JURUSAN TEKNOLOGI INFORMASI

Software Engineering Course

07. Design (Part-1).pptx

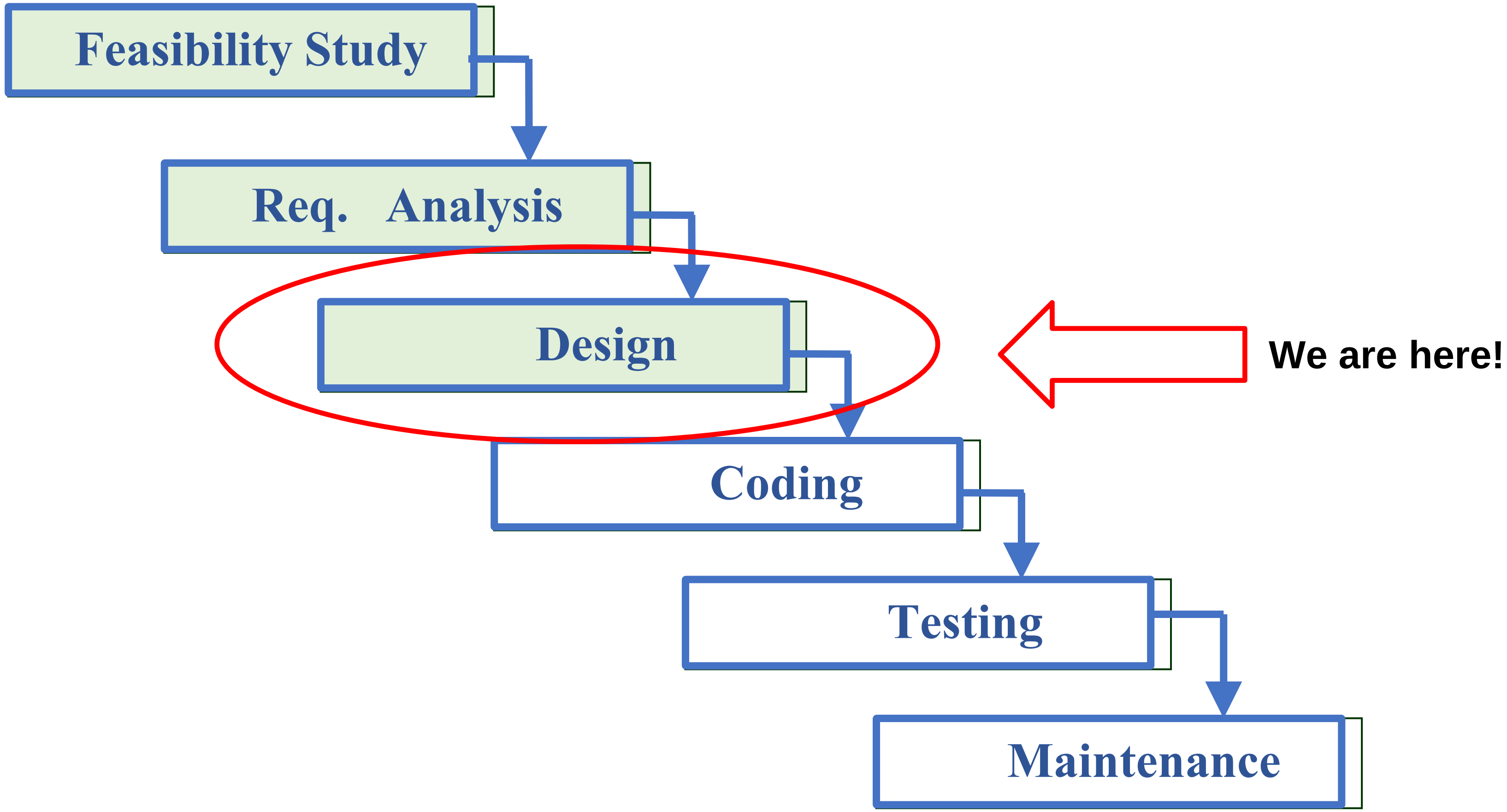
Yoppy Yunhasnawa, S.ST., M.Sc.

Topics

1. Software Design Basics
2. UML
3. Use Case Detailing

Topic #1: Software Design Basics

1. Software Design Basics



1. Software Design Basics

- Software design is a process to transform user requirements into some suitable form, which helps the programmer in software coding and implementation. [1]

- Requirements Analysis output → SRS Document [2]; Must be detailed to assist coding and implementation.
  - That's why design process is needed.
  - The output of this process can directly be used into implementation in programming languages.

- It is the first phase that moves from "problem" domain to "solution" domain.

- Primary goal: Tries to specify *"how"* to fulfill requirements in SRS Document.

1. Software Design Basics

Software Design Level

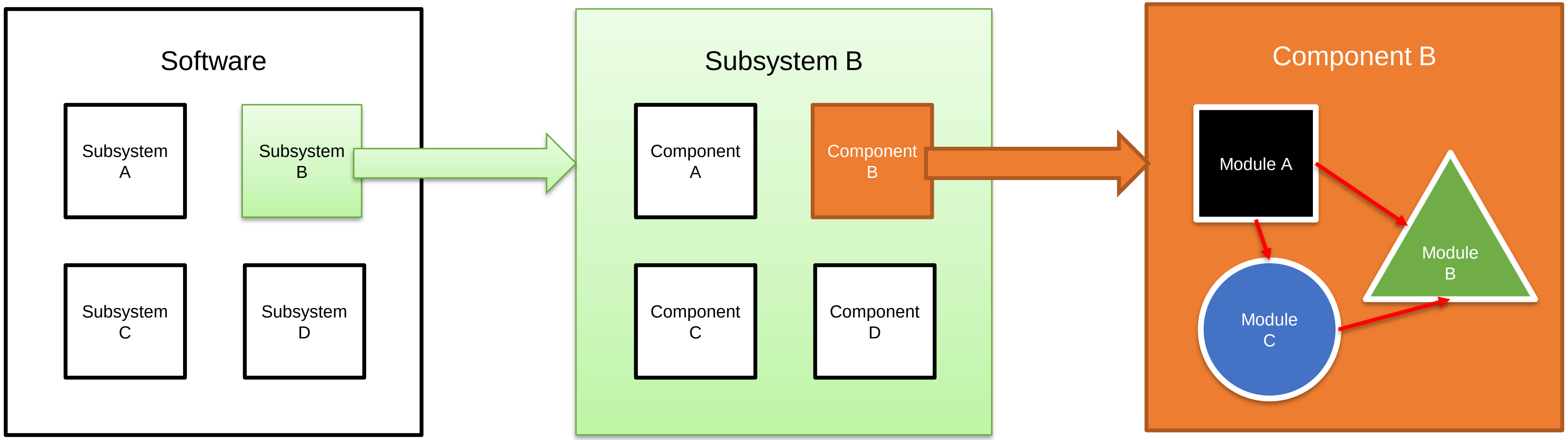
- **Architectural** Design
  - Highest abstract version of the system
  - It identifies the software as a system with many components interacting with each other.
  - At this level, the designers get the idea of proposed solution domain.

- **High-level** Design
  - Less-abstracted view of sub-systems and modules and depicts their interaction with each other
  - Focuses on how the system along with all its components can be implemented in forms of modules.

- **Detailed** Design
  - Detailed design deals with the implementation part of the previous two design levels.
  - More detailed towards modules and their implementations.
  - It defines logical structure of each module and their interfaces to communicate with other modules.

1. Software Design Basics

Software Design Level



Architectural Level

High Level

Detailed Level

1. Software Design Basics

Modularization

- Modularization is a technique to divide a software system into multiple discrete and independent modules.
  - Which are expected to be capable of carrying out task(s) independently.

- Modules work as basic constructs for the entire software.
  - They should be able to be executed and/or compiled separately and independently.

- It follows "**divide and conquer**" problem solving strategy.

- Advantage of modularization:
  - Smaller components are easier to maintain
  - Program can be divided based on functional aspects
  - Desired level of abstraction can be brought in the program
  - Components with high cohesion can be re-used again
  - Concurrent execution can be made possible
  - Desired from security aspect

1. Software Design Basics

Modularization

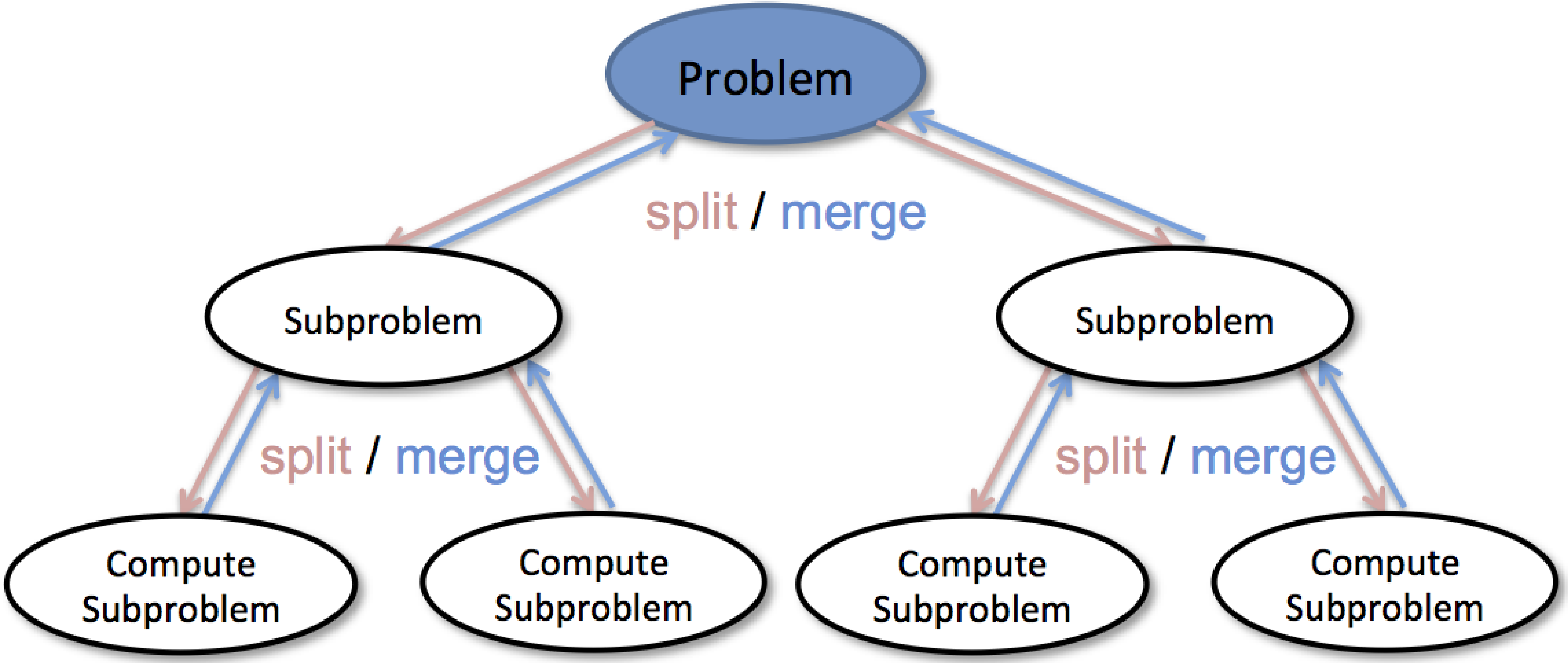


Figure: Divide and Conquer. [3]

1. Software Design Basics

Concurrency

- In previous time, software can only be executed sequentially.
  - One coded instruction executed at one time before any others.

- More modern hardware now have capacity to execute instruction in parallel manner.
  - Multiple smallest unit of instructions (thread) can be executed in one time.
  - It dramatically increase the responsiveness of the software.

- That is why, modern software should be able to work utilize the parallel capability of their hardware host.

- It is necessary for the programmers and designers to recognize the modules as units made to enable parallel execution.

- **Example**
  - The spell check feature in word processor is a module of software, which runs along side the word processor itself.

Coupling and Cohesion

- When software is modularized, there will be aspects known as coupling and cohesion.

- Module are set of instructions put together to achieve certain task.
  - They are single entities.
  - But may refer to each other to work together.

- There are measures by which the quality of a design of modules and their interaction among them can be measured.
  - These measures are called coupling and cohesion.

- **Cohesion**: Intra dependability within elements of a module → The greater the better.

- **Coupling**: Inter dependability among modules of a program. → No coupling is the best.

Design Verification

- The output of software design process are:
  - Design documentation,
  - Pseudo codes
  - Detailed logic diagrams
  - Process diagrams, and
  - Detailed description of all functional or non-functional requirements.

- The next phase, which is the implementation of software, depends on all outputs mentioned above.

- It is necessary to **verify the output** before proceeding to the next phase.
  - The early any mistake is detected, the better.
  - Or it might not be detected until testing of the product.

- By structured verification approach, reviewers can detect defects that might be caused by overlooking some conditions.
  - A good design review is important for good software design, accuracy and quality.

Topic #2: **UML**

2. UML

- UML → Unified Modeling Language
- Is a standardized modeling language consisting of an integrated set of diagrams, developed to help system and software developers
  - Specifying, visualizing, constructing, and documenting the artifacts of software systems
  - Business modeling and other non-software systems.
- The UML represents a collection of best engineering practices that have proven successful in the modeling of large and complex systems.
- The UML is a very important part of **designing** & developing object-oriented software.
- Uses mostly graphical notations to express the **design of software projects**.
- Benefit → Help project team members to:
  - Communicate
  - Explore potential designs
  - Validate the architectural design of the software.

2. UML

Short History

- Proposal from OMT (Object Management Group) [1996] → Rational UML Partners Consortium work & collaborations → UML 1.0 [1997].

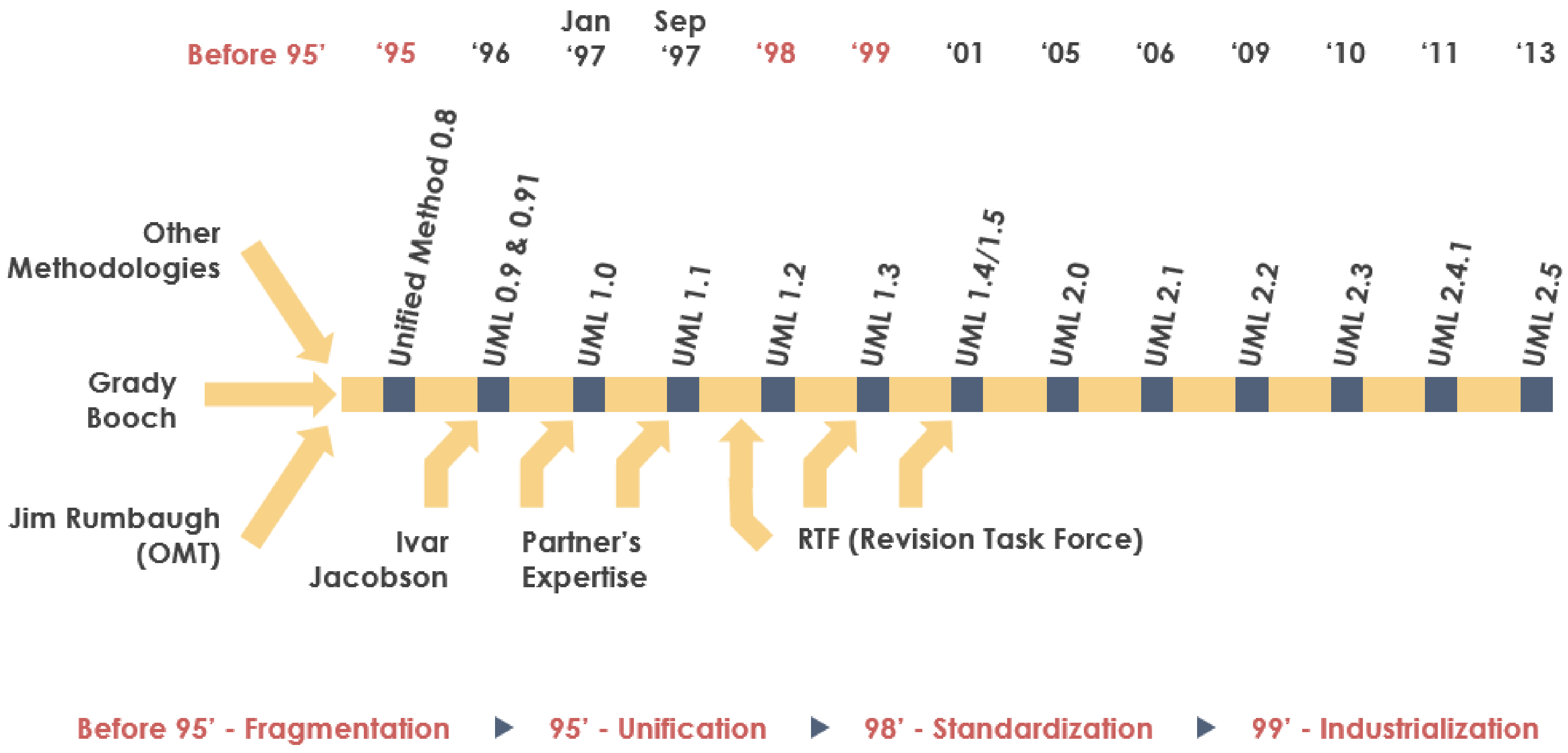


Figure: UML Timeline

2. UML

Importance

- Provide users with a ready-to-use, expressive visual modeling language so they can develop and exchange meaningful models.
- Provide extensibility and specialization mechanisms to extend the core concepts.
- Be independent of particular programming languages and development processes.
- Provide a formal basis for understanding the modeling language.
- Encourage the growth of the OO tools market.
- Support higher-level development concepts such as collaborations, frameworks, patterns and components.
- Integrate best practices.

2. UML
Overview

- The first thing to notice about the UML → A lot of different diagrams (models)!
- WHY?? Because software has many different viewpoints:
  - Analysts
  - Designers
  - Coders
  - Testers
  - QA
  - The Customer
  - Technical Authors
- All of these people are interested in different aspects of the system, and each of them require a different level of detail.
  - A coder needs to understand the design of the system and be able to convert the design to a low-level code.
  - A technical writer is interested in the behavior of the system as a whole
- UML attempts to provide a language so expressive that all stakeholders can benefit from at least one UML diagram.

2. UML
Diagrams

- Again, UML has many diagrams (for many stakeholders of the system).
  - In general, those diagrams divided into **structure** and **behavior** diagram types.

- **Strucuture:**
  - Class Diagram
  - Component Diagram
  - Deployment Diagram
  - Object Diagram
  - Package Diagram
  - Composite Structure Diagram
  - Profile Diagram

- **Behavior:**
  - Use Case Diagram
  - Activity Diagram
  - State Machine Diagram
  - Sequence Diagram
  - Communication Diagram
  - Interaction Overview Diagram
  - Timing Diagram

2. UML
Diagrams

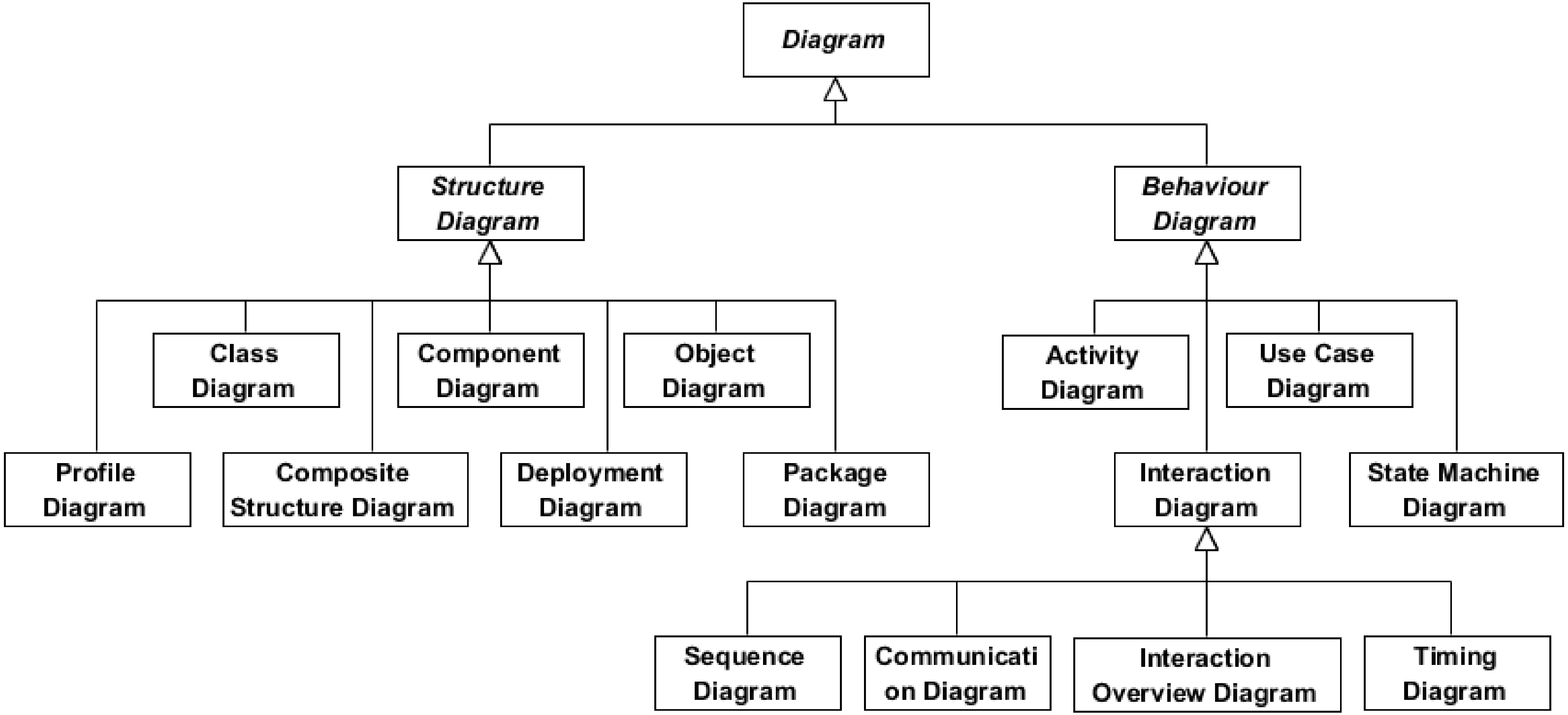


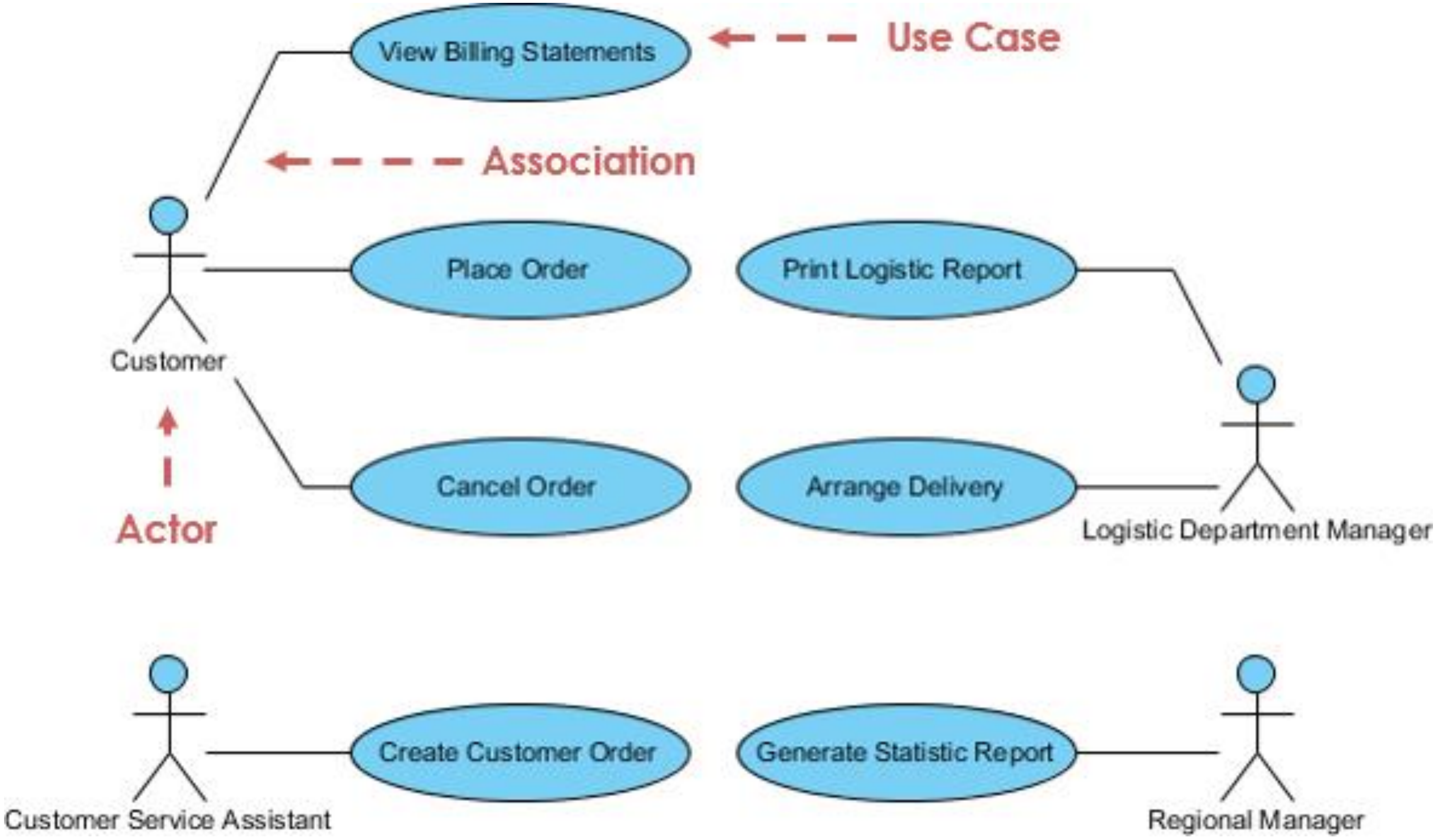
Figure: UML Diagrams

2. UML

Several Important Diagrams

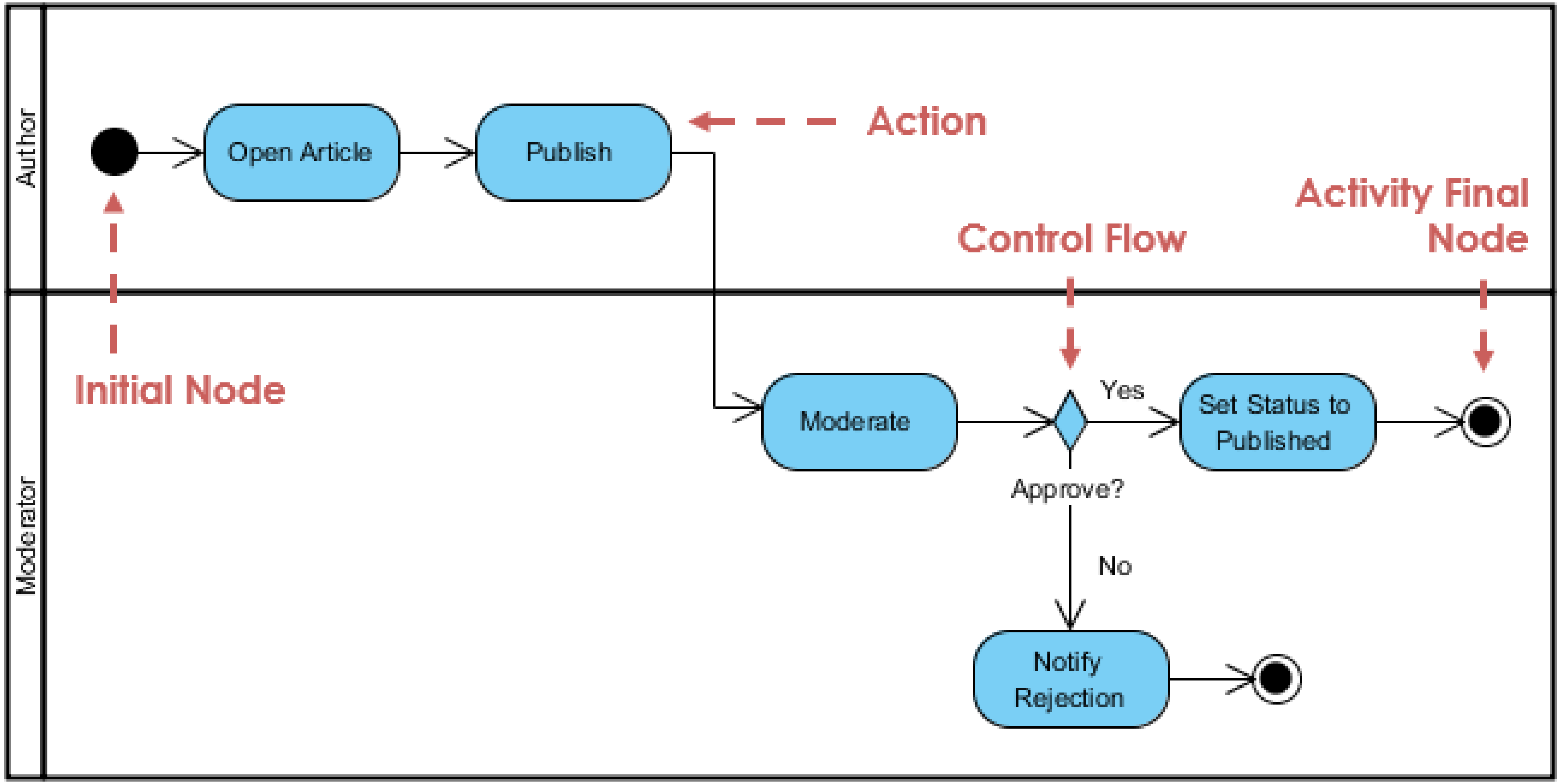
- Which diagrams should we use?
  - No single answer → UML made to satisfy different point of view.
- At least for a developer, please try to understand at least 3 diagrams:
  - Use Case Diagram
  - Activity Diagram
  - Class Diagram

2. UML
Use Case Diagram Example



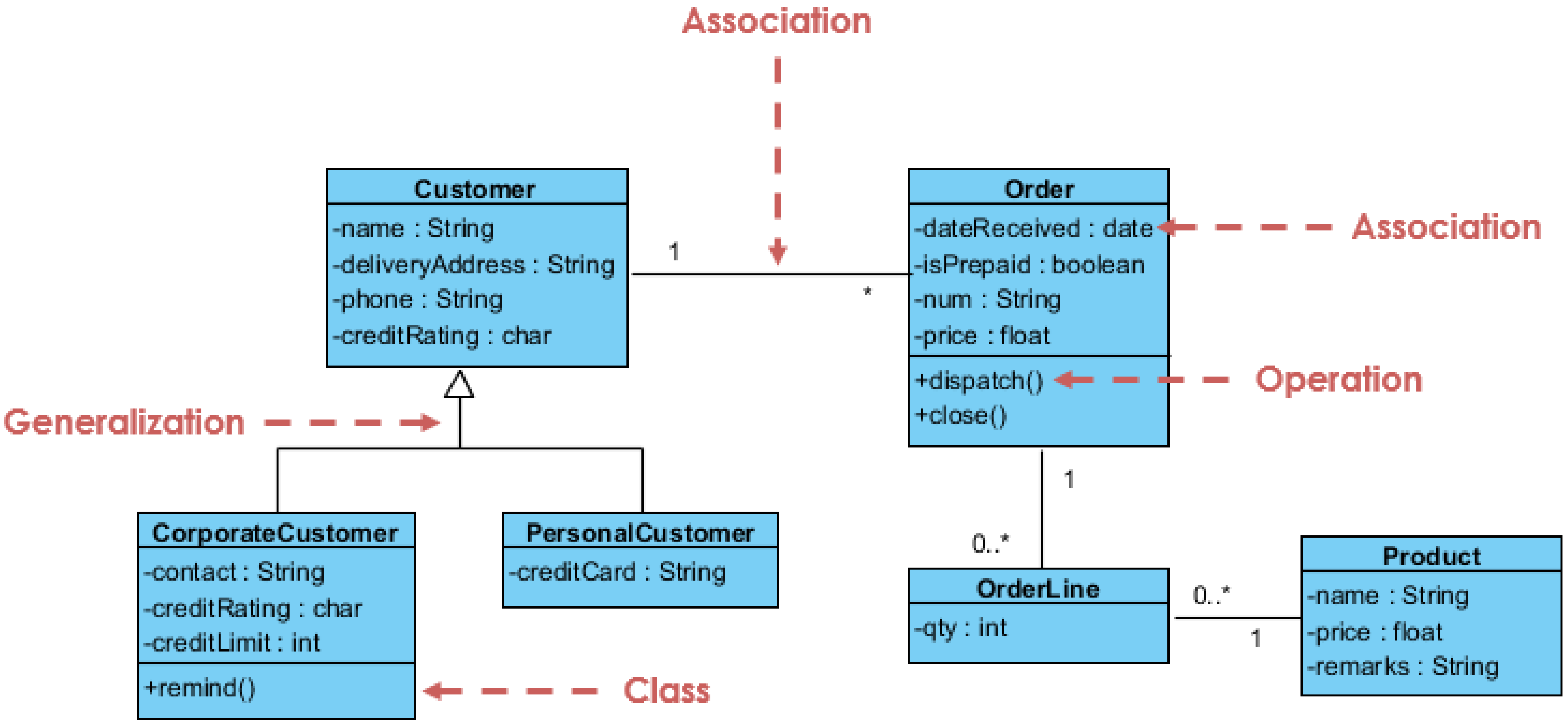
2. UML

Activity Diagram Example



2. UML

Class Diagram Example



Topic #3: Use Case Detailing

3. Use Case Detailing

- Use Case is the earliest diagram created after doing textual analysis.

- 1 Use Case → 1 Task: an activity that the actor/actor wants to do.

- Use case diagrams are very general so they need to be detailed.
  - Understand clearly what you want to achieve in the use case.
  - In order to facilitate the process of designing the User Interface
  - Avoid missing anything during the implementation process.

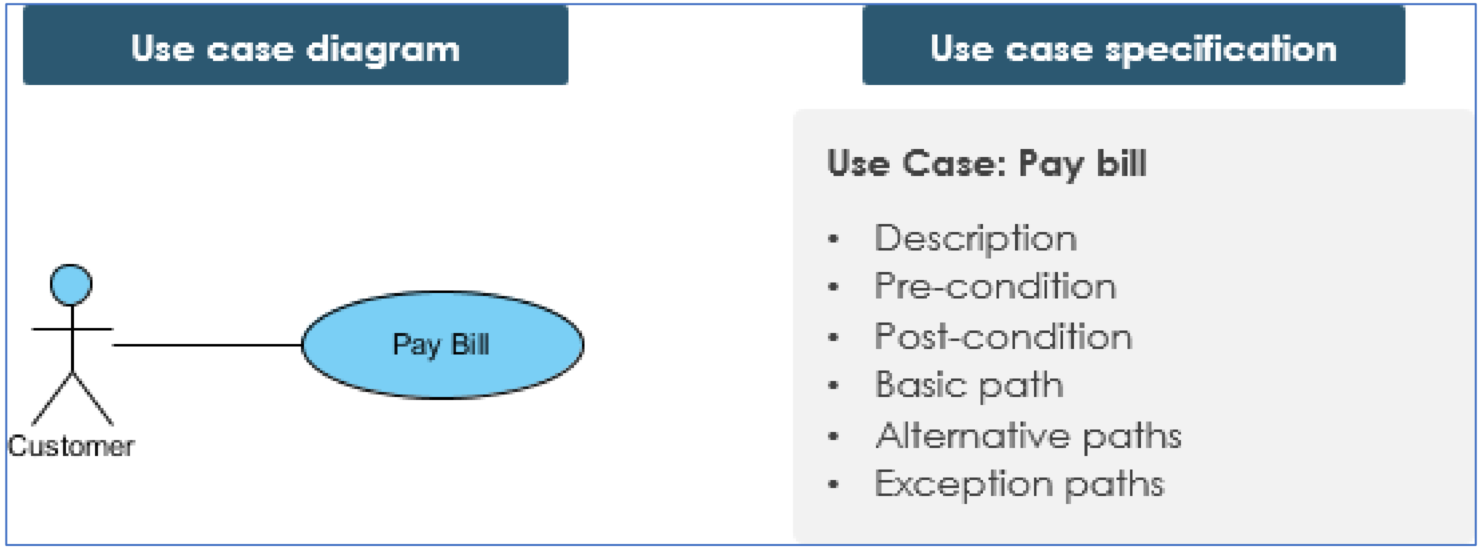
- Done by adding to each use case (1 ellipse):
  - 1 Use Case Specifications
  - 1 Activity Diagrams

3. Use Case Detailing

Use Case Specification

- Characteristics of a Use Case:
  - Has only 1 goal
  - Has only 1 endpoint
  - There can be multiple paths from start to finish:
    - Variations of things that can be done to achieve the goal
    - Where each condition requires specific action

- That's why we need to make detailed explanation from each of the use case → Use Case Specification.



Use case specification

Use Case: Pay bill

- Description
- Pre-condition
- Post-condition
- Basic path
- Alternative paths
- Exception paths

Use Case Specification

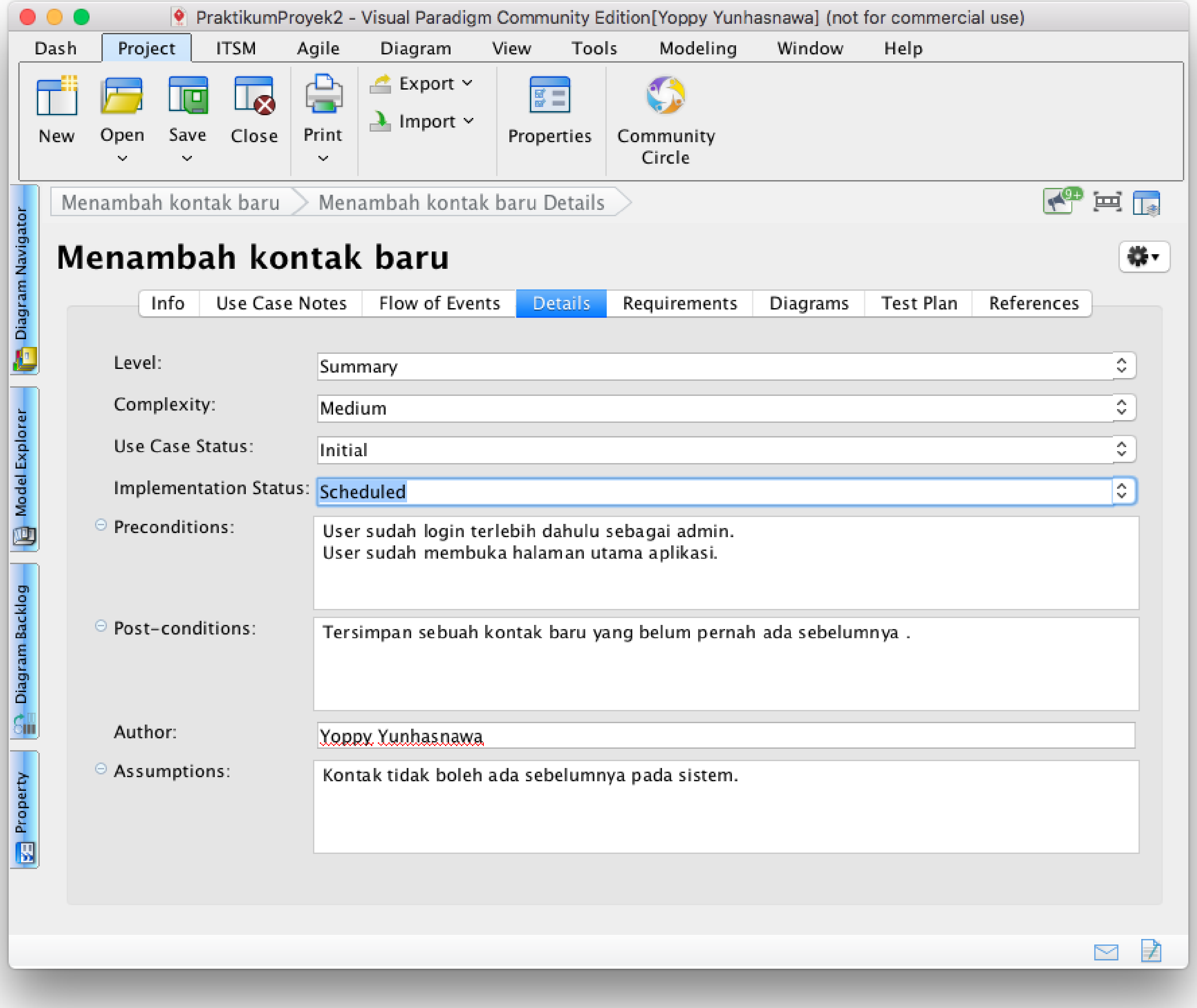
- Is a formal document containing text explaining a use case in detail.
  - Again, 1 ellipse = 1 use case → 1 use case specification

- In the form of formatted-text, NOT a diagram.

- Minimally consists of:
  - The title of the Use Case
  - Actor
  - Brief description.
  - Level
  - Priority
  - Implementation status (Agile)
  - Pre-conditions
  - Post-Conditions.

3. Use Case Detailing

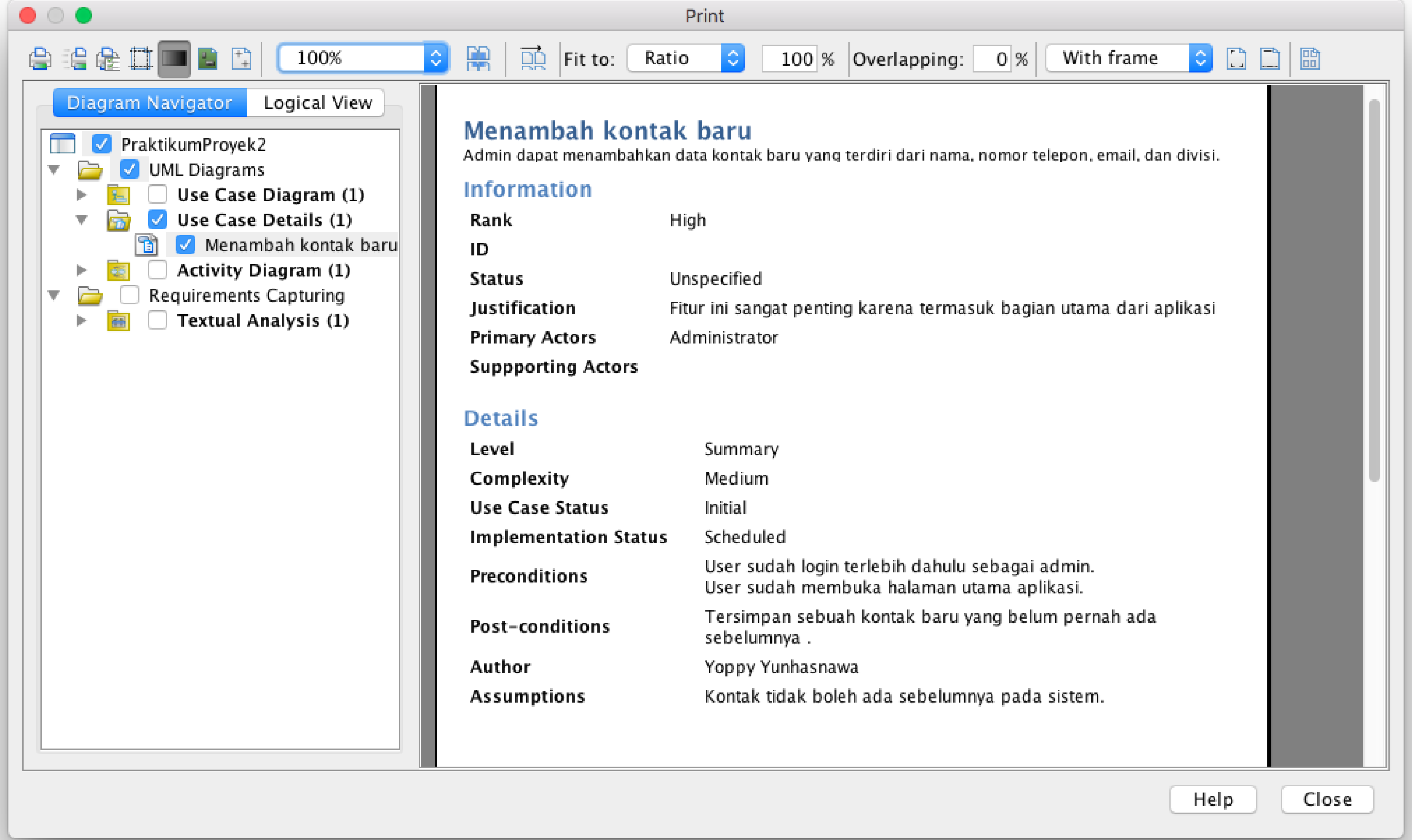
Use Case Specification in Visual Paradigm



Menambah kontak baru

Level:	Summary
Complexity:	Medium
Use Case Status:	Initial
Implementation Status:	Scheduled
Preconditions:	User sudah login terlebih dahulu sebagai admin. User sudah membuka halaman utama aplikasi.
Post-conditions:	Tersimpan sebuah kontak baru yang belum pernah ada sebelumnya .
Author:	Yoppy Yunhasnawa
Assumptions:	Kontak tidak boleh ada sebelumnya pada sistem.

3. Use Case Detailing
Use Case Specification in Visual Paradigm



Menambah kontak baru

Admin dapat menambahkan data kontak baru yang terdiri dari nama, nomor telepon, email, dan divisi.

Information

Rank	High
ID	
Status	Unspecified
Justification	Fitur ini sangat penting karena termasuk bagian utama dari aplikasi
Primary Actors	Administrator
Suppporting Actors	

Details

Level	Summary
Complexity	Medium
Use Case Status	Initial
Implementation Status	Scheduled
Preconditions	User sudah login terlebih dahulu sebagai admin. User sudah membuka halaman utama aplikasi.
Post-conditions	Tersimpan sebuah kontak baru yang belum pernah ada sebelumnya .
Author	Yoppy Yunhasnawa
Assumptions	Kontak tidak boleh ada sebelumnya pada sistem.

3. Use Case Detailing

Activity Diagram

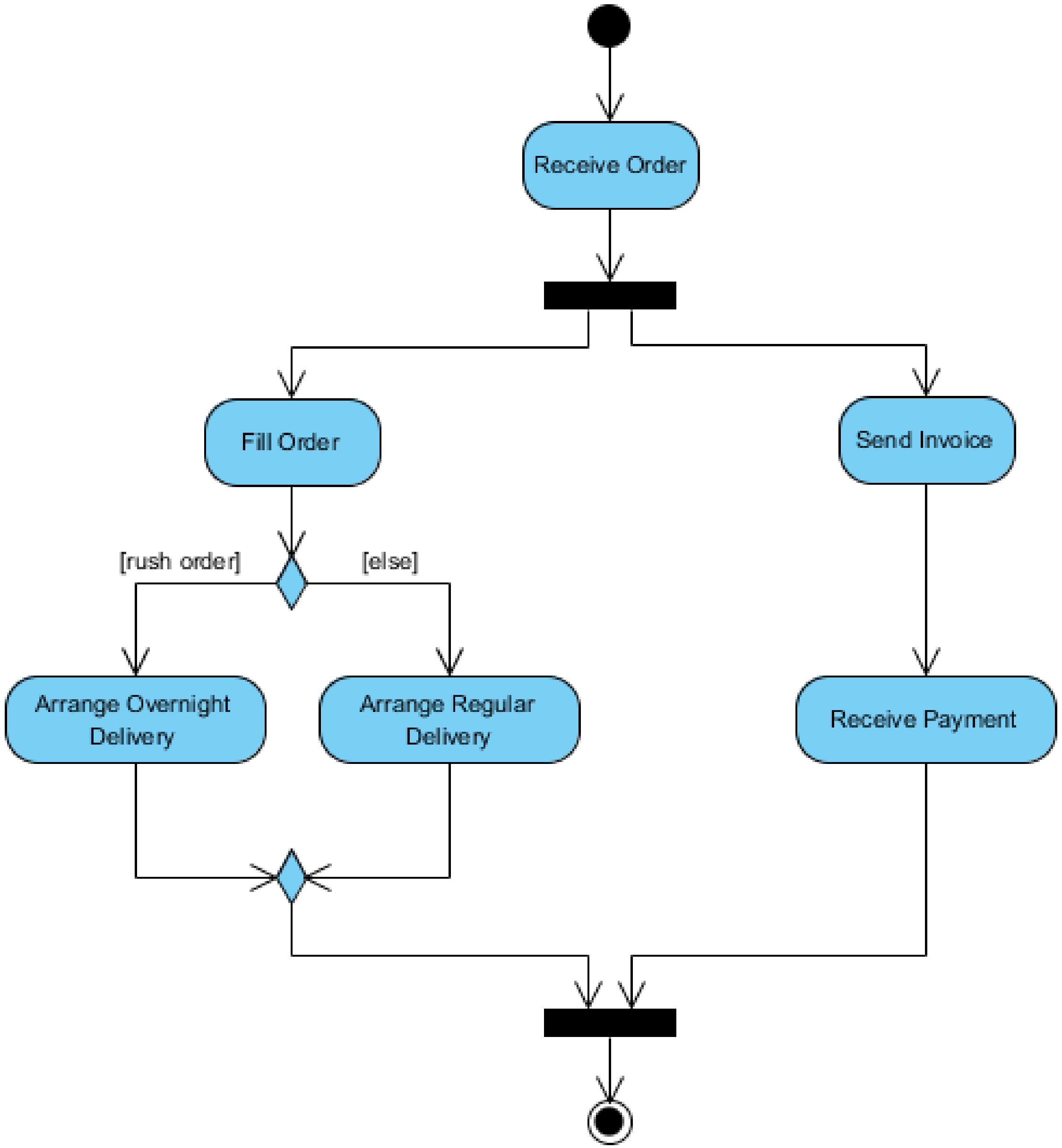
- Is a kind of diagram explaining the steps need to be taken by certain actor in order to achieve certain goal of a use case.

- An activity diagram can show:
  - Concurrency
  - Branching
  - Control Flow
  - Object Flow

- 1 Ellipse = 1 Use Case → 1 Use Case Specification + 1 Activity Diagram.

- What is the difference between **activity diagrams** vs **flowcharts**?
  - Activity diagrams represent the flow of activities within a system and depict the workflow/interaction between these various system activities.
  - Flowcharts, on the other hand, represent the sequence of steps in an event, process, or system.

3. Use Case Detailing
Activity Diagram Example



Questions?

Thank You

Task

- Create one Use Case Specification and one Activity diagram for one ellipse of the previous meetings' Use Case Diagram!

- Classroom Code: 6axztdt

References

[1] https://www.tutorialspoint.com/software_engineering/software_design_basics.htm

[2] https://www.reqview.com/papers/ReqView-Example_Software_Requirements_Specification_SRS_Document.pdf

[3] https://jameskle.com/writes/divide-and-conquer

[4] https://creately.com/guides/activity-diagram-tutorial/#:~:text=What%20is%20the%20difference%20between,event%2C%20process%2C%20or%20system.