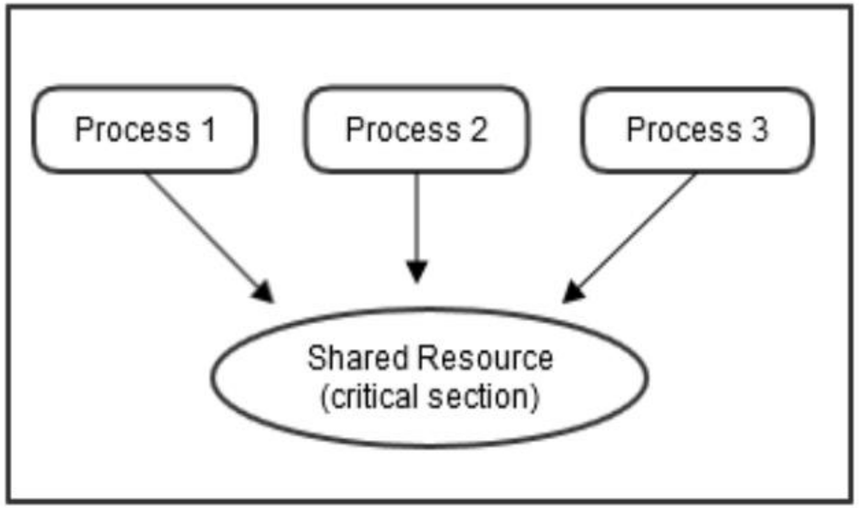
- Review week 3 - 1
- Race condition
- Unified modeling language (UML)
- UML diagrams
  - Structure diagram - Class diagrams
  - Behavior diagram – use case diagram, state transition diagrams, Sequence diagrams
- Design principles (3 principles)
  - Decomposition (divide and conquer)
  - Cohesion
  - Coupling
- Summary

Review week 3-1

- Software security
  - Concept
- User interface
  - Concept
- Architecture design
  - Monolithic
  - Client/server
  - Component-based
  - Service-oriented
  - Data-centric
  - Event-driven
  - Distributed

Race condition

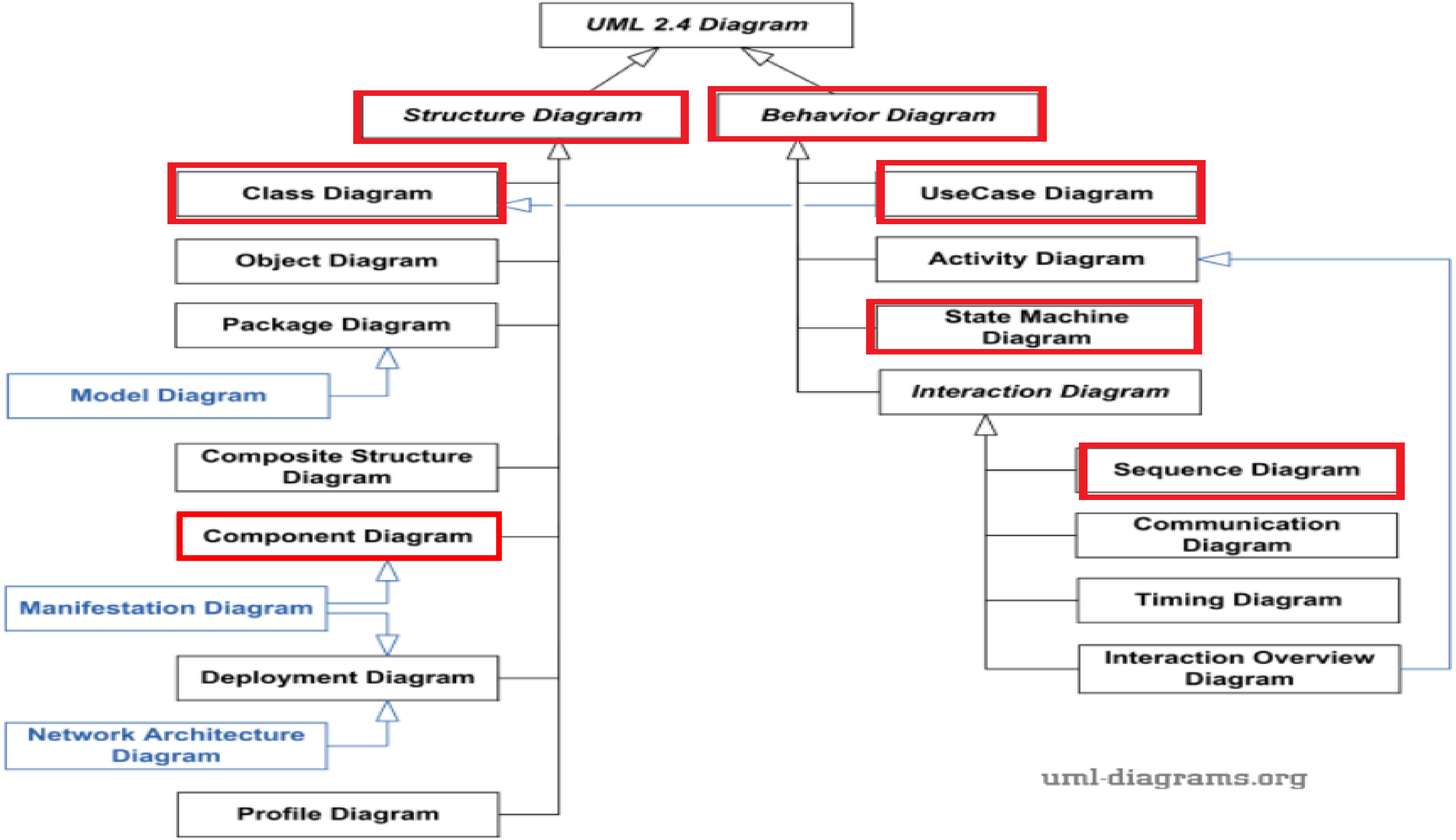
- Event-driven architecture or distributed architecture can cause race condition
- A race condition is an undesirable situation that occurs when more than one process attempt to modify the same object at the same time.
- What is the problem of a race condition?
  - lost data consistency
- Prevention techniques:
  - Mutual exclusion
  - Semaphores



Unified modeling language (UML)

- UML
  - Definition: UML is a general-purpose, developmental, modeling language in the field of software engineering that is intended to provide a standard way to visualize the design of a system.
  - Note: UML is not actually a single unified language. Instead, it defines several kinds of diagrams that is used to represent the system.
  - UML version 2.0 defines 13 diagrams

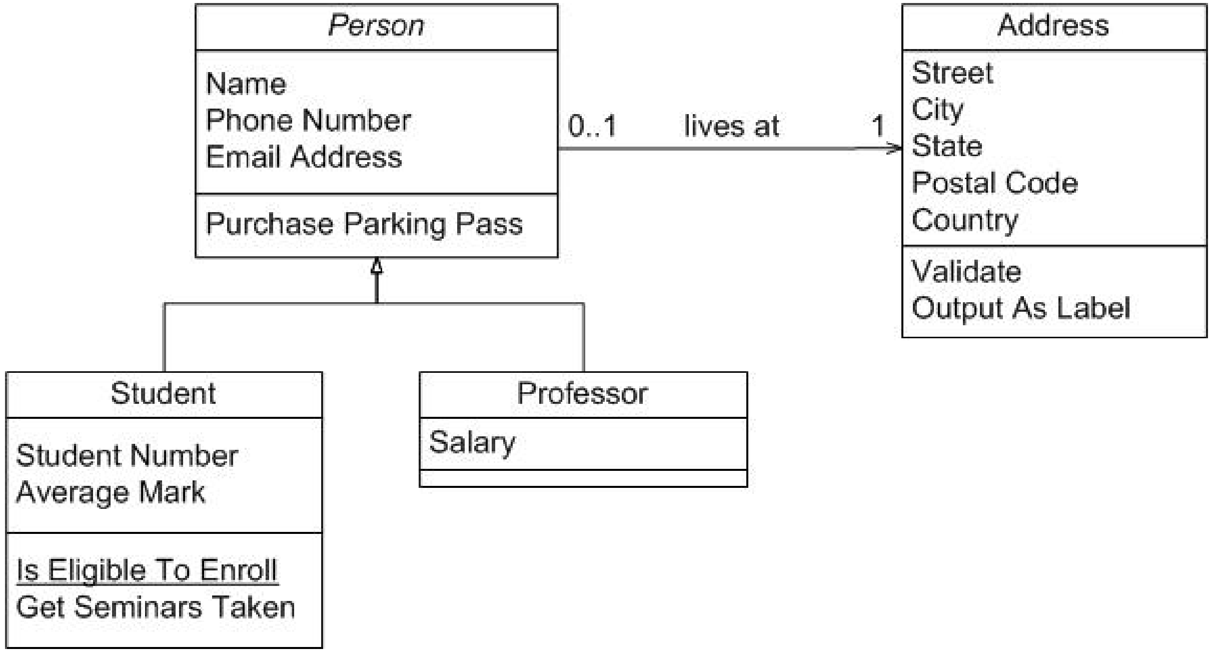
UML diagrams



uml-diagrams.org

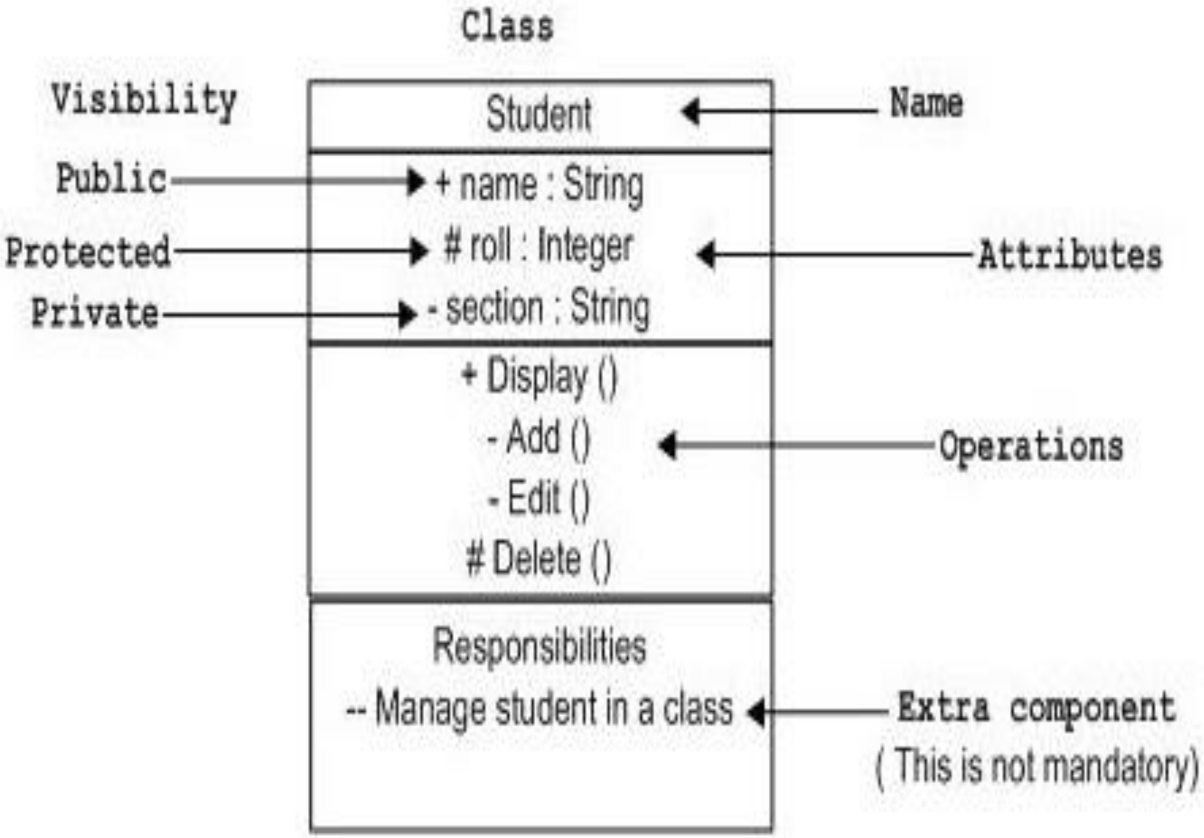
UML: Structure diagrams – cont.

- Structure diagram: A structure diagram is a diagram that show the static structure of the system and its parts on different abstraction and implementation levels and how they are related to each other.
- Examples:
  - Class diagram
  - …
    - Exactly one - 1
    - Zero or one - 0..1
    - Many - 0..* or *
    - One or more - 1..*



UML: Structure diagrams – cont.

- Class diagram symbols



Class

Visibility

Public

Protected

Private

Student

+ name : String

roll : Integer

- section : String

+ Display ()

- Add ()

- Edit ()

Delete ()

Responsibilities

-- Manage student in a class

Name

Attributes

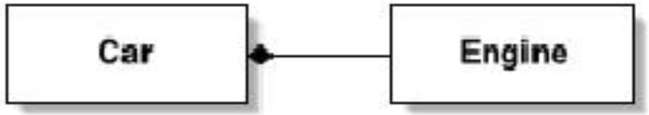
Operations

Extra component

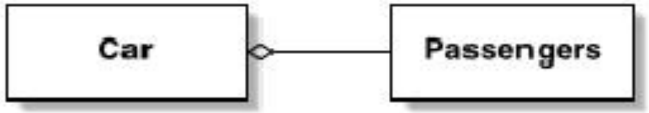
(This is not mandatory)

UML symbols

Association	Symbol
Composition	◆————
Aggregation	◇————
Inheritance	————▷
Implementation	- - - - - - - - -▷



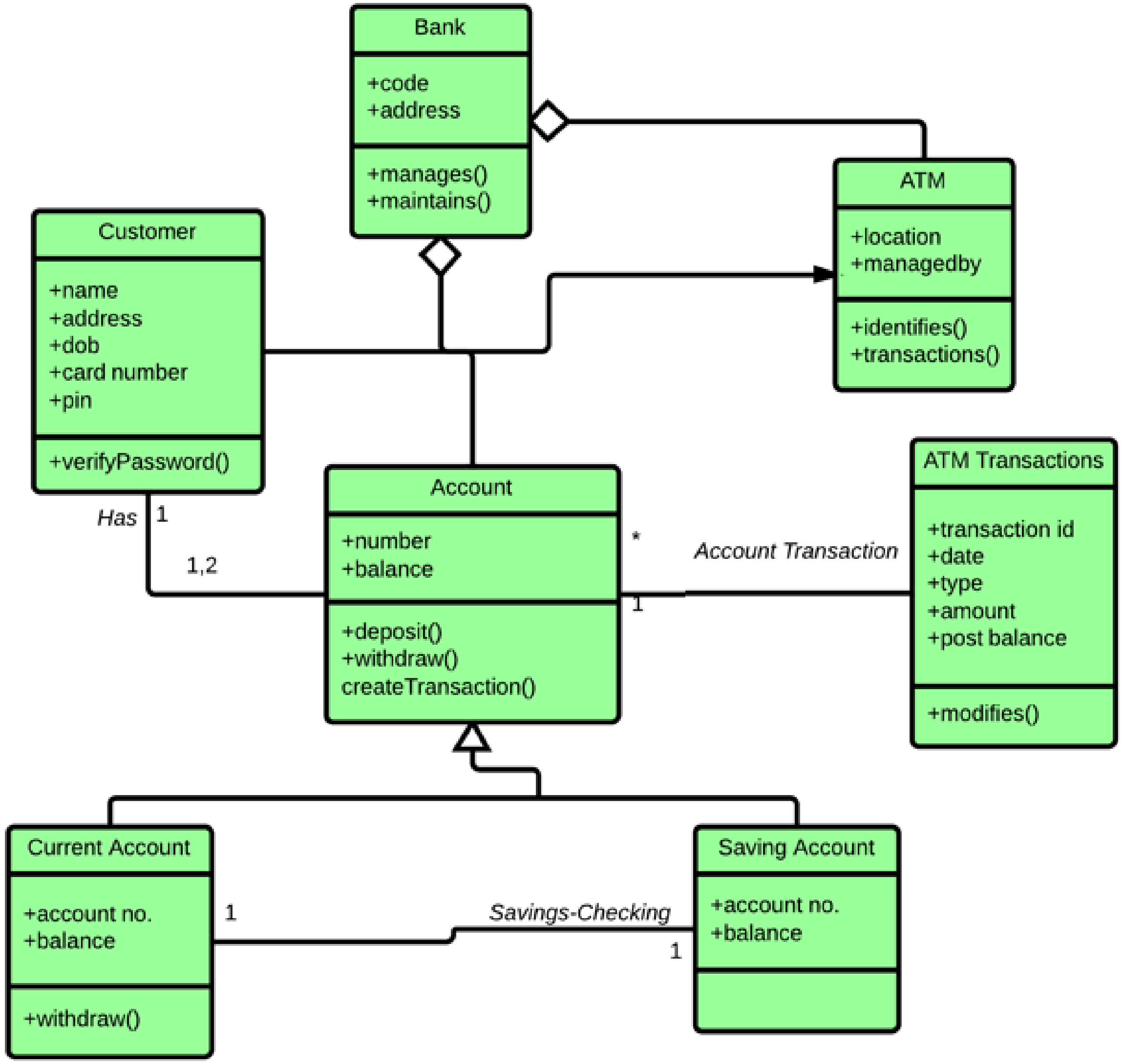
Composition: every car has an engine.



Aggregation: cars may have passengers, they come and go

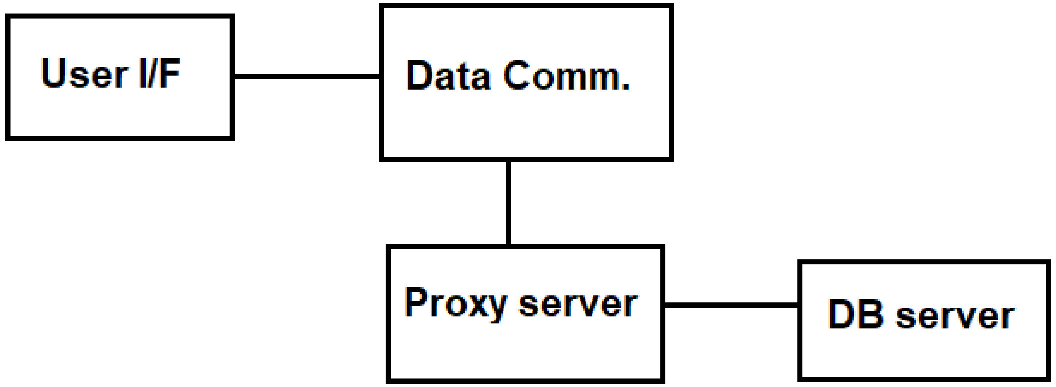
UML: Structure diagrams – cont.

- Class diagram example



UML: Structure diagrams – cont.

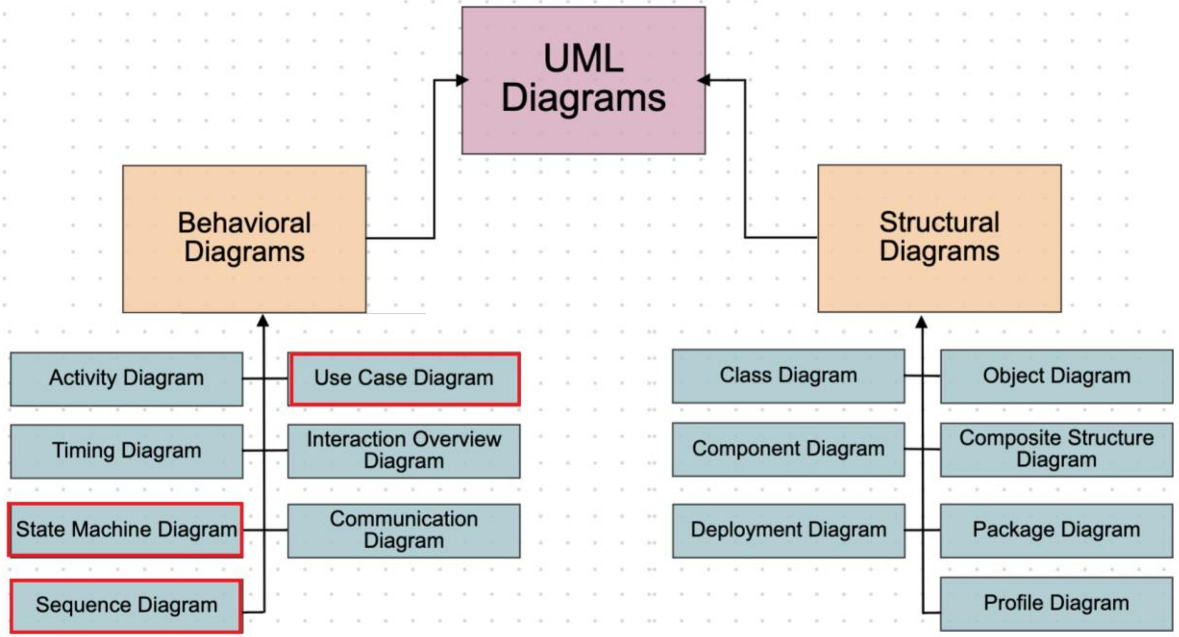
- Component diagram
  - Component diagrams are used in modeling a system, which are used for visualizing, specifying, and documenting all possible components in a system.
- Examples:
  - ATM



UML: Behavior diagrams

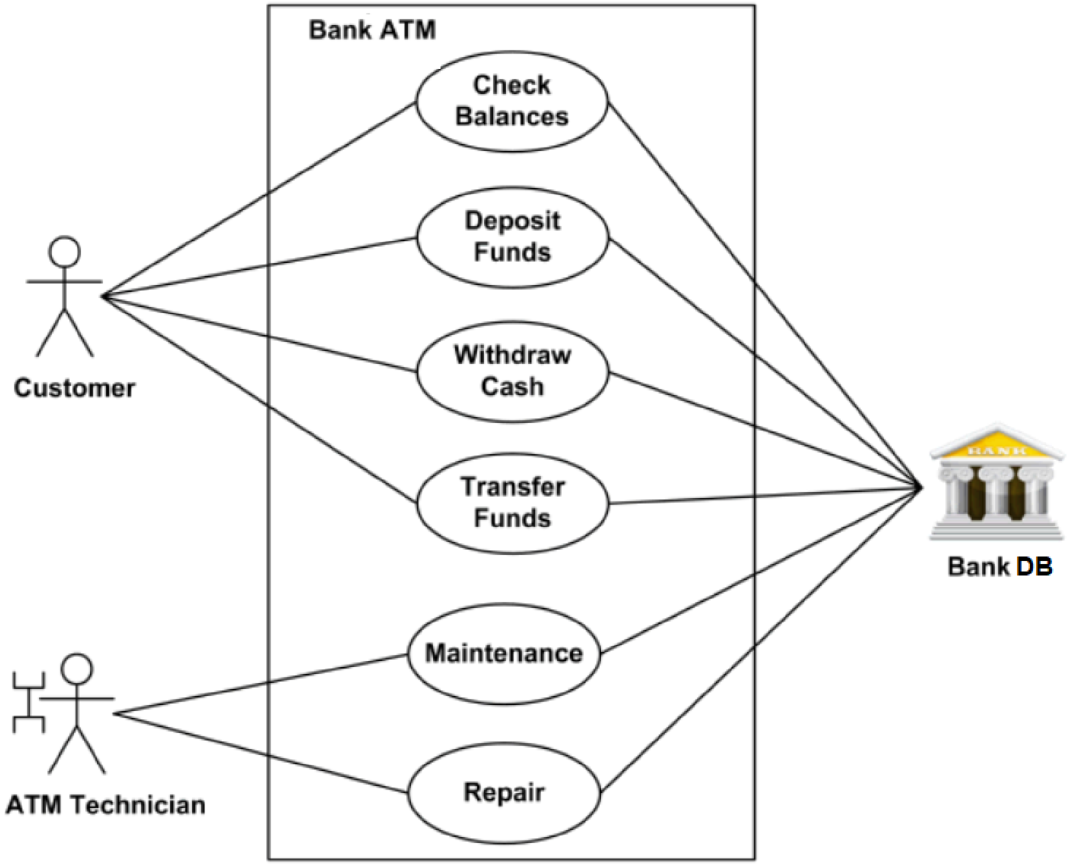
- Behavior diagram: A behavior diagram is a diagram that show the dynamic behavior of the objects in a system, which can be described as a series of changes to the system over time.

- Examples:
  - Use case diagram
  - State machine diagram
  - Sequence diagram
  - Interaction diagram
  - …



UML: Behavior diagrams – cont.

- Use-case diagrams
- A use-case diagram is a diagram that consists of actors, use cases and their relationships among the actors and the use cases.
- The diagram is used to model the system/subsystem of an application. A single use case diagram captures a functionality of a system.
- Example: ATM

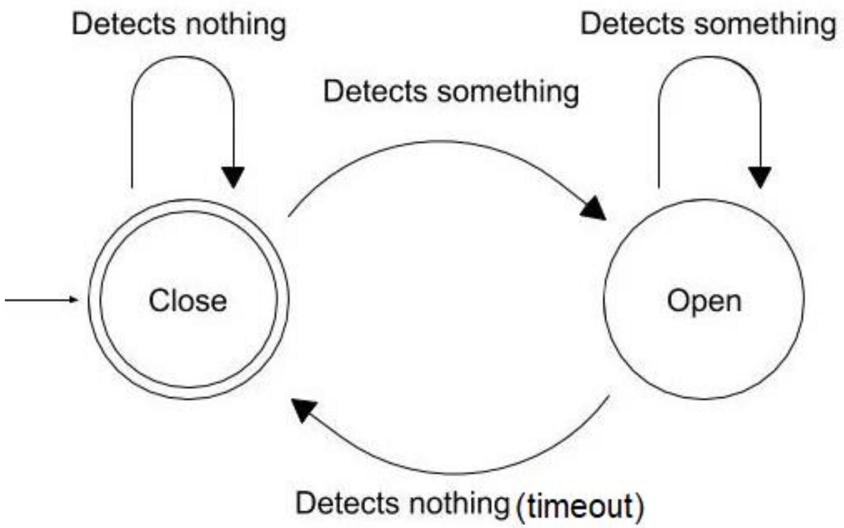


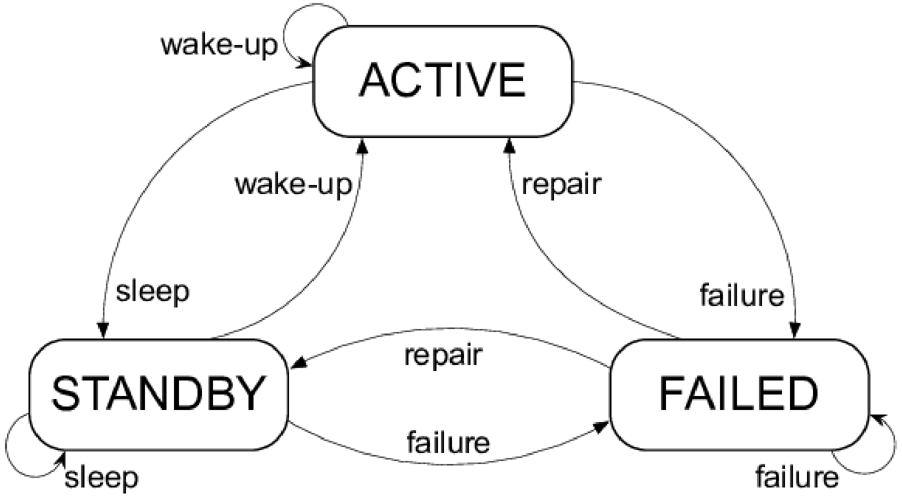
UML: Behavior diagrams – cont.

- Use-case diagrams – cont.
- The purposes of use case diagrams:
  - To gather the requirements of a system.
  - To get an outside view of a system.
  - Identify the external and internal factors influencing the system.
  - Show the interaction among the requirements.

UML: Behavior diagrams – cont.

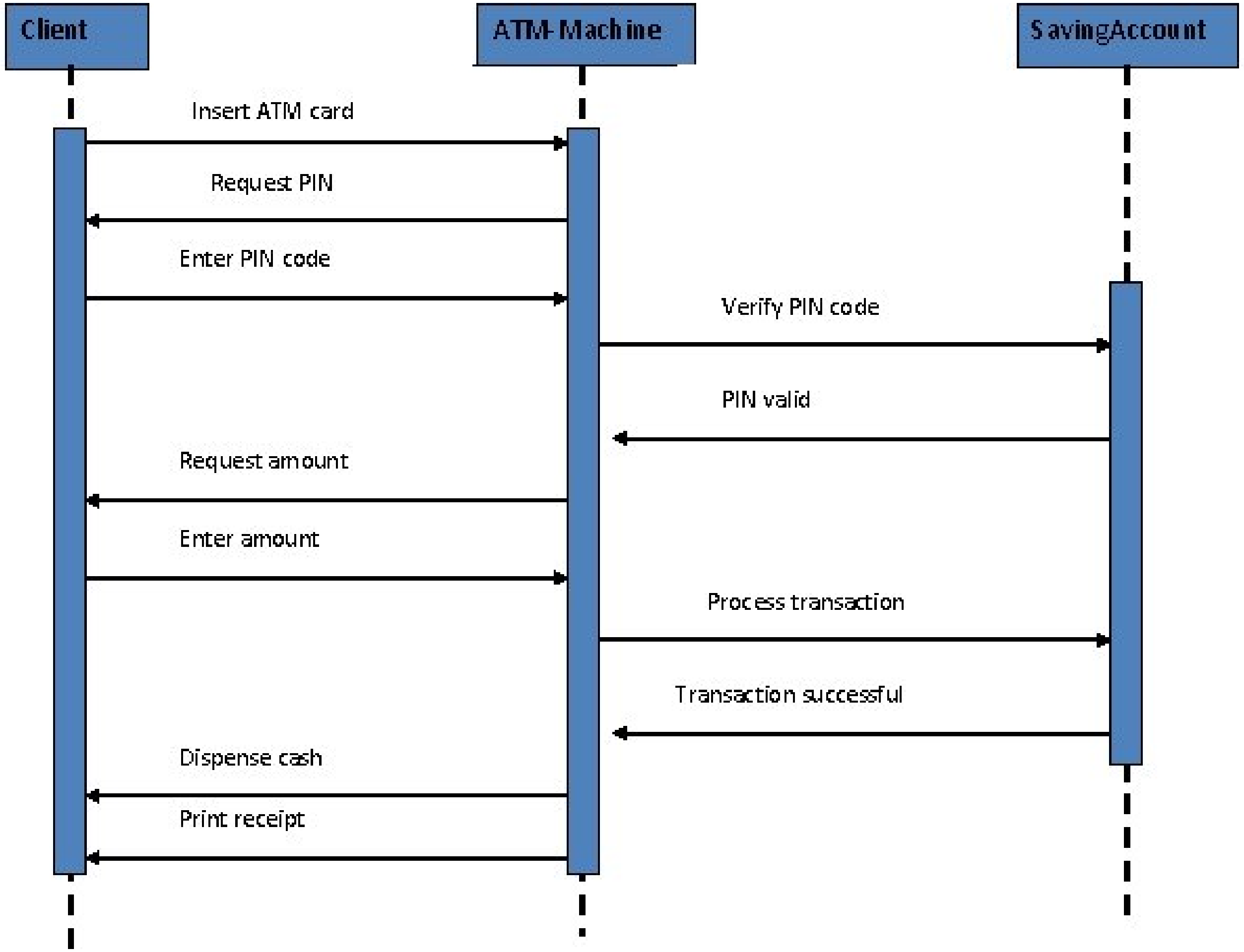
- State machine/transition diagram
  - A state transition diagram is a diagram that is used for modeling discrete behavior through finite state transitions.
  - It describes the behavior of the system.
  - It consists of states and events
  - Examples:
    - Automatic door system
    - Operating systems
    - TCP data communications





UML: Behavior diagrams – cont.

- Sequence diagrams
  - A **sequence diagram** is a diagram that shows object interactions arranged in **time sequence.**
  - It depicts interactions between objects in a **sequential** order i.e. the order in which these interactions take place.
  - It contains (1) objects, (2) interactions, (3) timeline.

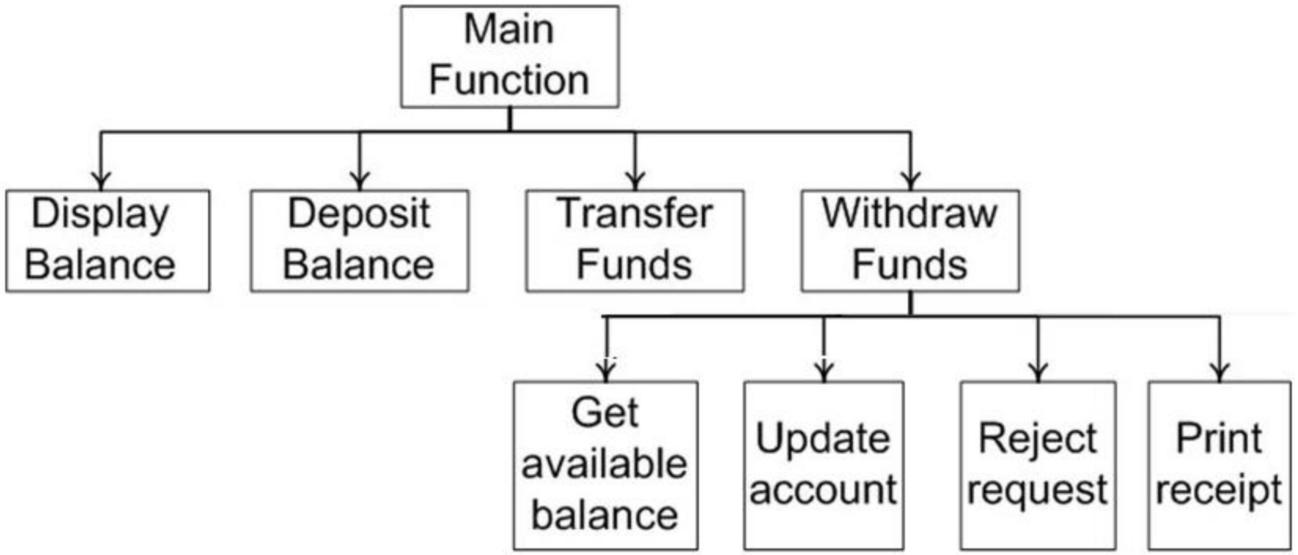


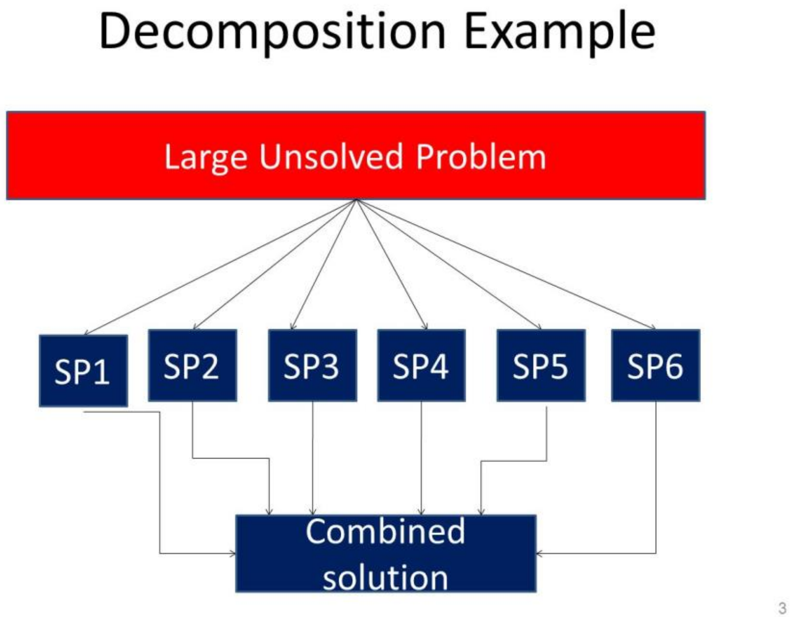
Think – Pair – Share

- Structure diagram
  - Class diagrams
  - Component diagrams
- Behavior diagram
  - Use case diagram,
  - State transition diagrams,
  - Sequence diagrams
- For the group project, which diagram(s) can be used? And why?

Design principle 1: decomposition

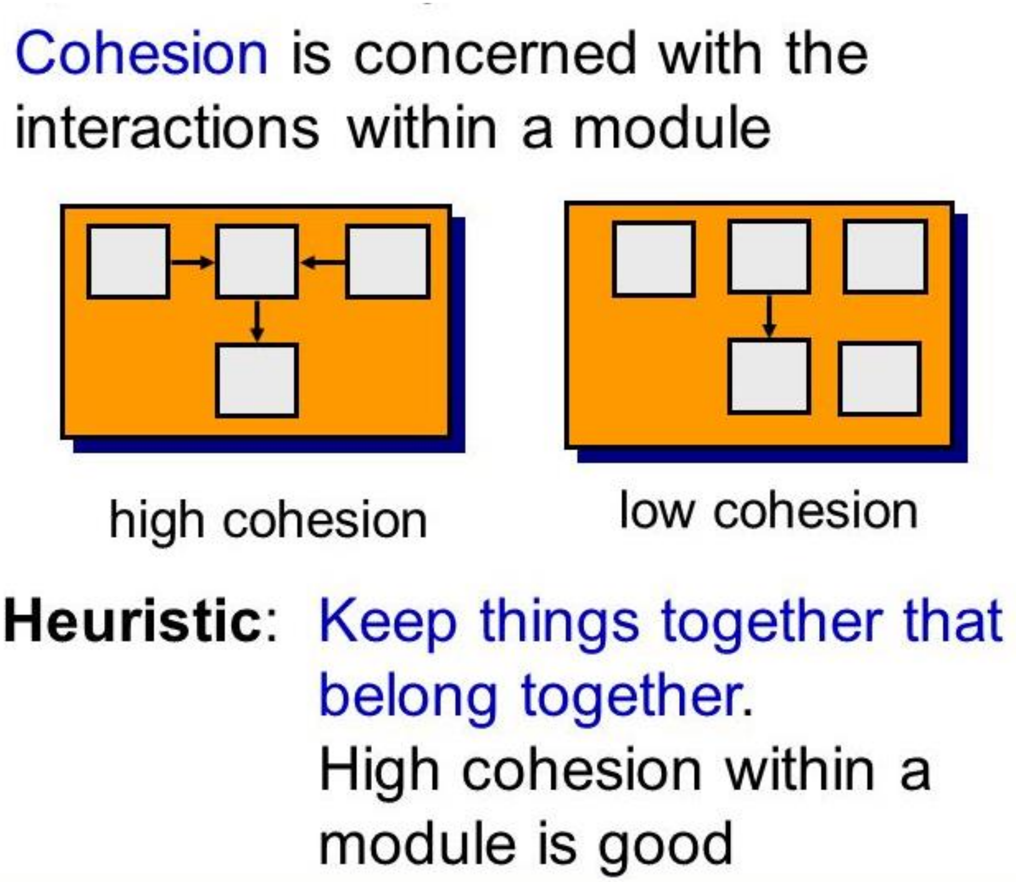
- Decomposition
  - Decomposition is also known as **factoring**, is breaking a complex problem or system into parts that are easier to conceive, understand, program, build, and maintain.
- An example:
  - ATM




Decomposition Example

Design principle 2: Cohesion

- Cohesion
  - Cohesion is a measure that defines the degree of intra-dependability among elements of a module.
  - The greater the cohesion, the better the software design.
- Example:
  - ATM



Cohesion is concerned with the interactions within a module

high cohesion

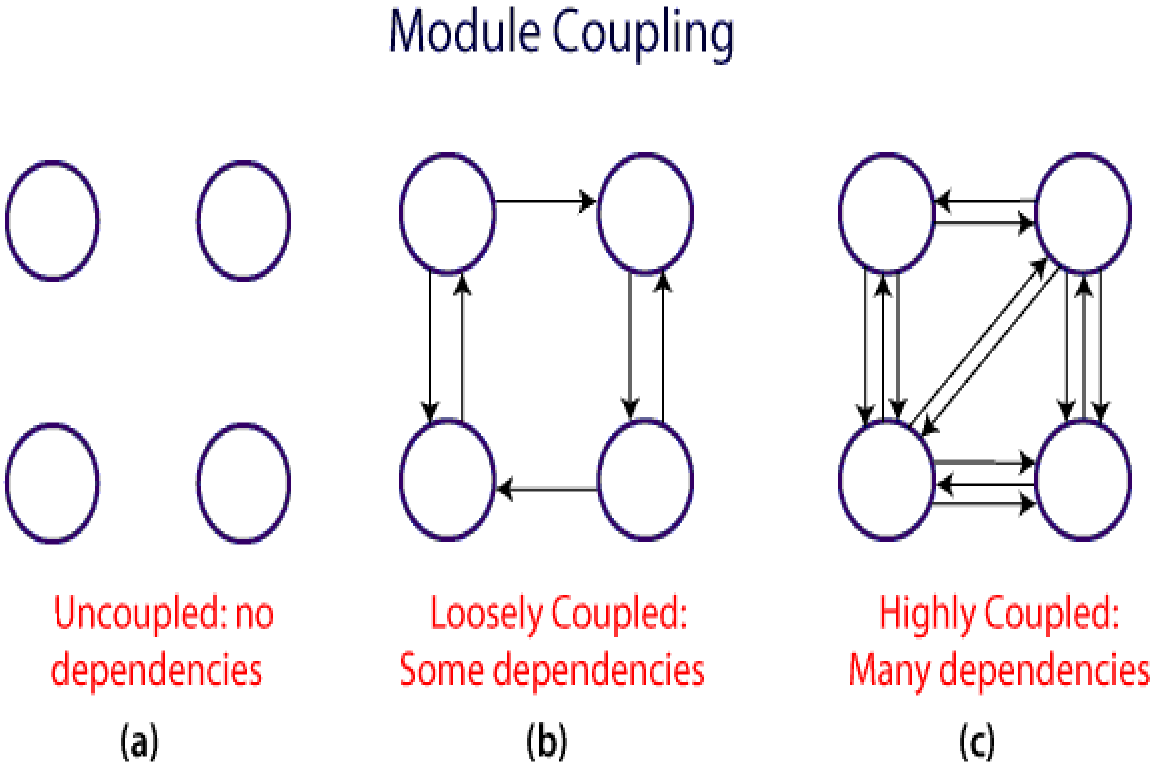
low cohesion

Heuristic: Keep things together that belong together.

High cohesion within a module is good

Design principle 3: Coupling

- Coupling
  - Coupling occurs when there are interdependencies between one component/module and another.
  - When interdependencies exist, changes in one place will require changes somewhere else.
  - The lower coupling the better software design



Module Coupling

Uncoupled: no dependencies

(a)

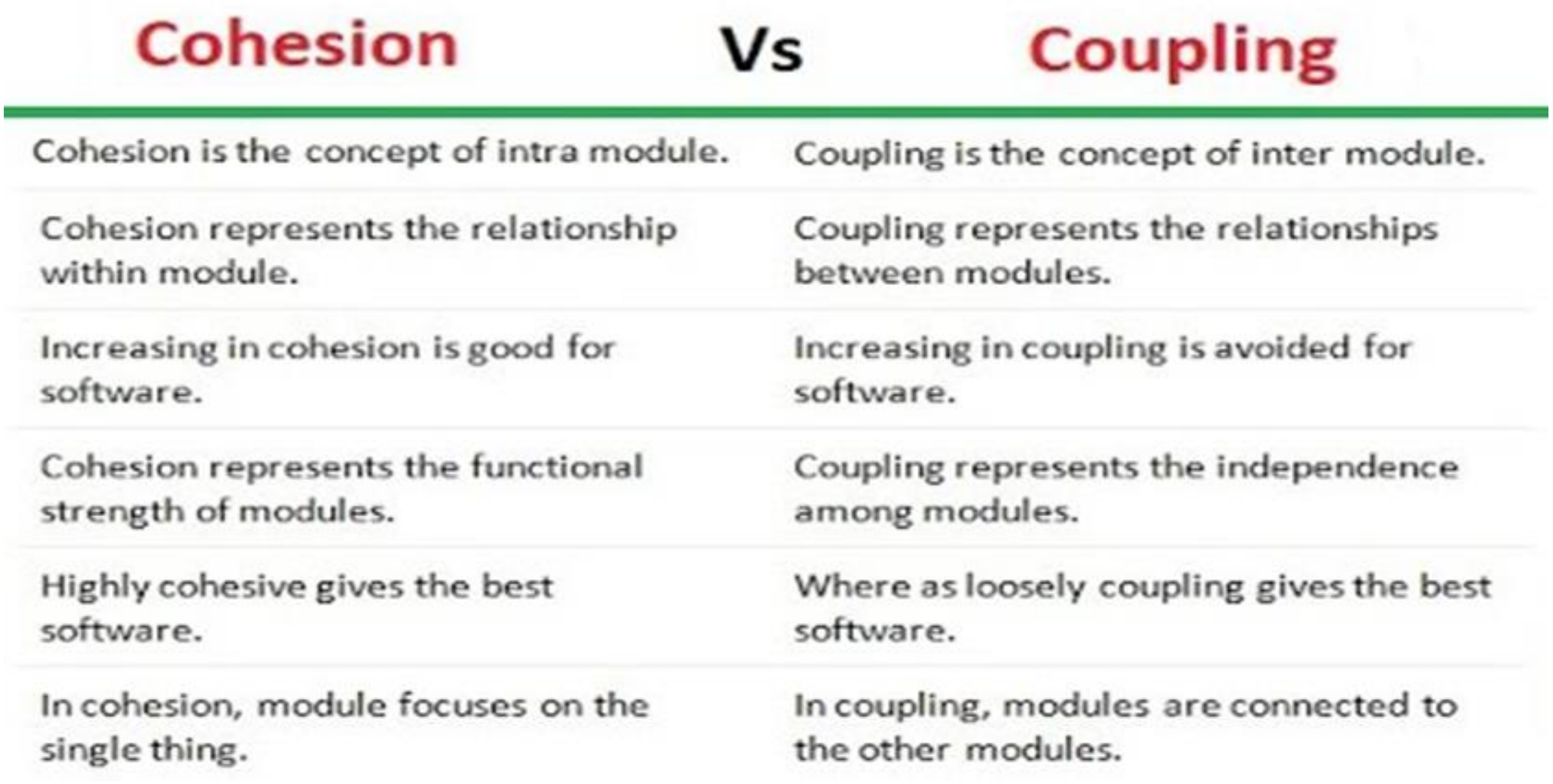
Loosely Coupled: Some dependencies

(b)

Highly Coupled: Many dependencies

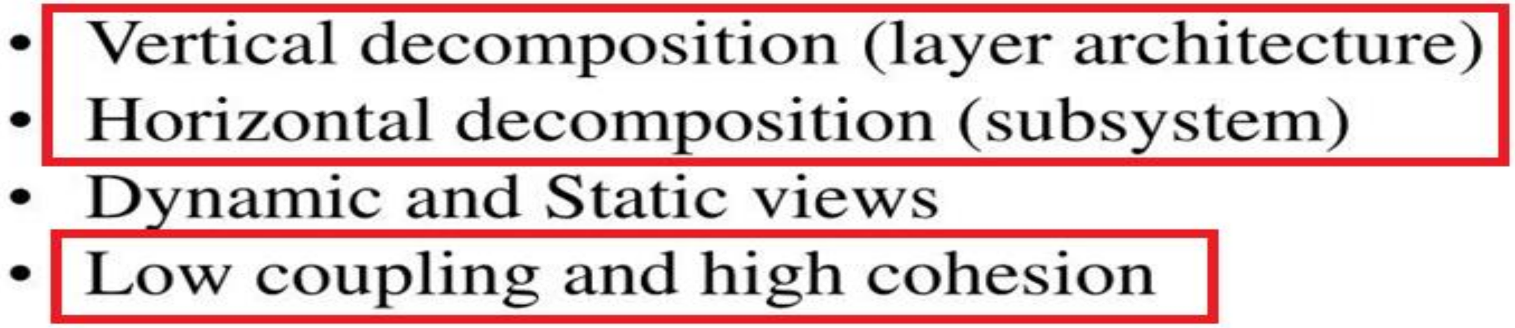
(c)

Cohesion vs. Coupling

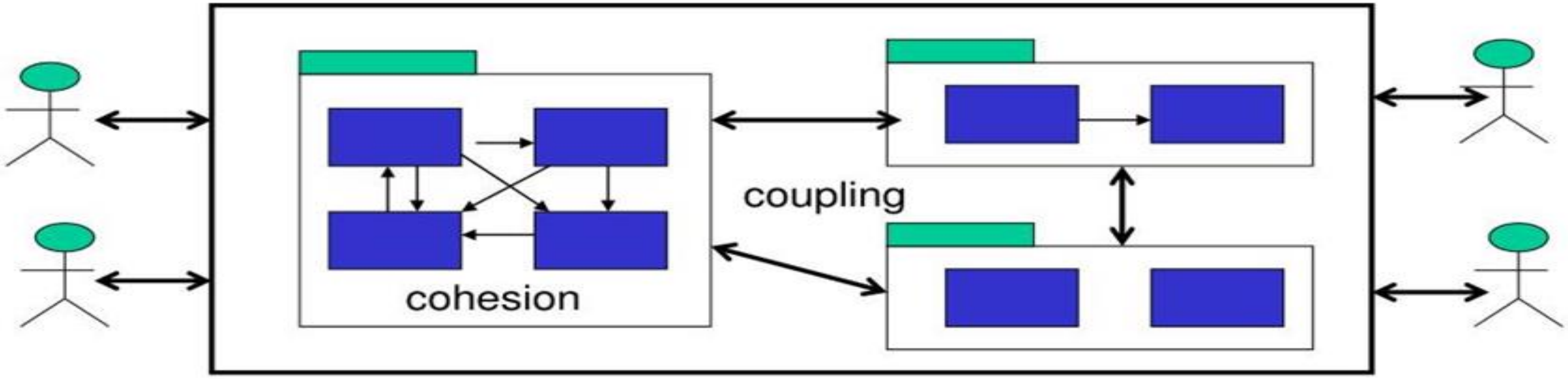


Cohesion	Vs	Coupling
Cohesion is the concept of intra module.		Coupling is the concept of inter module.
Cohesion represents the relationship within module.		Coupling represents the relationships between modules.
Increasing in cohesion is good for software.		Increasing in coupling is avoided for software.
Cohesion represents the functional strength of modules.		Coupling represents the independence among modules.
Highly cohesive gives the best software.		Where as loosely coupling gives the best software.
In cohesion, module focuses on the single thing.		In coupling, modules are connected to the other modules.

Design principal summary



- Vertical decomposition (layer architecture)
- Horizontal decomposition (subsystem)
- Dynamic and Static views
- Low coupling and high cohesion



Summary

- Race condition
- Unified modeling language (UML)
- UML diagrams
  - Structure diagram - Class diagrams, component diagrams
  - Behavior diagram – use case diagram, state transition diagrams, Sequence diagrams
- Design principles (3 principles) – **also apply to detailed design**
  - Decomposition
  - Cohesion
  - Coupling

Announcement

- Please start the group project from writing the project report (SDD)

- Please let me know if you need help for finding a group

- Low level design from next week