Software Analysis and Design: Modeling

Hogeschool Rotterdam
2021

Informatica::Analysis::INFSAD01-A

Part 0: Course

Course Information

- Topic: Software Analysis and Design
- Credits: 4 credits
- Assessment:
    - **Written exam**: Multiple Choice Exam.
    - **Assignment**:
        - Assignment: Summative, Will be evaluated, Result as **Pass** / **Fail**.
    - Condition for credit: **(written exam >= 5.5)** <u>**AND**</u> **(Pass for the assignment)**

Note: Depending on the submission, the teacher may arrange an oral check with the student to evaluate the assignment.

Objectives of the course.

- Review the learning of the course.
  - What do you expect at the end?
- PLEASE go in the course manual and check the learning objectives of this course!
  - IT IS IMPORTANT!

How to succeed?

- Self-study:
  - Read provided reading materials.
- Lessons:
  - Try to attend all the classes: **Active Participation**.
- Exercises and Assignments:
  - Be critical about your solutions.
- Slides:
  - Review the slides and videos regularly after each lesson.

Warning ...

The course gradually **introduce *enormous* amount of symbols** along **with their semantics.**

- **Read, study, practice and discuss from right now!**
  - **And after today... EVERY WEEK!**

Part 1: Methodologies

Questions

Part 1:

- What are the common challenges in software projects?
- What are main responsibilities of a software engineer?
- Why do we need SDLCs?
- Name and discuss four common SDLC: context, pros- , cons- ?

Challenges

- Software is complex to construct.
- Impossible for the individual developer to comprehend all the subtleties of its design.
- Future users and customers have a hard time to accurately specify system requirements.
- Difficult to manage *‘evolving’* requirements

Motivation

Why Software analysis and design?

- Quality
- Costs
- Lessen complexity
- Predictability
- Risk reduction
- Failure mitigation
- Work in parallel teams

Methodology

To manage complexity we need a methodology:

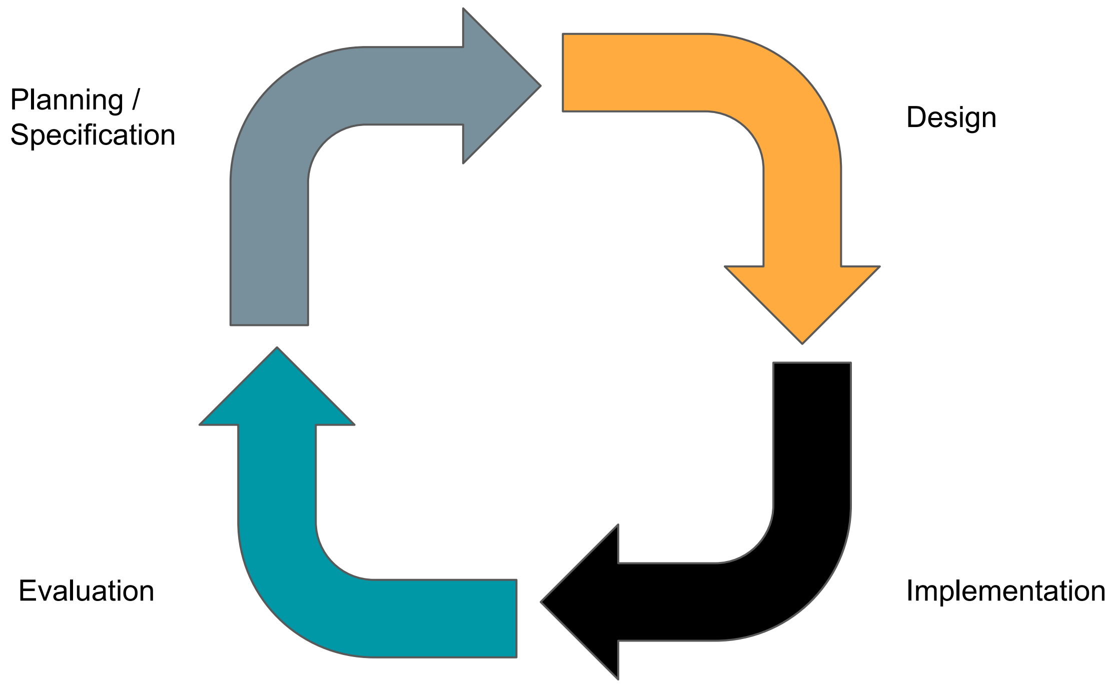
- Plan, Control, Estimate, …
- Simply setting up some norms to develop the system

Software Development Methodology

Software **d**evelopment **l**ife **c**ycle

- Referred to as the **SDLC**
- **It defines:**
  - The general steps that are taken to build software
  - The responsibilities for team members during each step or phase
    - Some steps may overlap, but generally define the phases off the project
    - If some steps are not successful the project may fall back on an earlier step.

Software Development Life Cycle (SDLC)

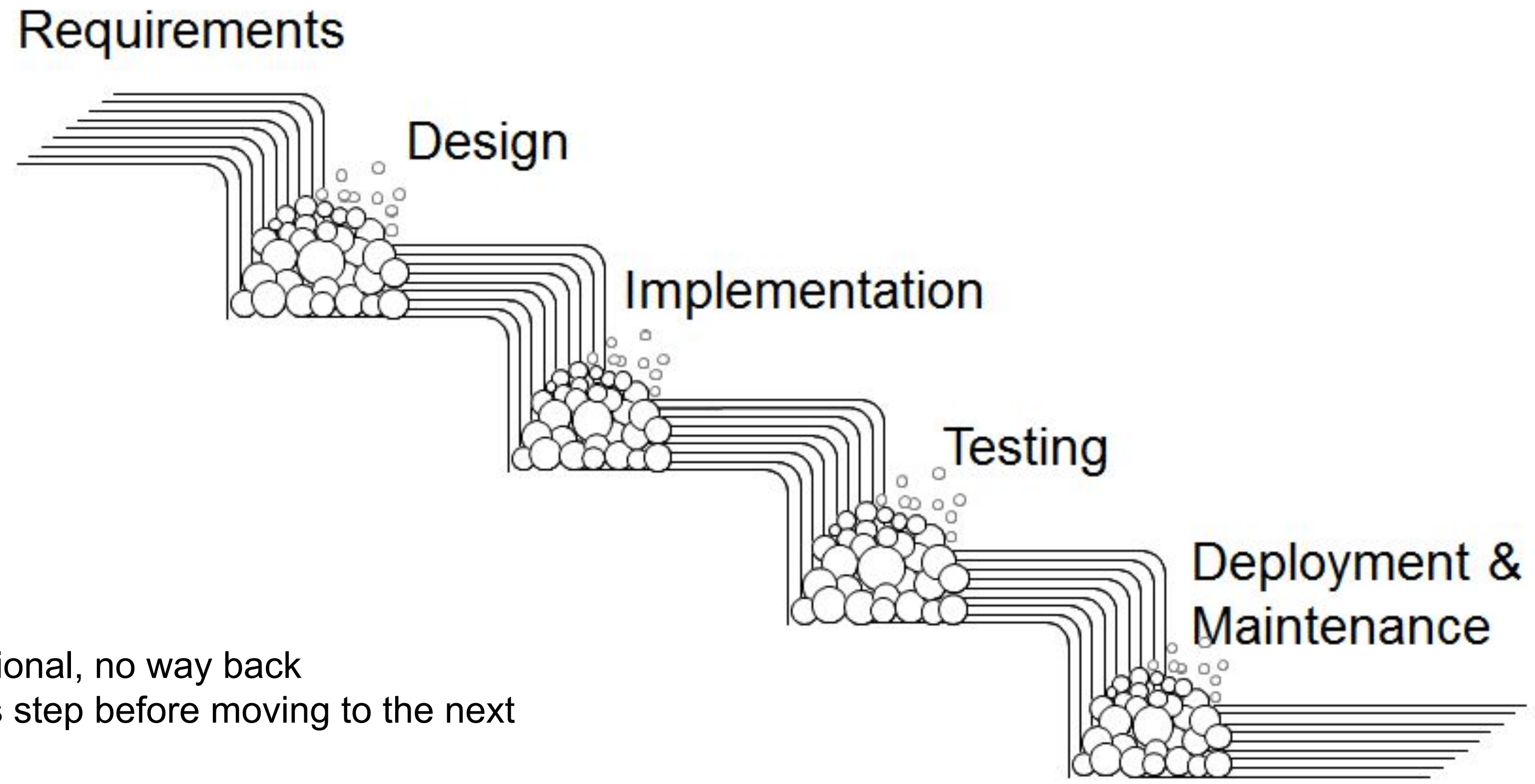


Software Development Methodology

- **Waterfall**
  - Unidirectional, finish this step before moving to the next
- **Iterative and Incremental**
  - Develop increment of functionality, repeat in a feedback loop
- **Agile**
  - User feedback essential; feedback loops on several levels of granularity

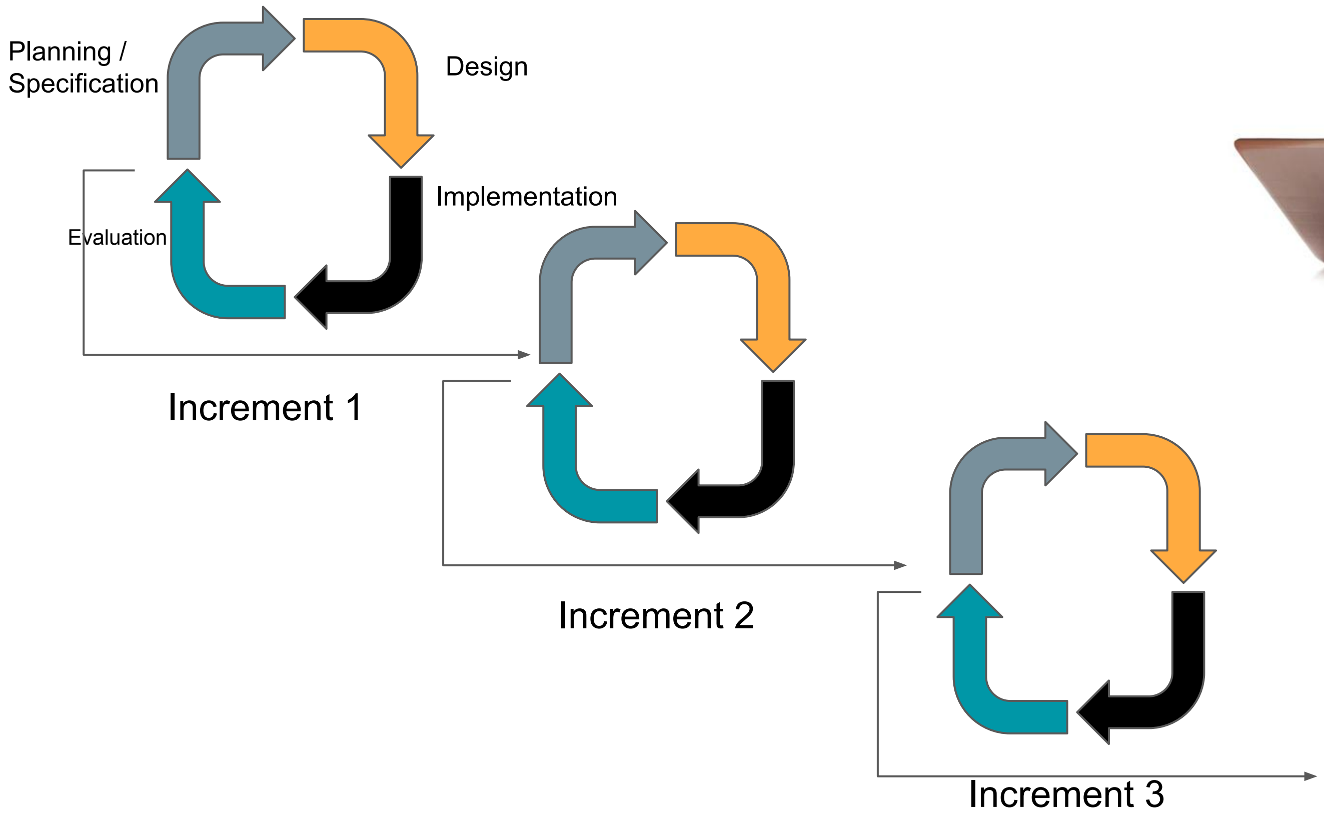
Read here: https://en.wikipedia.org/wiki/Software_development_process

Waterfall method



Unidirectional, no way back
finish this step before moving to the next

Incremental development



End user involved in evaluation phase

Agile development

Continuous customer involvement

Questions

Part 1:

- What are the common challenges in software projects?
- What are main responsibilities of a software engineer?
- Why do we need SDLCs?
- Name and discuss four common SLDC: context, pros- , cons- ?

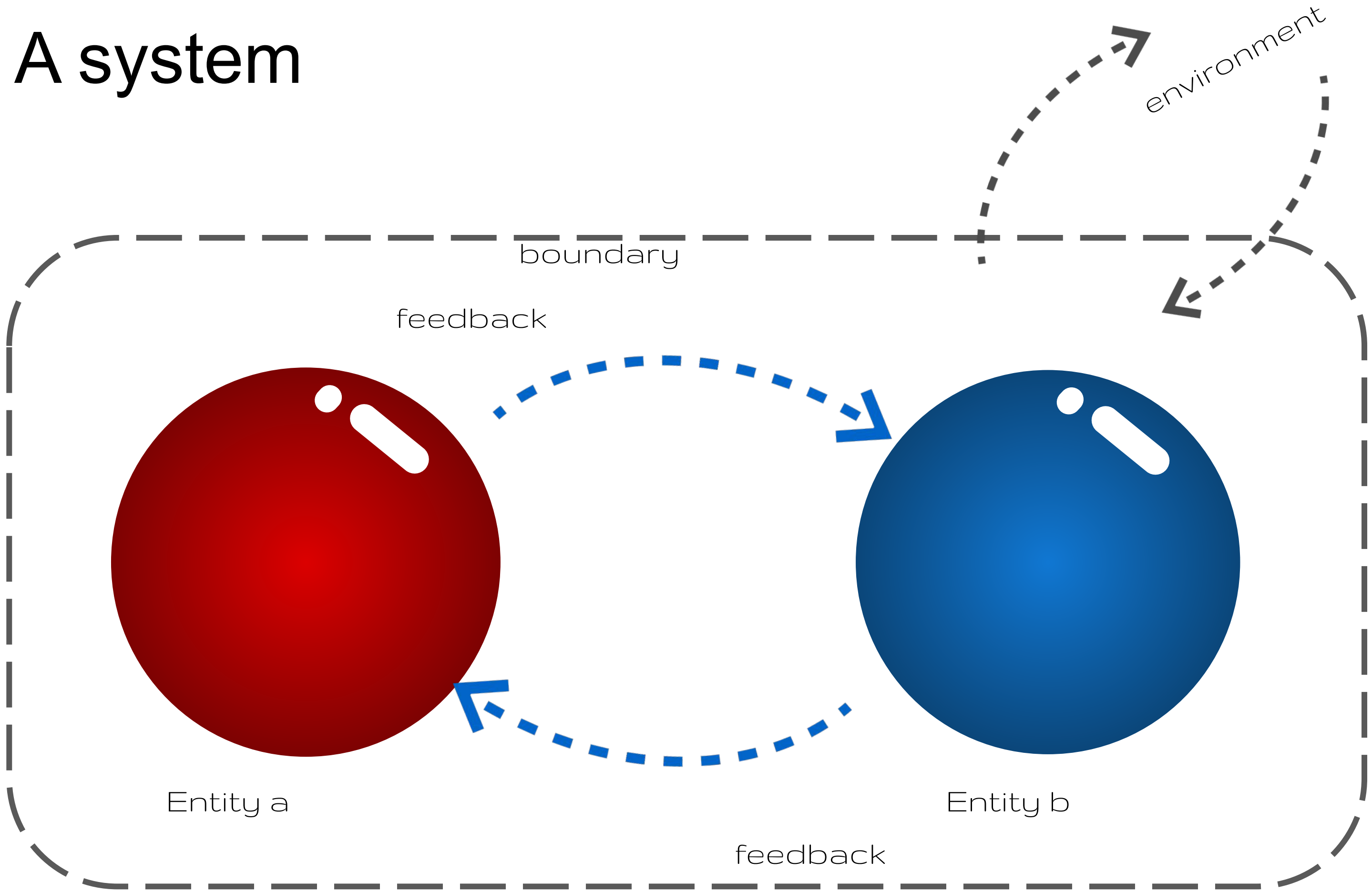
Part 2: Requirements and Modeling

Questions

Part 2:

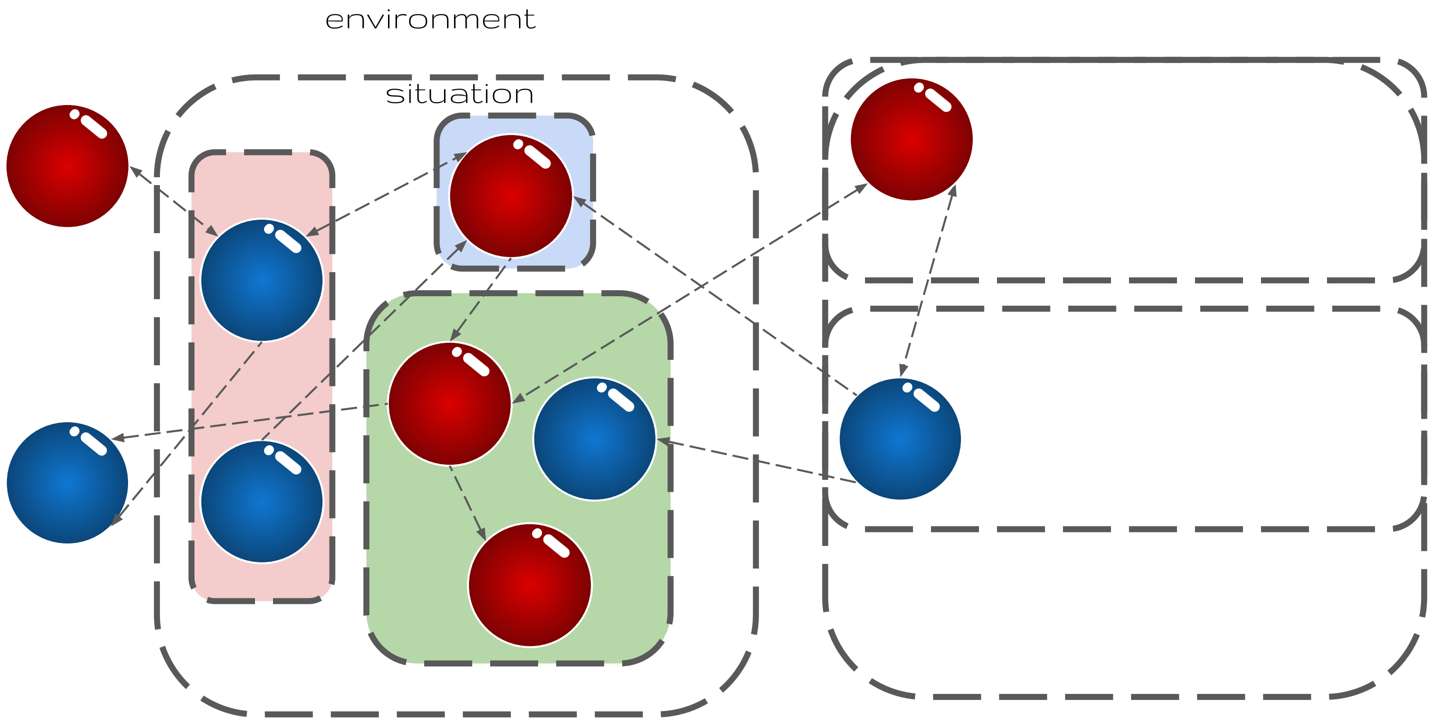
- What is a system boundary?
- What is the difference between functional and non-functional requirements?
- What are the categories of non-functional requirements?
- What are the objectives of modeling?

A system

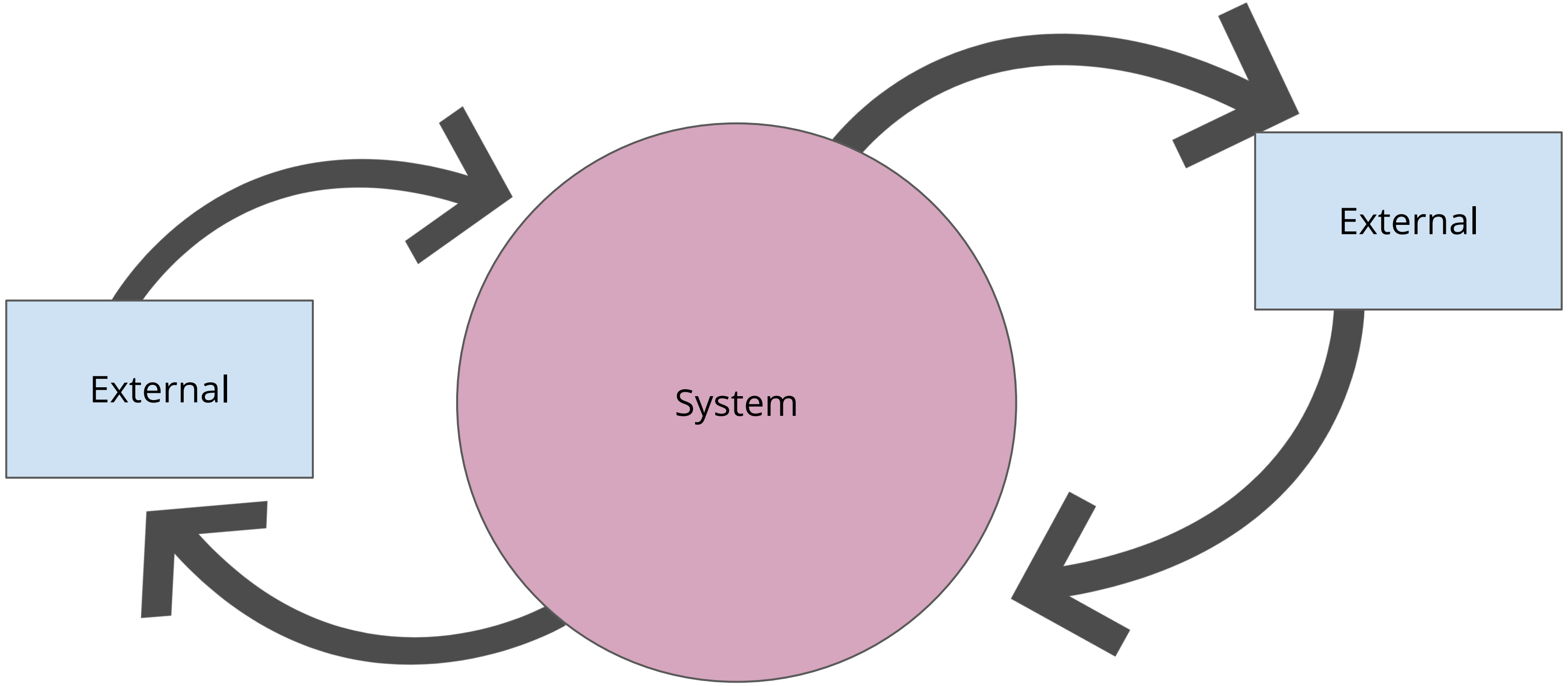


What is a system?

Interacting systems and subsystems



Boundaries



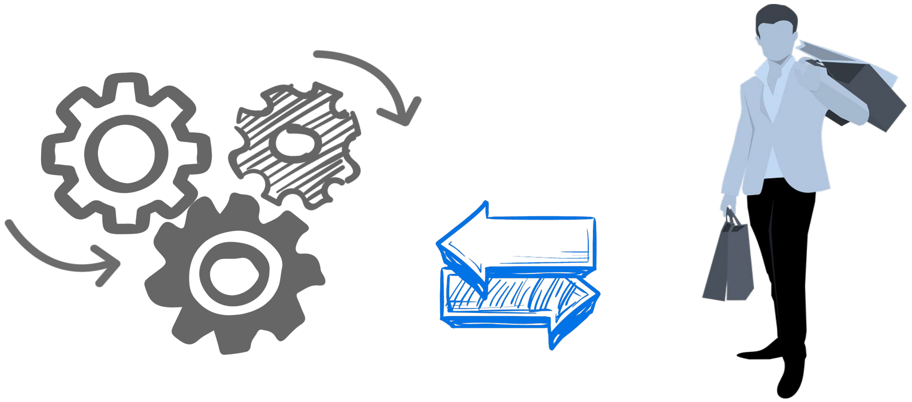
Software analysis and design

Goal: we need to detect the borders and find externals.

Externals are the entities that interact with our system.

The end of this section: externals have expectations, then requirements will appear

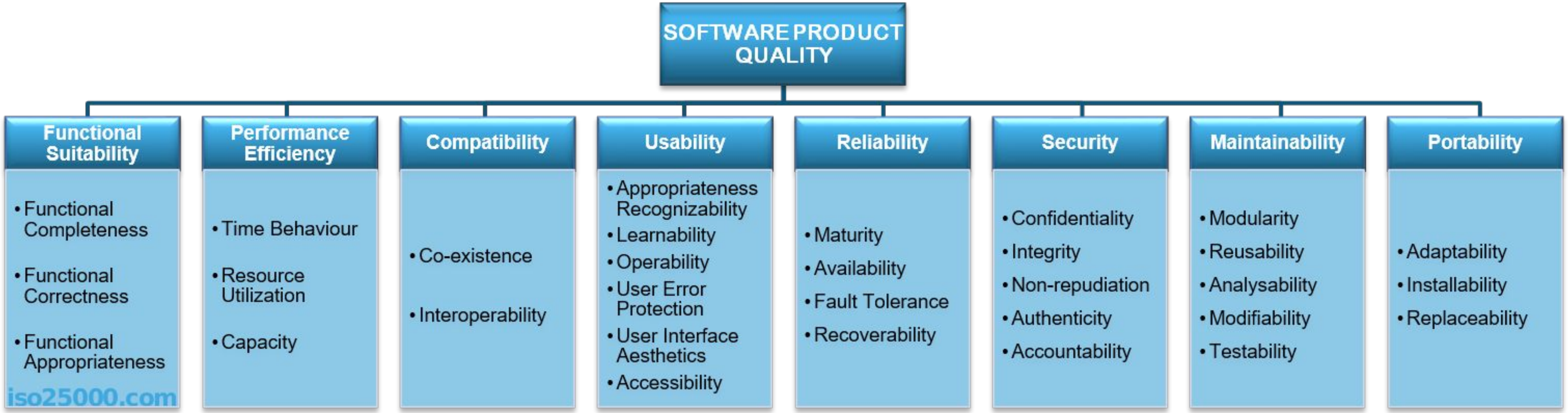
"Requirements describe what a system should be able to do"



Functional, **non-functional** requirements and constraints

- Functional
  - What should the system do
    - **As a** user I **need to** authenticate myself **to be able to** interact with the system
- Non-functional
  - How (in what quality) would you like the system to do it
    - *A page must be loaded **within 3 seconds***
  - Constraints
    - ***The boundaries** within the system operates*
    - *We can only provide **3 fte** end users*
    - *We use **only sql** server databases*

ISO 25010

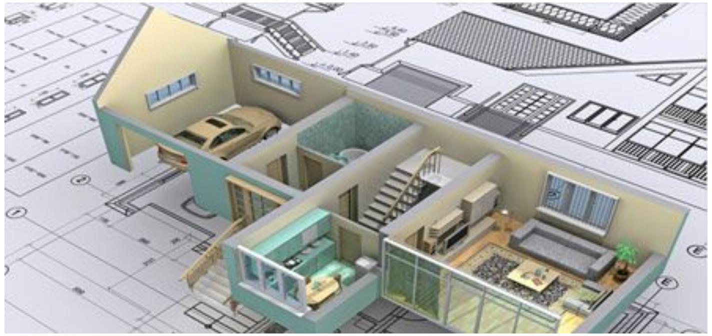


Ref: https://iso25000.com/index.php/en/iso-25000-standards/iso-25010

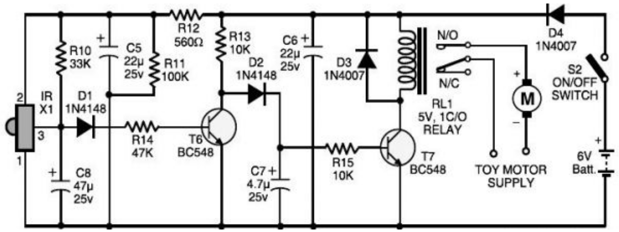
Modeling

What is a model?

- **A simplification of reality.**



“The model captures the *<u>important aspects</u>* of the thing being modeled from a *<u>certain point of view</u>* and *<u>simplifies or omits the rest</u>*.”



- To **<u>manage</u>** understandability of the system we are developing.

The goal of modeling

- **Visualization**: Models help us to visualize a system as it is or as we want it to be.
- **Specification**: Models permit us to specify the structure or behavior of a system.
- **Guideline**: Models give us a template that guides us in constructing a system.
- **Documentation**: Models document the decisions we have made.

Levels of Models

Models take on *different forms* for *various purposes* and appear at *different levels of abstraction*.

- **High-level models**: in early stages of the project serve to focus on the requirements and explore possible options.
- **Analysis level**: in analysis or preliminary design stages focus on key concepts and mechanisms of the eventual system. No detail yet!
- **Implementation model**: includes semantics, algorithms, data-structures, mechanisms required to build the system

Evolution and Iteration

It is *impossible* to understand a large system in a *single*, *linear* pass:

- Models *evolve over time*: starting from high-level models, over time, much more detail is added.
- Models are *iterated at all levels*: as developers work with a system and understand it better, the model must be iterated at all levels to capture that understanding.

Unified Modeling Language

UML is a general-purpose visual modeling language to:

- *Specify*
- *Visualize*
- *Construct*
- *Document*

the artefacts of a software system.

Unified Modeling Language

UML *is a standard modeling language* for Object-Oriented systems.

UML is:

- **NOT** A programming language.
- **NOT** A programming tool.
- **NOT** A development process.

What does “*unified*” mean?

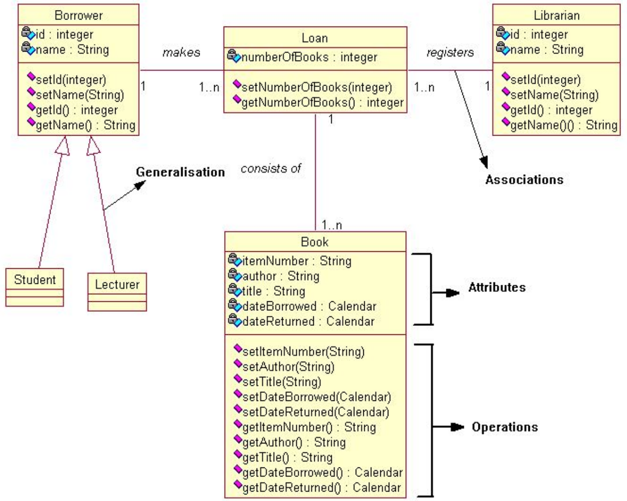
In UML, unified can have the following relevant meanings:

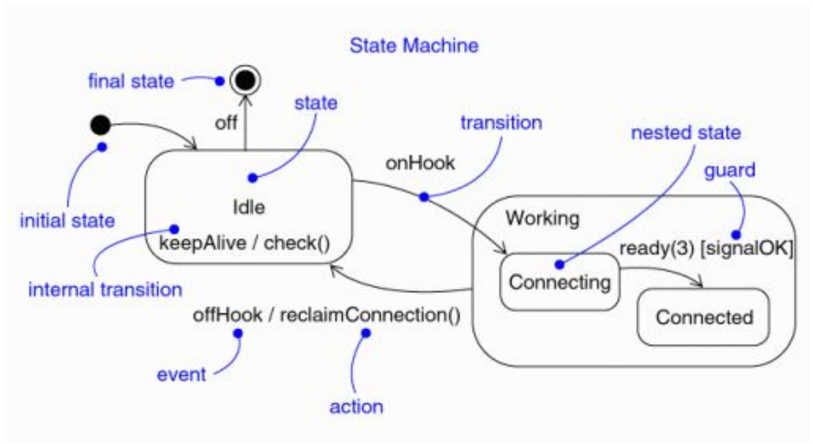
- Across historical methods and notations.
- Across the development lifecycle.
- Across the application domain.
- Across implementation languages and platforms.
- Across the development process.

Logical View

Describes the abstract descriptions of a system’s parts:

- What a system is made up of and how the parts interact with each other.
- UML diagrams: **class**, object, **state machine**, and interaction diagrams.

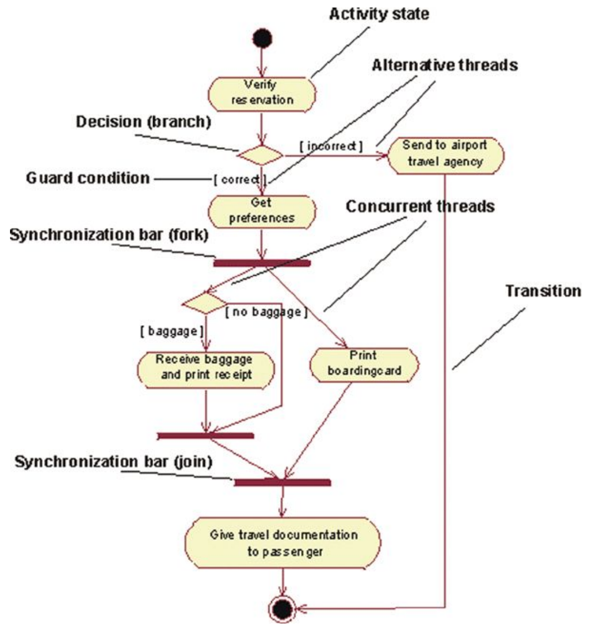




Process View

Describes the processes within your system.

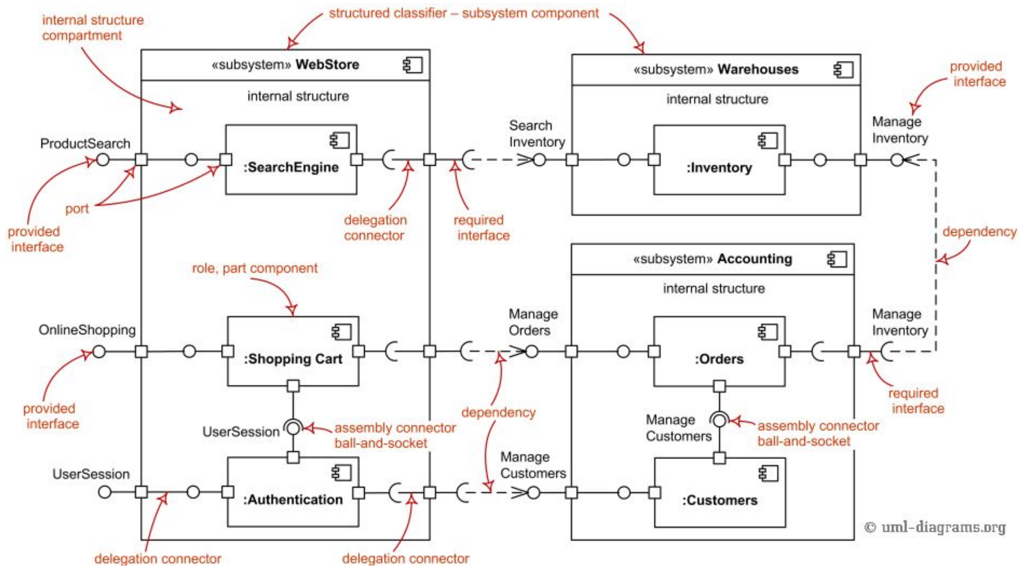
- Visualizing what must happen within your system.
- UML diagrams: **activity diagrams**.



Development View

Describes how your system's parts are organized into modules and components.

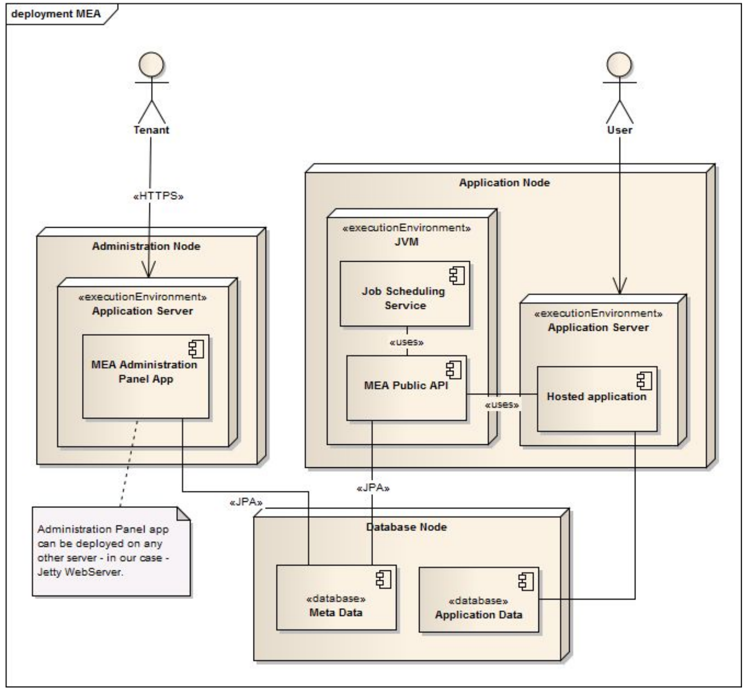
- Manage layers within your system's architecture.
- UML diagrams: package and **component diagrams**.



Physical View

Describes how the system's design is brought to life as a set of real-world entities.

- How the abstract parts map into the final deployed system.
- UML diagram: **deployment** diagrams.


deployment MEA
Tenant
User
«HTTPS»
Administration Node
«executionEnvironment» Application Server
MEA Administration Panel App
Application Node
«executionEnvironment» JVM
Job Scheduling Service
«uses»
MEA Public API
«uses»
«executionEnvironment» Application Server
Hosted application
«JPA»
«JPA»
Database Node
«database» Meta Data
«database» Application Data
Administration Panel app can be deployed on any other server - in our case - Jetty WebServer.

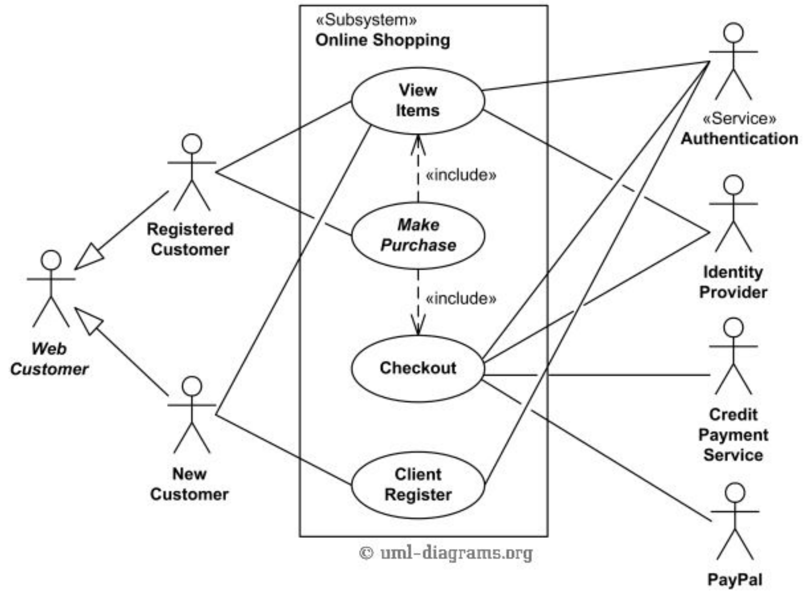
Use Case View

Describes the functionality of the system being modeled from the perspective of the outside world.

- Model what the system is supposed to do.
- UML diagram: **use case diagrams**, descriptions, and overview diagrams.

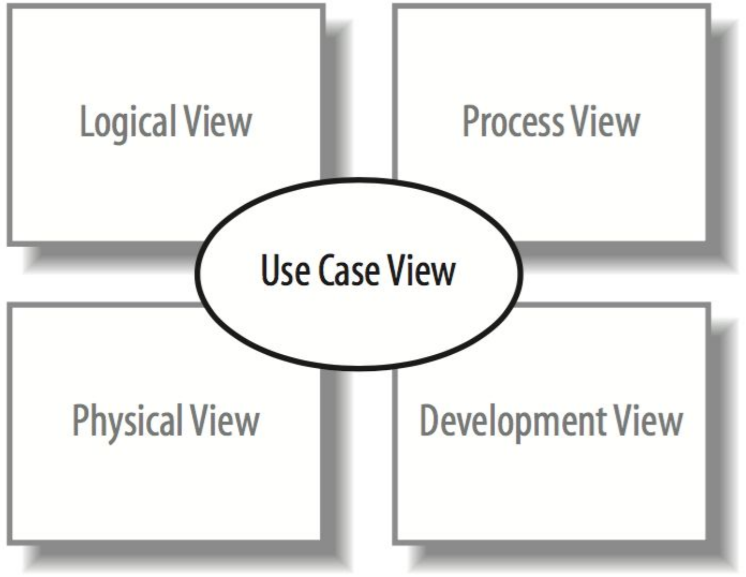
Note: All of the other views rely on the use case view to guide them

- that’s why the model is called 4+1.



Various views

Designing a software-intensive system, Kruchten’s 4+1 views addresses the concerns of the various stakeholders, i.e. end-users, developers, system engineers, etc.



Questions

Part 2:

- What is a system boundary?
- What is the difference between functional and non-functional requirements?
- What are the categories of non-functional requirements?
- What are the objectives of modeling?

Part 3: Use Case Modeling

Review of Basics

Review!

Use Case Diagram

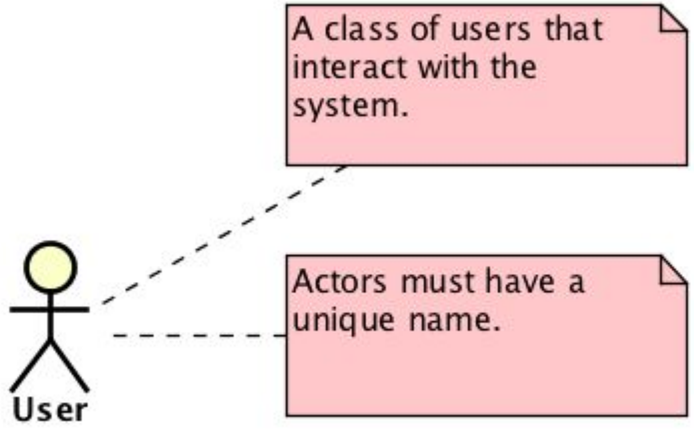
Use Case Diagram (UCD): to capture requirements of a system.

Major syntactical elements of a UCD:

- Actor: identifies entities that the system you are describing interacts with;
- Use Case: the use cases, or services, that the system knows how to perform;
- Relations: the lines that represent relationships between these elements.

Actor

Actor: is an identity outside the scope of the (sub)system under consideration, but that has significant interactions with it.


A class of users that interact with the system.
Actors must have a unique name.
User

Review!

Actor: How to identify?

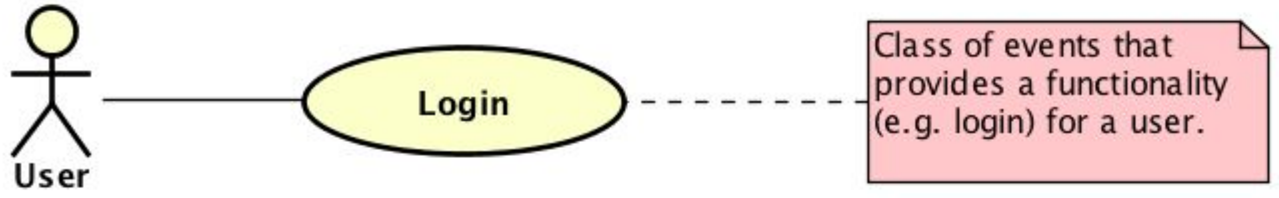
How to discover actors?

- Identify entities that are using or interacting with the system.
- Think about roles rather than people or job titles.

Actors can be: a human, a device, an executable process, a system.

Use Case

Use Case: A ***behaviorally related sequence of transactions*** formed by an ***actor*** in a ***dialogue with the system*** to provide some measurable value to the actor [Jacobson et al].


User
Login
Class of events that provides a functionality (e.g. login) for a user.

Review!

Use Case

A use case is a case (or situation) where your system is used to fulfill one or more of your actor's requirements.

- a use case captures a piece of ***functionality*** that the system provides.
- a use case ***does not*** specify ***non-functional*** requirements.

Review!

Use Case

A use case is a specific way of using the system by using some part of the functionality

Each use case is a complete course of events (a class of scenarios) in the system from a user's perspective.

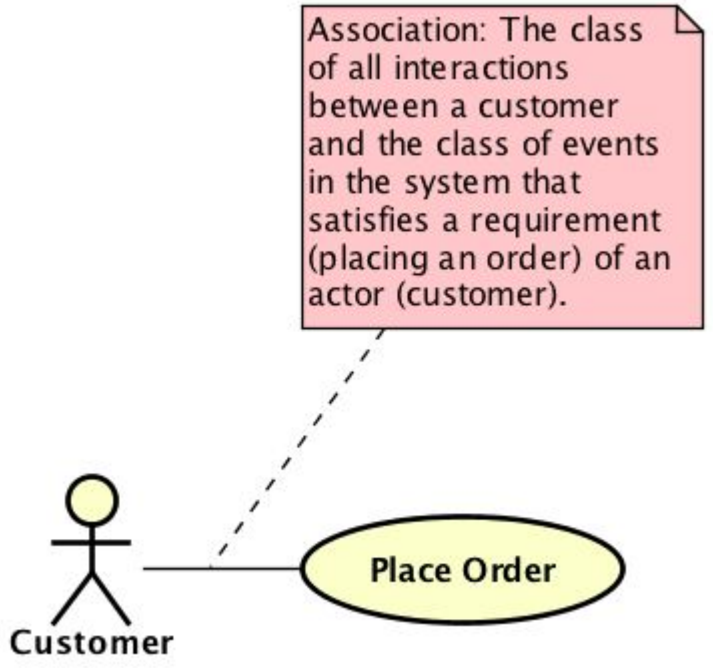
Review!

Relationships: Actor-Use Case Association

Association: The communication path between an actor and a use case that it participates in.

- The ***class of interactions*** between an actor and the system.

Relationships: Actor-Use Case Association


Association: The class of all interactions between a customer and the class of events in the system that satisfies a requirement (placing an order) of an actor (customer).
Customer
Place Order

UCD Relationships: In depth

Questions

Part 3:

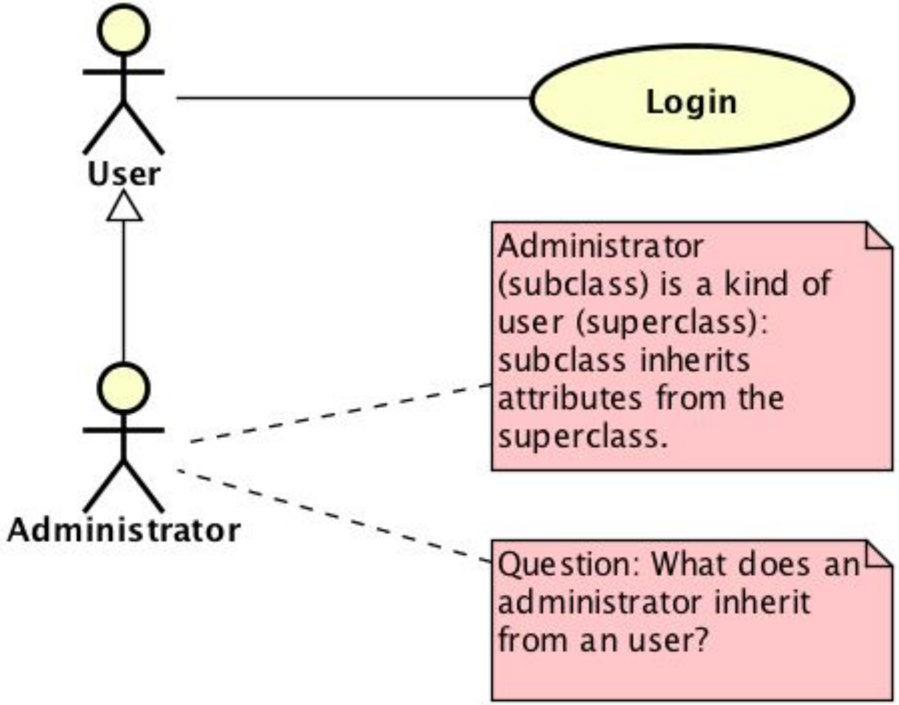
- What is the difference between *extend* and *include*?
- What is the semantics of generalization?
  - Pay attention on how the arrows point and how the information can be inherited or not.

Relationships: Actor Generalization

Generalization (inheritance):

the only relationship between two actors.

Relationships: Actor Generalization


User
Login
Administrator
Administrator (subclass) is a kind of user (superclass): subclass inherits attributes from the superclass.
Question: What does an administrator inherit from an user?

Relationships: Use Cases

Relationships between use cases: breaks your system's behavior into manageable chunks.

When filling out use case descriptions you may notice that:

- there is some similarity between steps in different use cases, or
- some use cases work in multiple optional flows throughout its execution.

Relationships: Use Cases

How to get rid of the repetition between use case descriptions?

- Using relationships between use cases one can show **reusable**, **optional**, and **specialized** use case behavior between use cases.

Relationships: Use Cases

Two Use Cases can relate via:

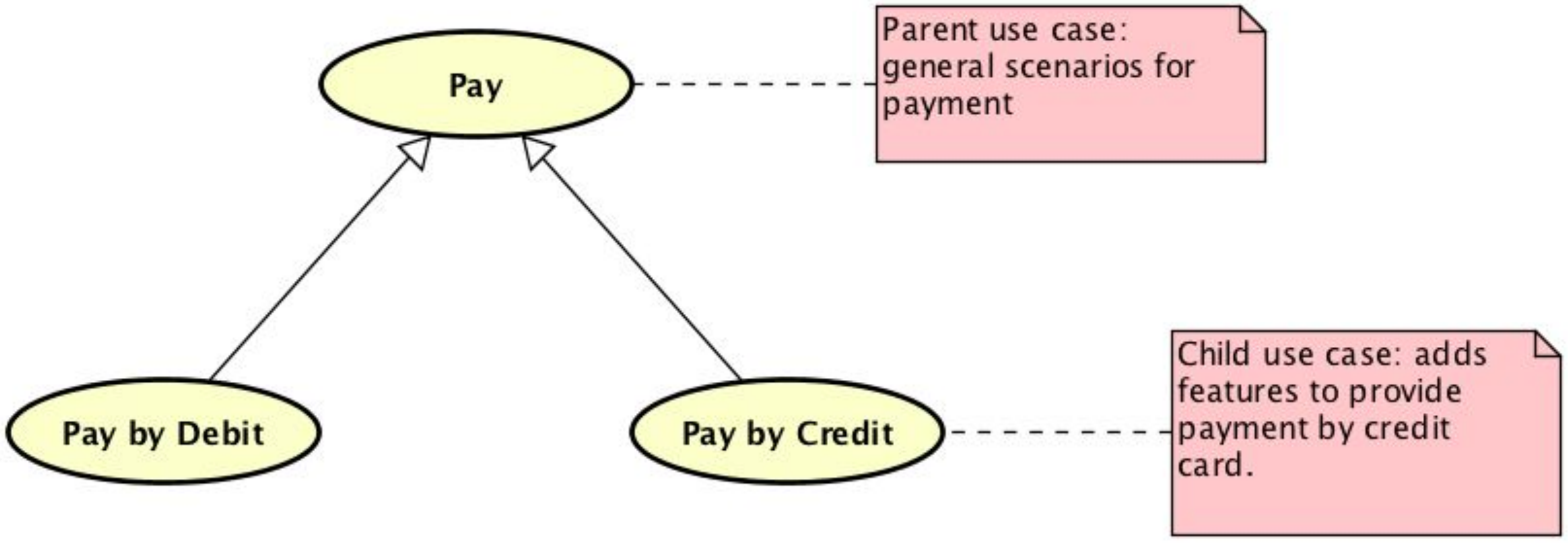
- Generalization
- Dependency
  - stereotyped as “include”
  - stereotyped as “extend”

Relationships: Use Case Generalization

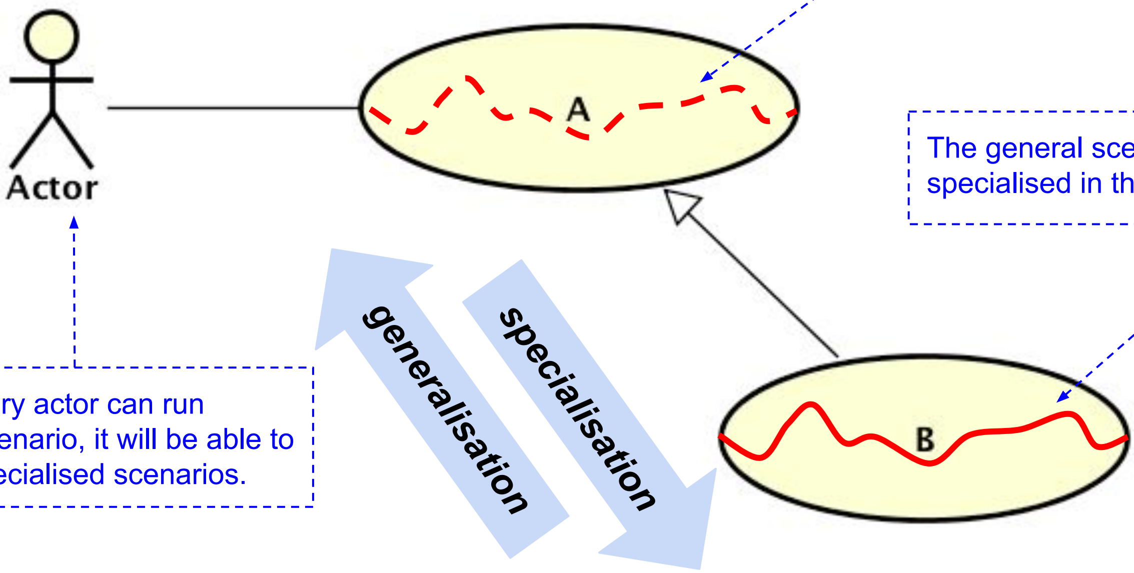
Generalization:

A relationship between a general use case and a more specific use case that inherits and adds features to it.

Relationships: Use Case Generalization


Pay
Parent use case: general scenarios for payment
Pay by Debit
Pay by Credit
Child use case: adds features to provide payment by credit card.

Use Case - generalisation



The general scenario (e.g. Buy Item) is abstracted in the parent use case.

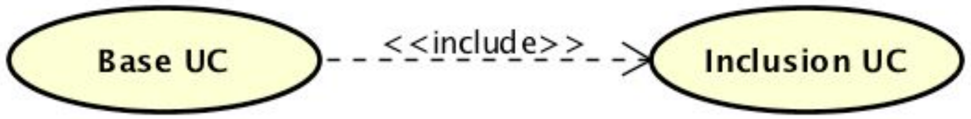
The general scenario (abstract workflow) is specialised in the child scenario (e.g. Buy Ticket).

If the primary actor can run abstract scenario, it will be able to run any specialised scenarios.

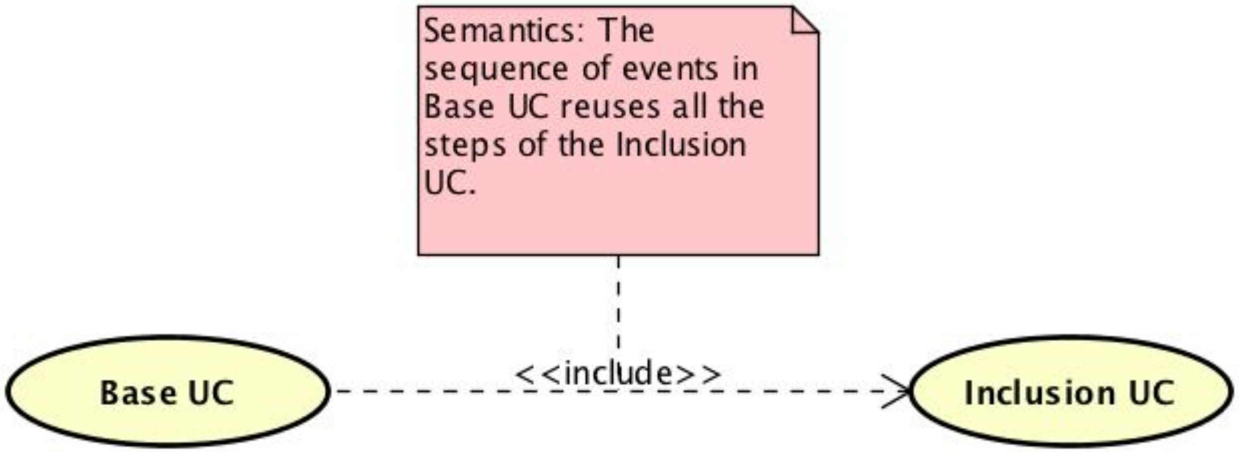
Relationships: include

A relationship from a ***base*** use case to an ***inclusion*** use case.

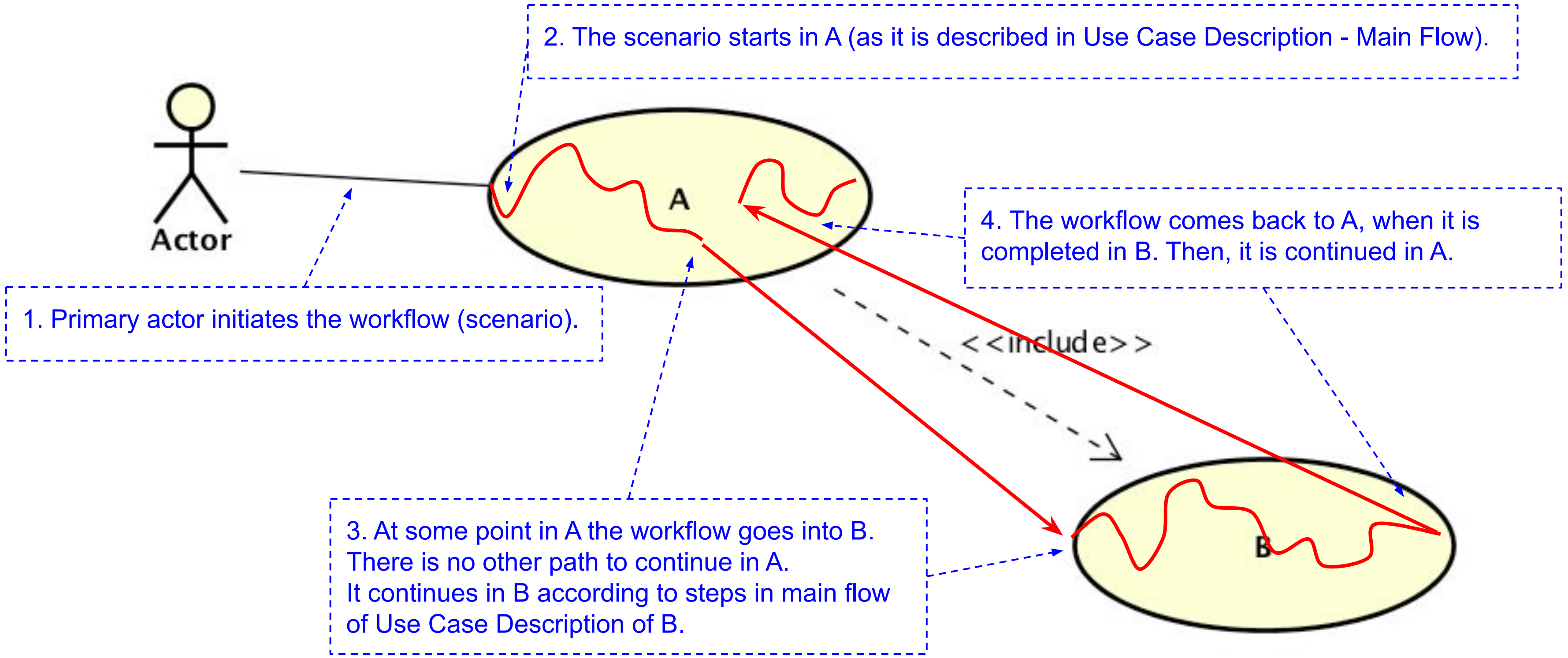
- specifying that the behaviour defined for the inclusion use case is to be inserted into the behaviour defined for the base use case.



Relationships: include


Semantics: The sequence of events in Base UC reuses all the steps of the Inclusion UC.
<<include>>
Base UC
Inclusion UC

Use Case - include



2. The scenario starts in A (as it is described in Use Case Description - Main Flow).

1. Primary actor initiates the workflow (scenario).

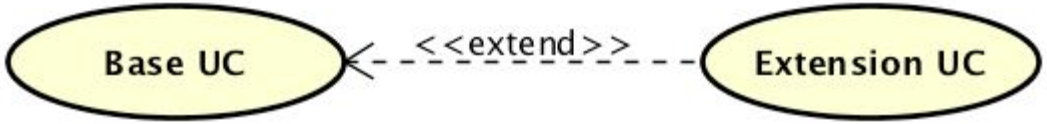
4. The workflow comes back to A, when it is completed in B. Then, it is continued in A.

3. At some point in A the workflow goes into B.
There is no other path to continue in A.
It continues in B according to steps in main flow of Use Case Description of B.

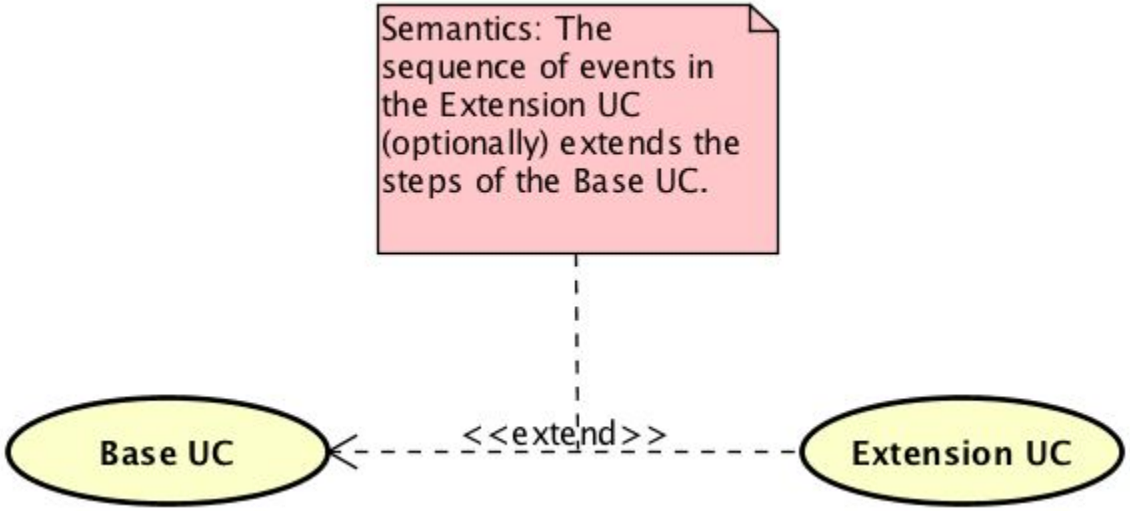
Relationships: extend

A relationship from an ***extension*** use case to a ***base*** (extended) use case.

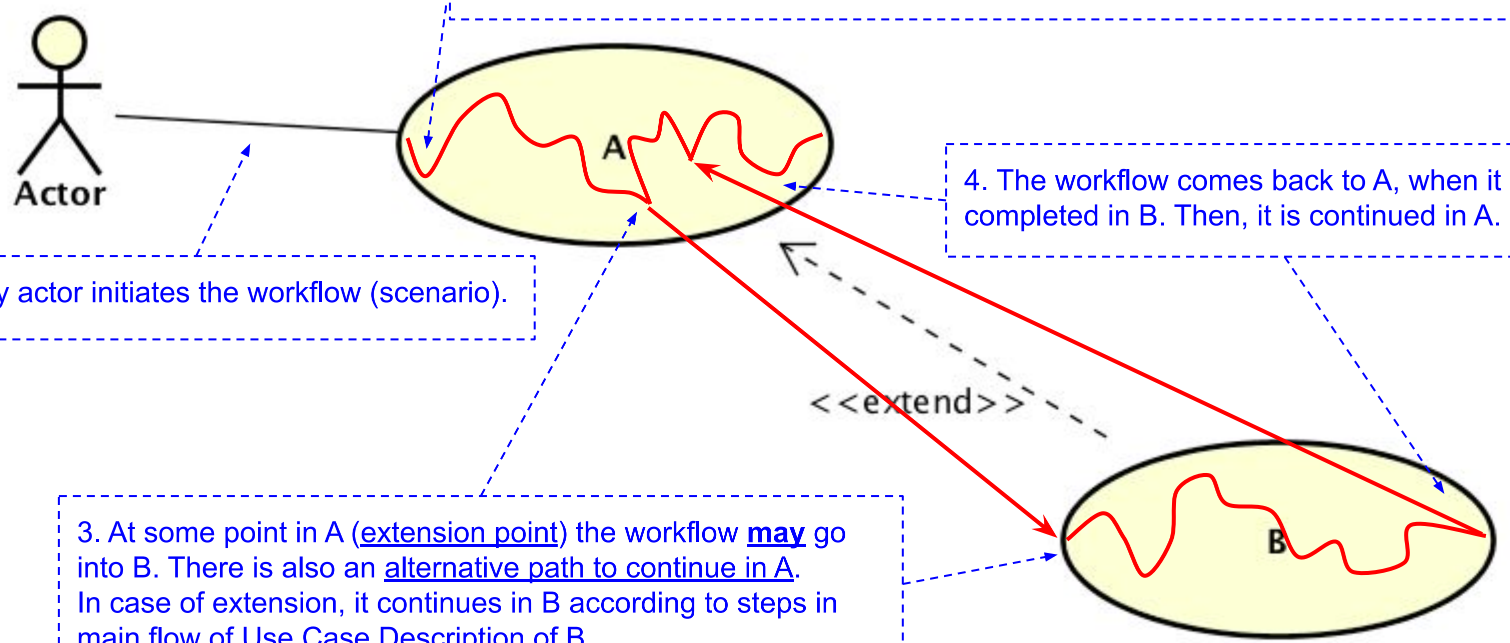
- specifying how the behaviour defined for the extension use case can be inserted into the behaviour defined for the base use case.



Relationships: extend


Semantics: The sequence of events in the Extension UC (optionally) extends the steps of the Base UC.
<<extend>>
Base UC
Extension UC

Use Case - extend



2. The scenario starts in A (as it is described in Use Case Description - Main Flow).

1. Primary actor initiates the workflow (scenario).

4. The workflow comes back to A, when it is completed in B. Then, it is continued in A.

3. At some point in A (<u>extension point</u>) the workflow <u>**may**</u> go into B. There is also an <u>alternative path to continue in A</u>. In case of extension, it continues in B according to steps in main flow of Use Case Description of B.

Remember ...

"In your use case diagram, check if all your use cases are ***reachable***.

Otherwise, there are isolated functionalities (use cases)."

Some notes (i)

- Do not get confused “extend” in use case relationships with “extend” in Java.
    - Check what extend in Java means.
- Be careful of the arrows for “include” and “extend”.
    - Direction of the arrows has precise semantic.

Some notes (ii)

- Note that an Use Case diagram without any actor is
  - WRONG!
- Note that a diagram with isolated actors (without any connection to use cases) is
  - WRONG!

Some notes (iii)

- Relating use cases affect use case descriptions.
- Be careful how generalization, extend and include affect the descriptions of the use cases.

More: Read here [1] and here [2]

[1] https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-use-case-diagram/

[2] https://thebadoc.com/ba-techniques/f/use-case-description-basics

Questions

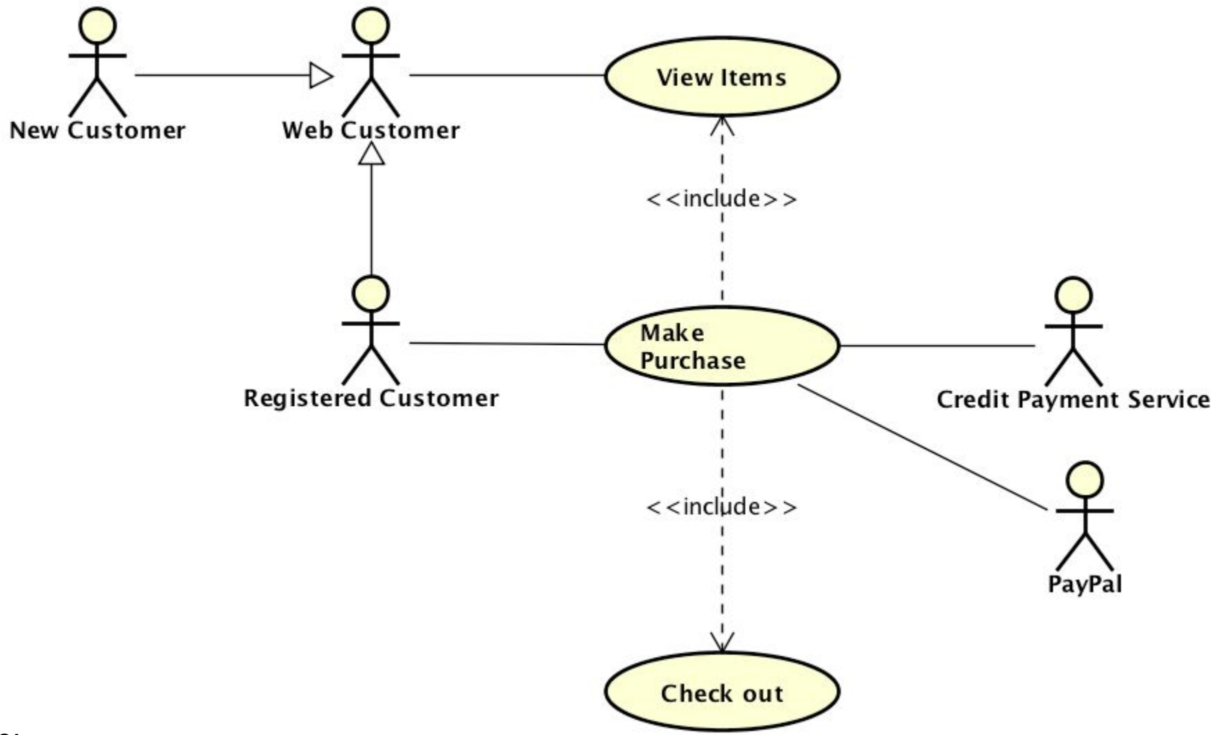
Part 3:

- What is the difference between *extend* and *include*?
- What is the semantics of generalization?
    - Pay attention on how the arrows point and how the information can be inherited or not.

Exercises

TO DO IN CLASS!

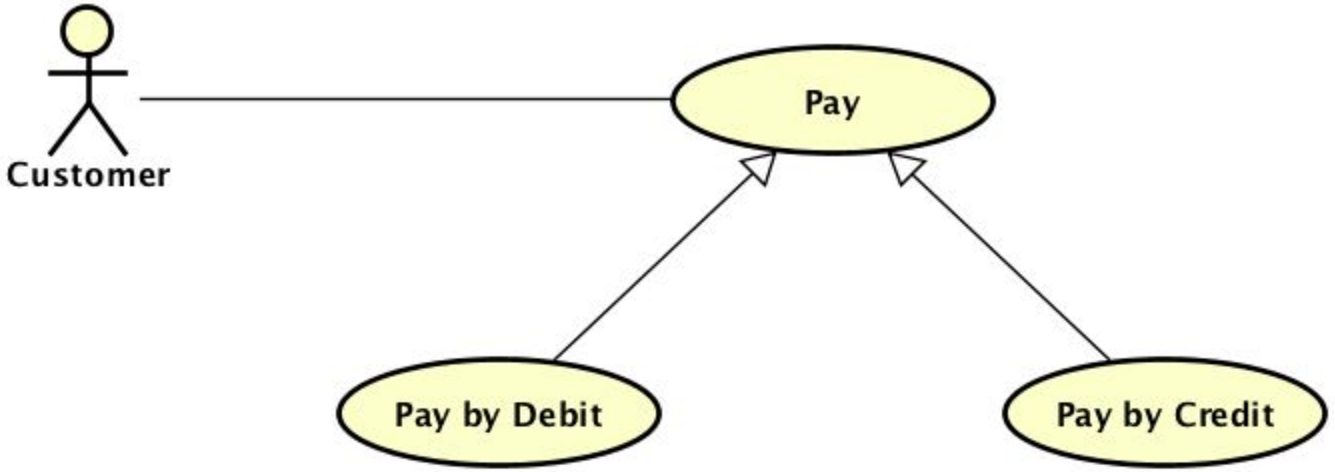
Explain the system



More explanation can be found here:
[http://www.uml-diagrams.org/examples/online-shopping-use-case-diagram-example.html?context=uc-examples]

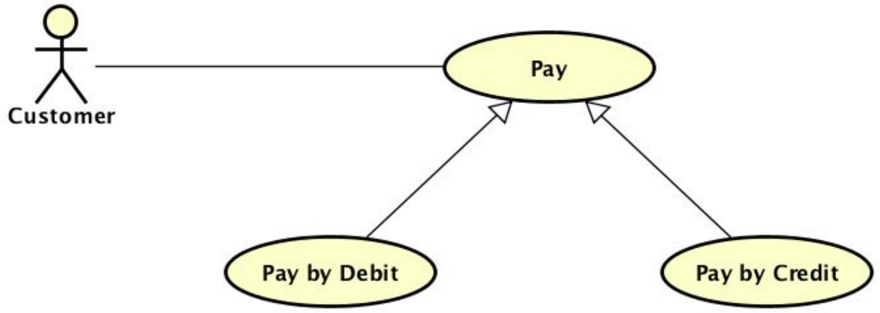
Explain:

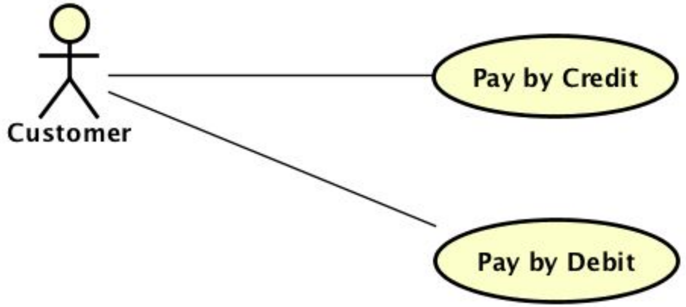
Discuss: How many options does a customer have for payment?



Discuss: design choice

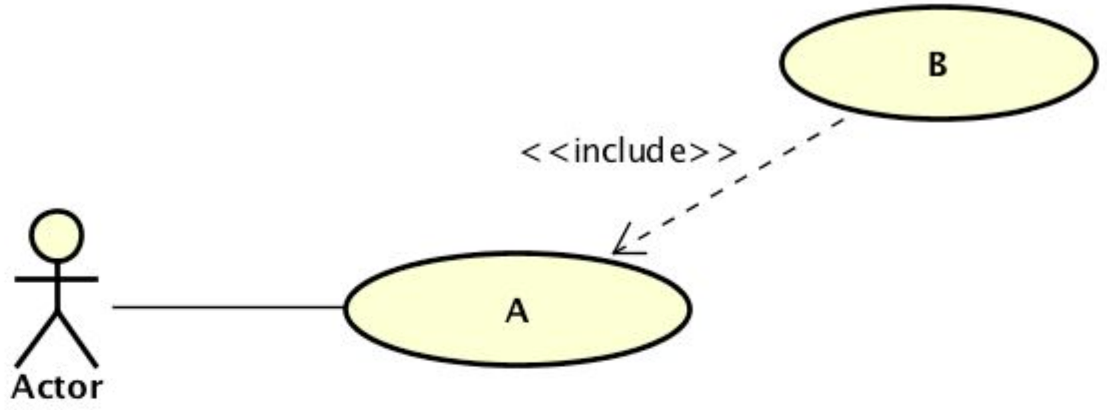
Compare and discuss the following models.





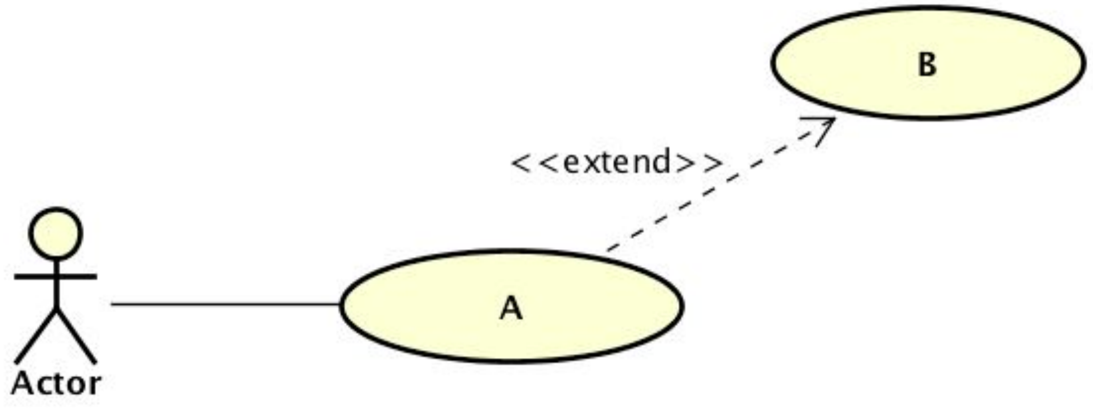
Check: Use Case

What is the problem of this model?



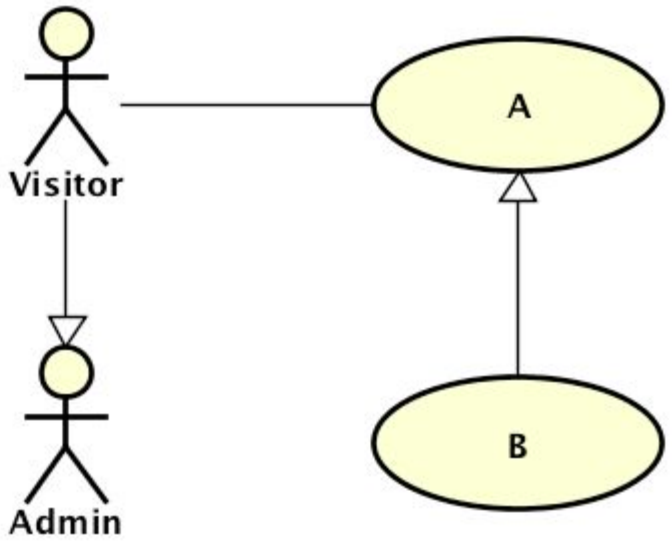
Check: Use Case

What is the problem of this model?



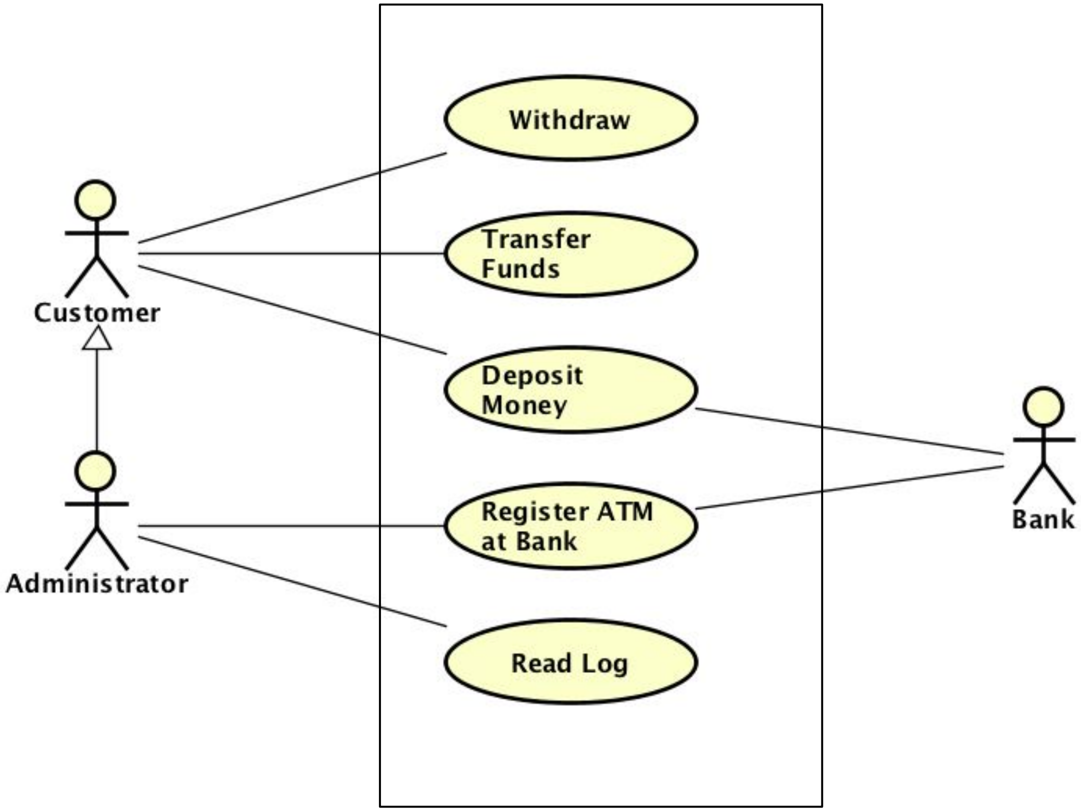
Check: Use Case

What is the problem of this model?

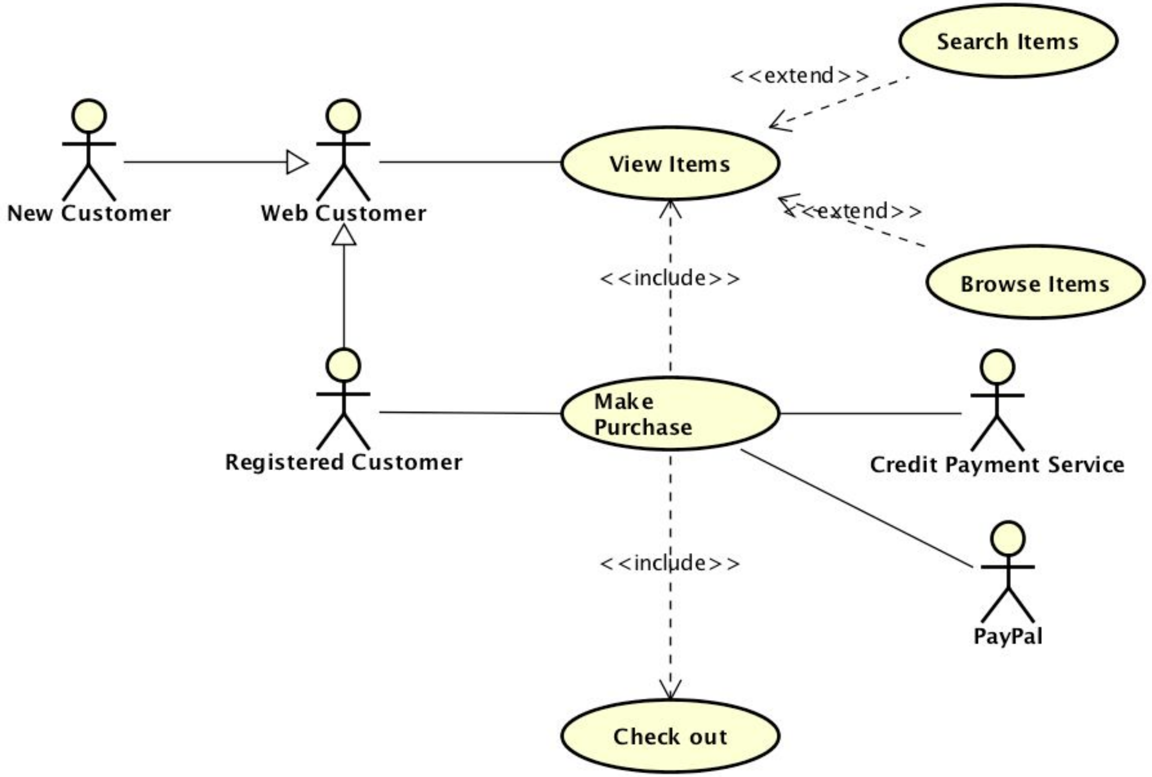


Discuss

Explain and Improve the model.

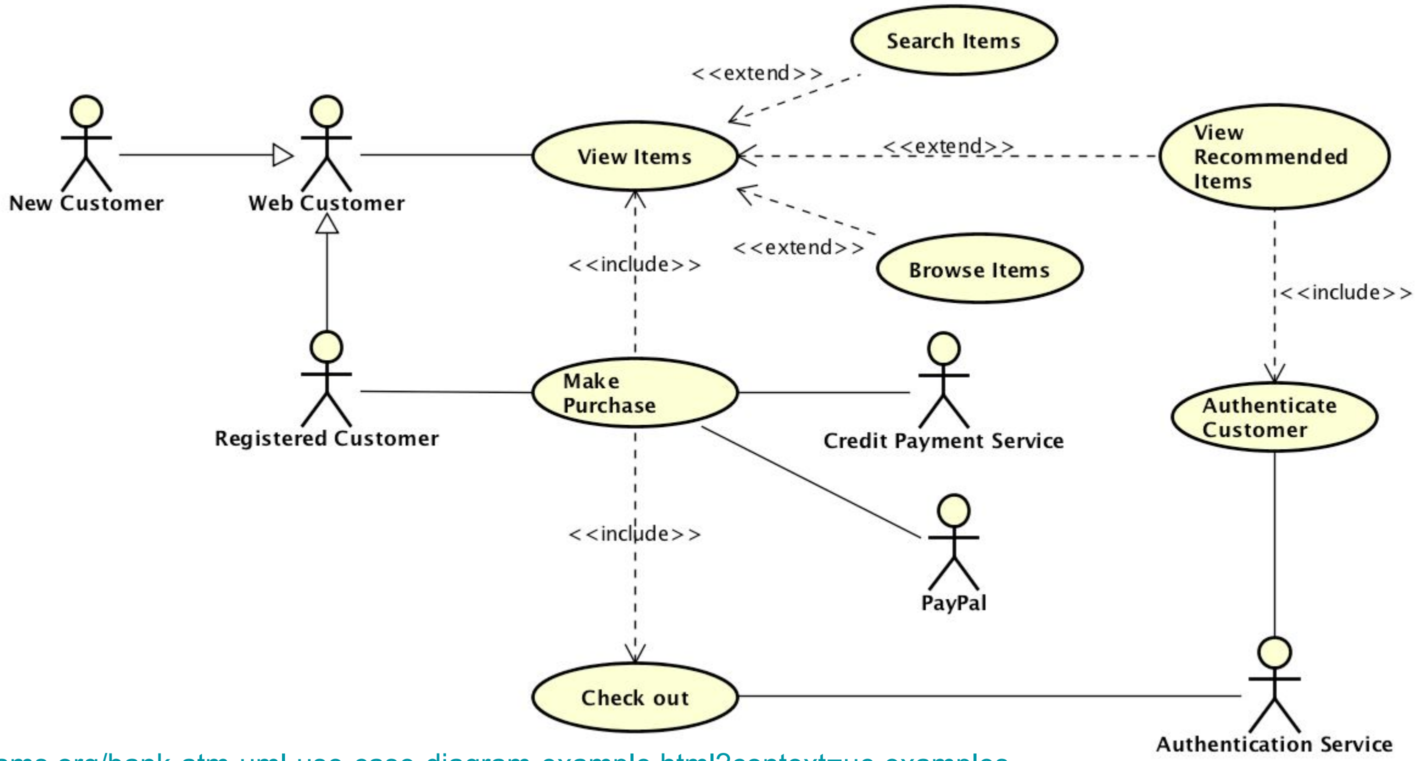


Exercise: Explain the system



More explanation can be found here:
[http://www.uml-diagrams.org/examples/online-shopping-use-case-diagram-example.html?context=uc-examples]

Explain UCD - Online Shopping



http://www.uml-diagrams.org/bank-atm-uml-use-case-diagram-example.html?context=uc-examples
http://agilemodeling.com/essays/useCaseReuse.htm#Figure1

Summary

Remember:

- To model functionalities of a system use use case diagrams.
- Use Case is the class of related events in the system to satisfy a requirement of a user.
- An actor is the class of users (human, device or software) that interact with the system.
- The communication between an actor and the system is classified as an association relationship.

Summary

Remember:

- UML helps us to understand and handle the complexity of software systems.
- UML is a modeling language independent from PLs, platforms and domain.
- UML covers various aspects of the system in all development phases.
- UML proposes several diagrams to cover different views of the system.

Summary

Remember:

- An actor can inherit the behaviour from another actor.
- An use case can:
  - Inherit from another use case,
  - Extend another use case,
  - Include another use case.

ROTTERDAM UNIVERSITY

UNIVERSITY OF APPLIED SCIENCES

exceed expectations