Software Engineering I (02161)

Week 2

Assoc. Prof. Hubert Baumeister

DTU Compute
Technical University of Denmark

Spring 2017

Contents

Basic Activities in Software Development

- Understand and document what kind of the software the customer wants
  - → Requirements Analysis / Engineering
- Determine how the software is to be built
  - → Design
- Build the software
  - → Implementation
- Validate that the software solves the customers problem
  - → Testing

Requirements Analysis

Requirements Analysis

Understand and *document* the kind of software the customer wants

- Describe mainly the *external behaviour* of the system and *not* how it is realised
  - → *what* not *how*
- Techniques for discovering, understanding, and documentation
  - Glossary and Business Processes: Understand the problem domain
  - Use Cases: Understand and discover the functionality of the system
  - User Stories: Understand and discover the functionality of the system

Domain model

Types of Requirements

- Functional Requirements
  - E.g. the user should be able to plan and book a trip
- Non-functional Requirements
  - All requirements that are not functional
  - E.g.
    - Where should the software run?
    - What kind of UI the user prefers?
    - What is the response time?
    - . . .

Who writes requirements?

- The customer:
  - User requirements
- The contractor together with the customer
  - System requirements
    - The requirements for the software development team how to build the system

    → more detailed than user requirements

    → basis for a contract between customer and contractor

Travel Agency Example: User Requirements

The travel agency TravelGood comes to you as software developers with the following proposal for a software project:

- Problem description / user requirements

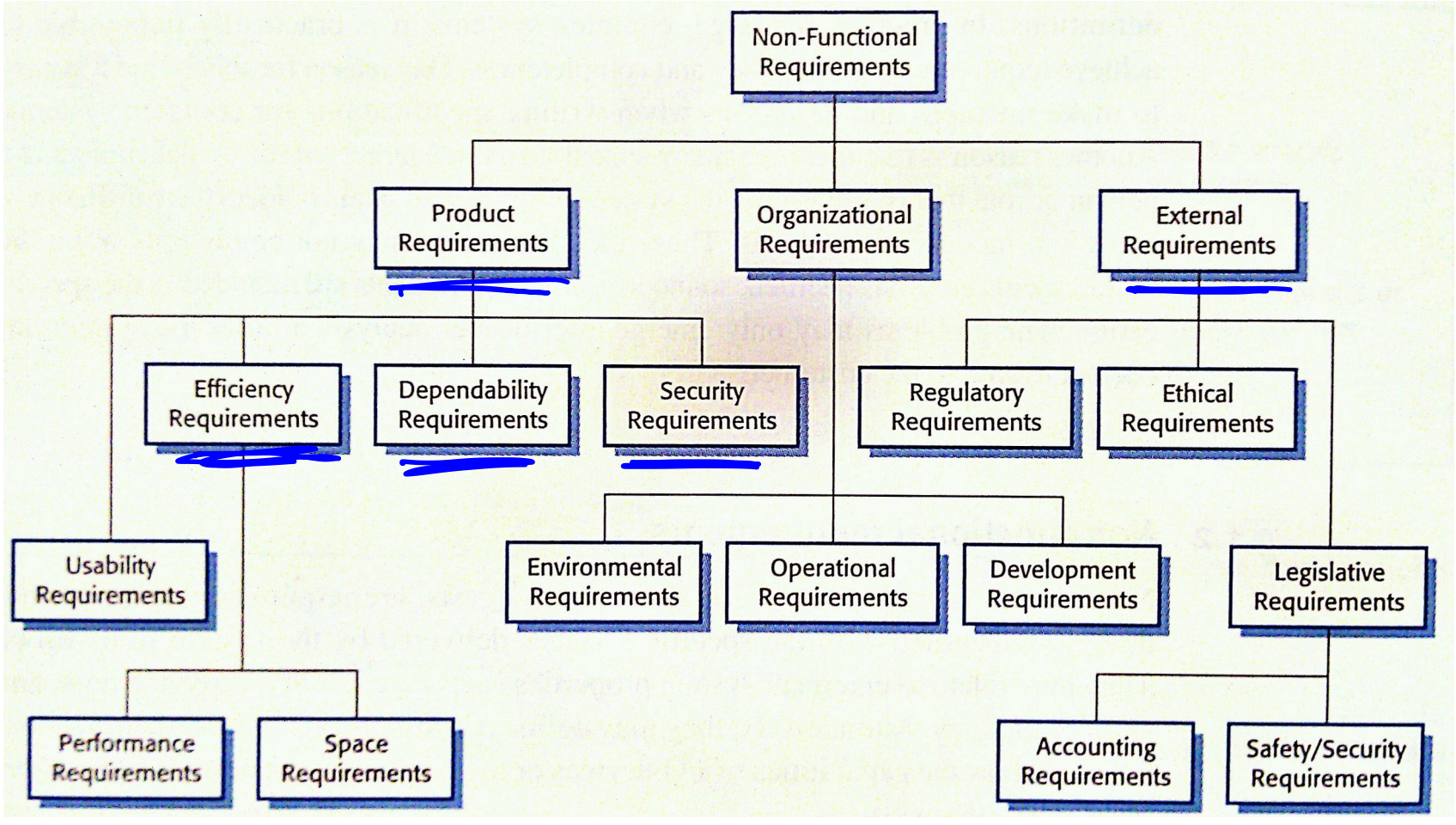
  "TravelGood wants to offer a trip-planning and booking application to its customers. The application should allow the customer to plan trips consisting of flights and hotels. First the customer should be able to assemble the trip, before he then books all the flights and hotels in on step. The user should be able to plan several trips. Furthermore it should be possible to cancel already booked trips."

$\rightarrow$ Not enough: Text needs to be analysed and system requirements extracted

Travel Agency

- Functional Requirements
  - ”plan a trip, book a trip, save a planned trip for later booking, . . . ”
- Non-functional requirements
  - ”System should be a Web application accessible from all operating systems and most of the Web browsers”
  - ”It must be possible to deploy the Web application in a standard Java application servers like GlassFish or Tomcat”
  - ”The system should be easy to handle (it has to a pass a usability test)” not clear

Non exclusive checklist of non-functional requirements



Characteristics of good requirements

- Testability
  - $\rightarrow$ manual/automatic acceptance tests
- Measurable
  - Not measurable: *The system should be easy to use by medical staff and should be organised in such a way that user errors are minimised*

Characteristics of good requirements

- Testability
  - → manual/automatic acceptance tests
- Measurable
  - Not measurable: *The system should be easy to use by medical staff and should be organised in such a way that user errors are minimised*
  - Measurable: *Medical staff shall be able to use all the system functions after four hours of training. After this training, the average number of errors made by experienced users shall not exceed two per hour of system use.*

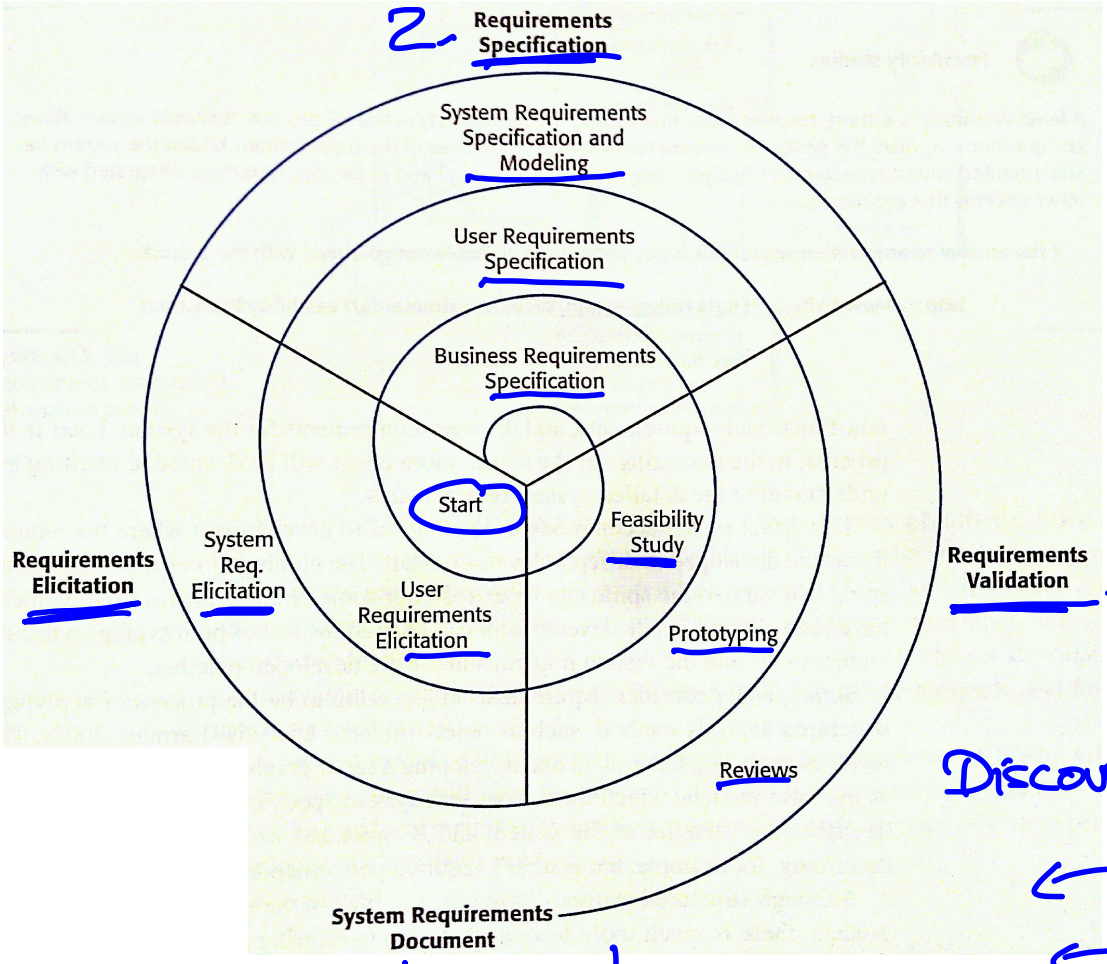
Possible measures

Property	Measure
Speed	Processed transactions/second User/event response time Screen refresh time
Size	Mbytes Number of ROM chips
Ease of use	Training time Number of help frames
Reliability	Mean time to failure Probability of unavailability Rate of failure occurrence Availability
Robustness	Time to restart after failure Percentage of events causing failure Probability of data corruption on failure
Portability	Percentage of target dependent statements Number of target systems

Contents

Requirements engineering process

A spiral view of the requirements engineering process



Requirements Engineering Process: Techniques

- Elicitation / Discovery
  - Problem description
  - Interviews
  - Domain Model plus Business Processes
  - Use Cases
- Specification / Documentation
  - Use Cases / User Stories
- Validation
  - Inspection
    - Validity, Consistent, Complete, Realistic, . . .
  - Creation of tests → acceptance test

Contents

Discovery & Documentation of Requirements

Domain model

- Purpose: capture the **customer's knowledge** of the domain so that the **system builders** have the *same knowledge*
- Helps customer and system builders to speak the **same language**

→ Necessary to define the **terminology** used

  → *Glossary*

→ **Relationships** between terms are shown in a *class diagram*

  → Related to the concept of an **ontology**

→ If necessary, make **business processes** visible

  → Represented by UML Activity Diagrams

Glossary

glossary (plural glossaries)

”1. (lexicography) A list of *terms* in a particular **domain of knowledge** with the **definitions** for those terms.” (Wikitionary)

- List of terms with explanations
- Terms can be nouns (e.g. those mentioned in a problem description) but also verbs or adjectives e.t.c.
- List of terms with explanations
- Terms can be nouns (e.g. those mentioned in a problem description) but also verbs or adjectives e.t.c.
- *Warning*
  - Capture only knowledge relevant for the application
  - Don’t try to capture all possible knowledge

Example

Part of a glossary for a library application

Book

- A book is a is a conceptual entity in a library. A book is defined by its title, the name of his authors, the publisher and the edition. A library can have several copies of the same book.

Copy

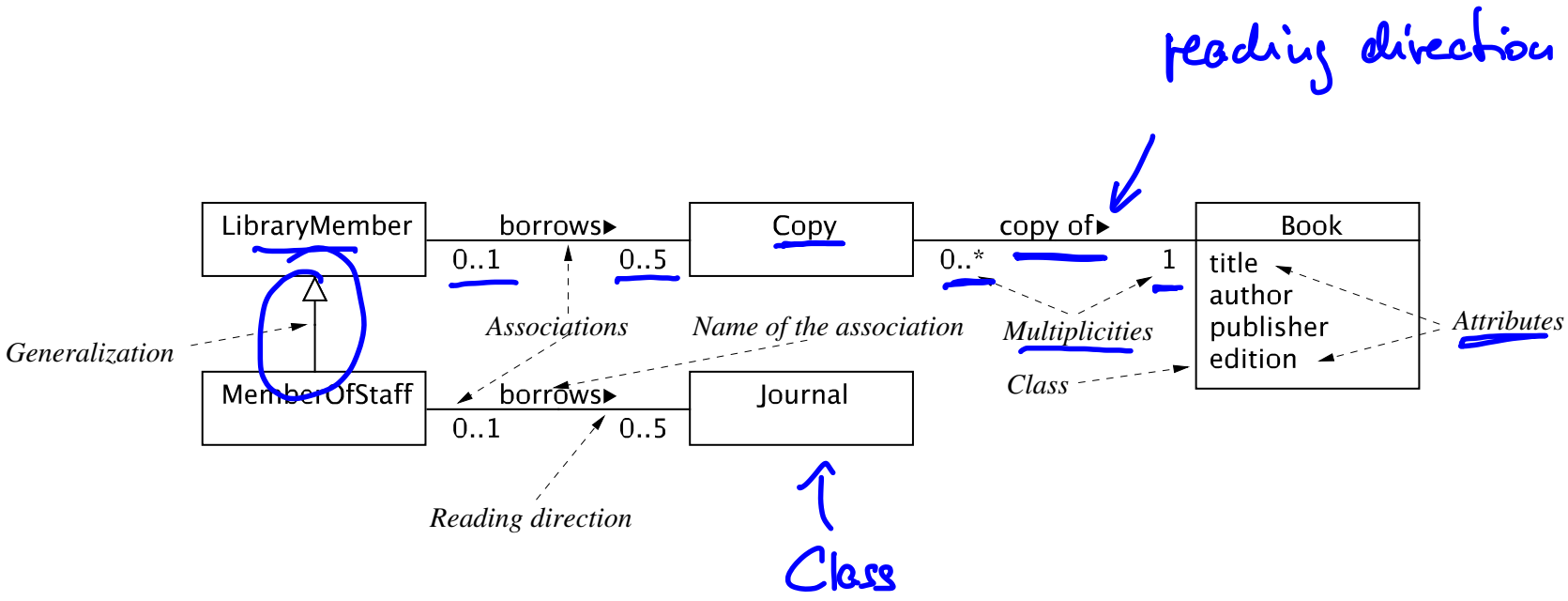
- A copy is a physical copy of a particular book. For example, the library has three copies of the book ”Using UML” by Perdiate Stevens. . . .

. . .

Terms and their relations

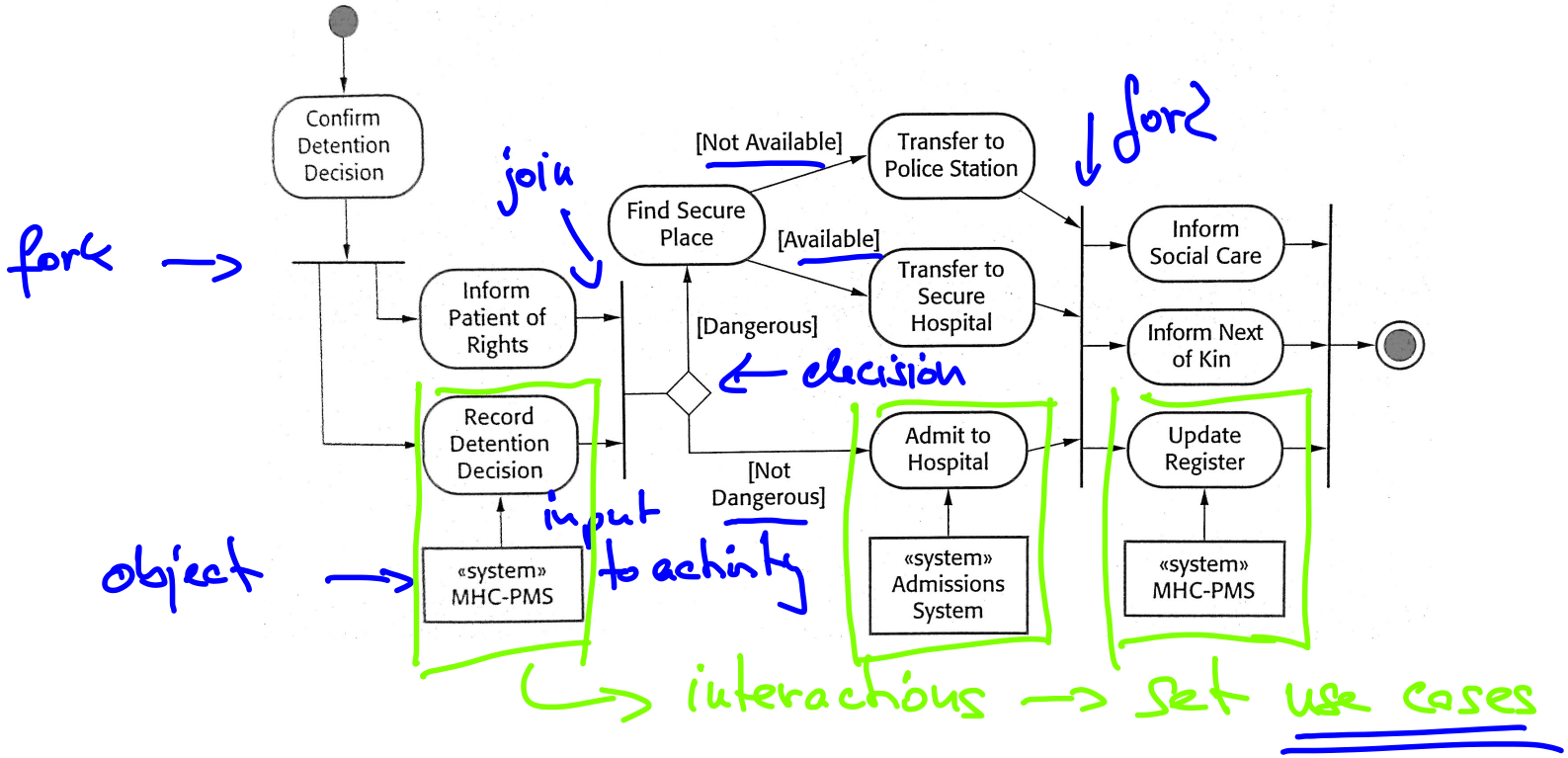
- Class diagrams
- Usually
  - Classes (for nouns)
  - Associations: for static relationships
  - Generalizations
  - Use of attributes possible
  - Often without **operations**
  - **verbs** $\rightarrow$ **operations**
- *Warning*
  - Shows *customer knowledge*
  - Should not be biased by implementation

Domain model (terms and their relations)



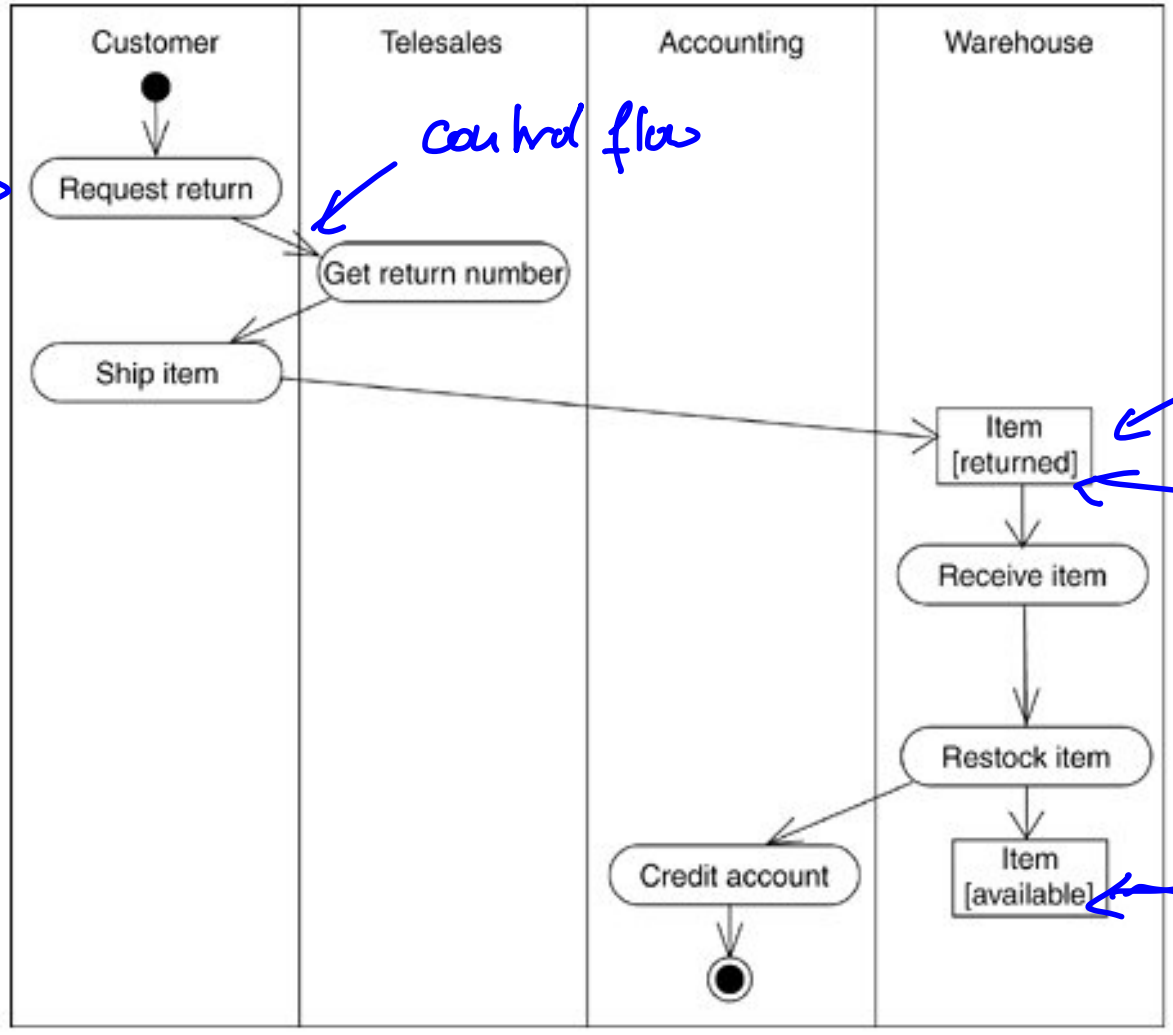
Contents

Activity Diagram: Business Processes

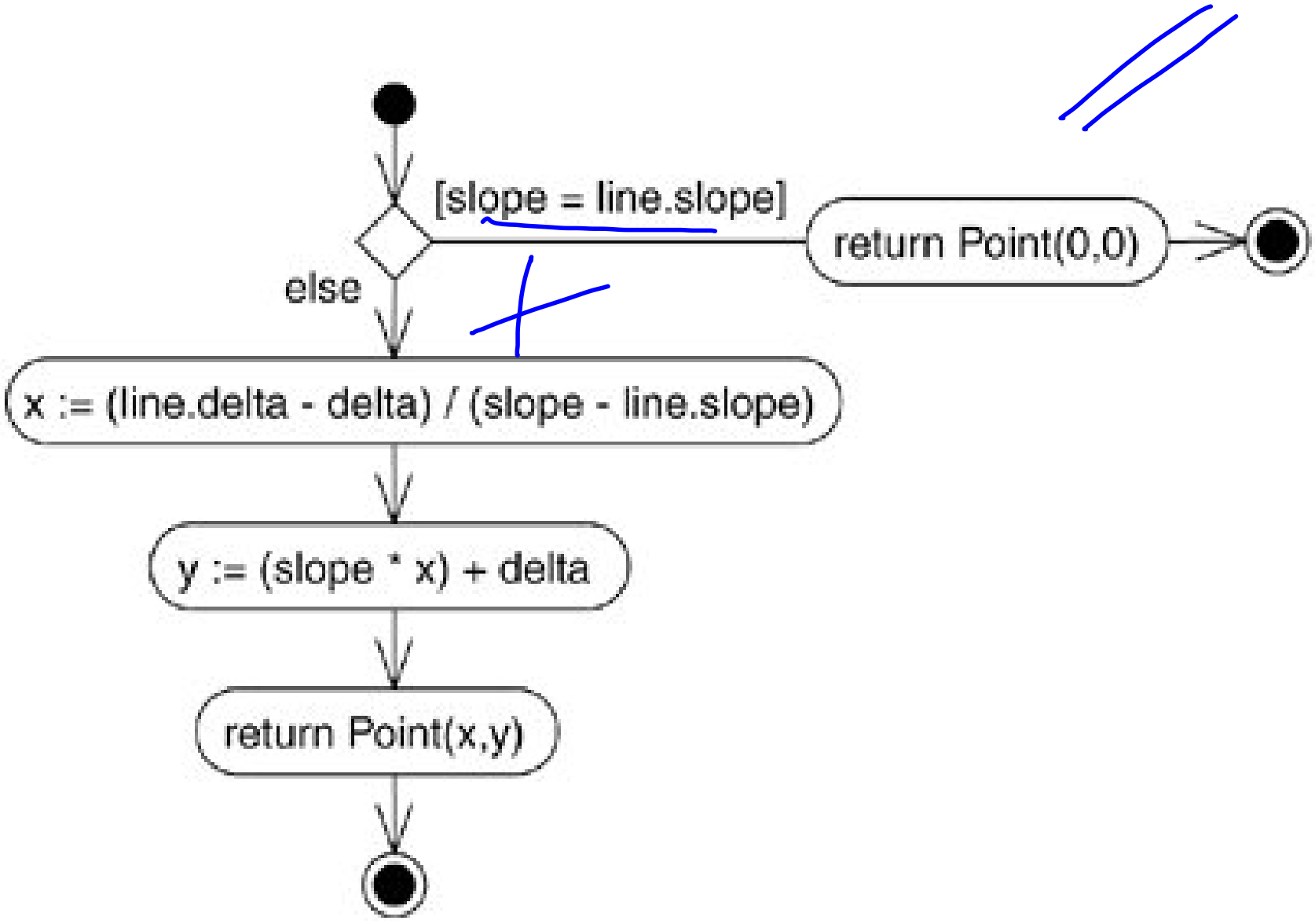


- Describe the *context* of the system
- Helps finding the requirements of a system
  - modelling business processes leads to suggestions for possible systems and ways how to interact with them
  - Software systems need to fit in into existing business processes

Activity Diagram Example Workflow



Activity Diagram Example Operation



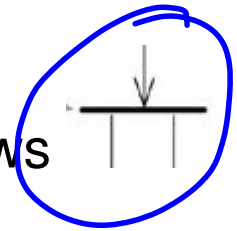
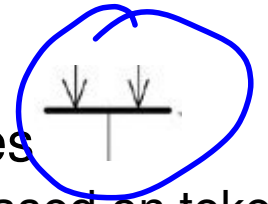
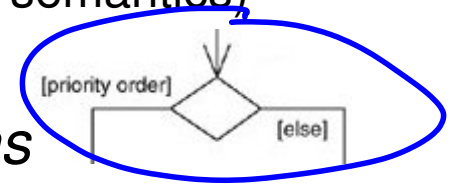
UML Activity Diagrams

- Focus is on *control flow* and *data flow*
- Good for showing *parallel/concurrent* control flow
- Purpose
  - Model business processes
  - Model workflows
  - Model single operations
- Literature: UML Distilled by Martin Fowler

UML User Manual

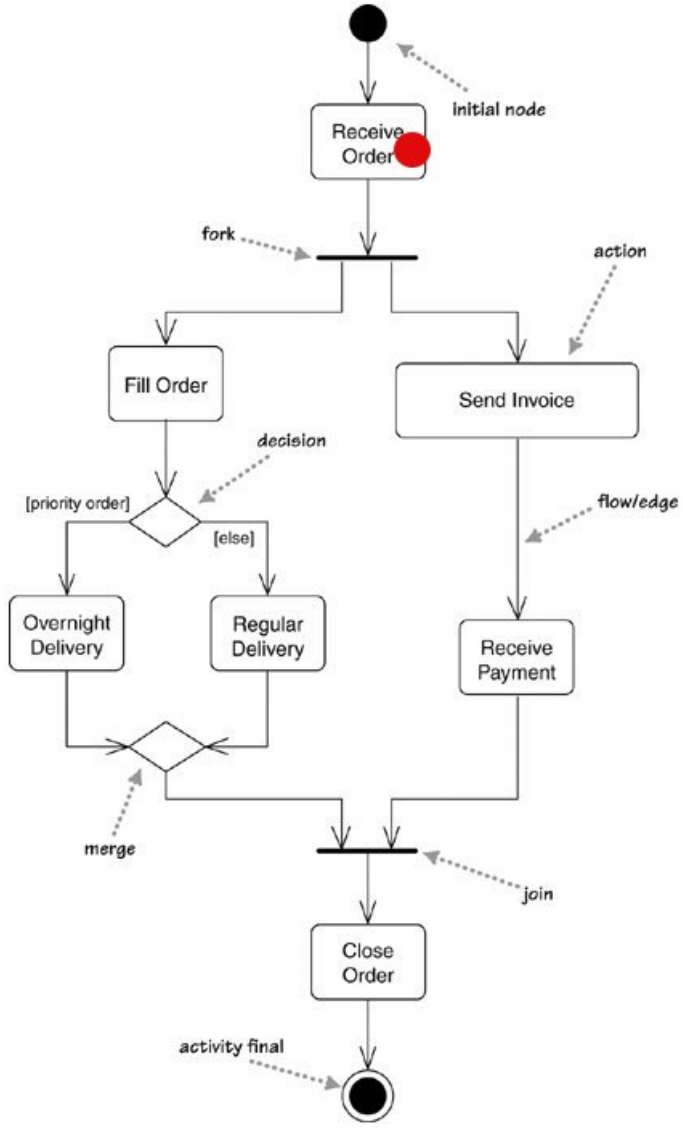
Activity Diagram Concepts

- *Actions* 

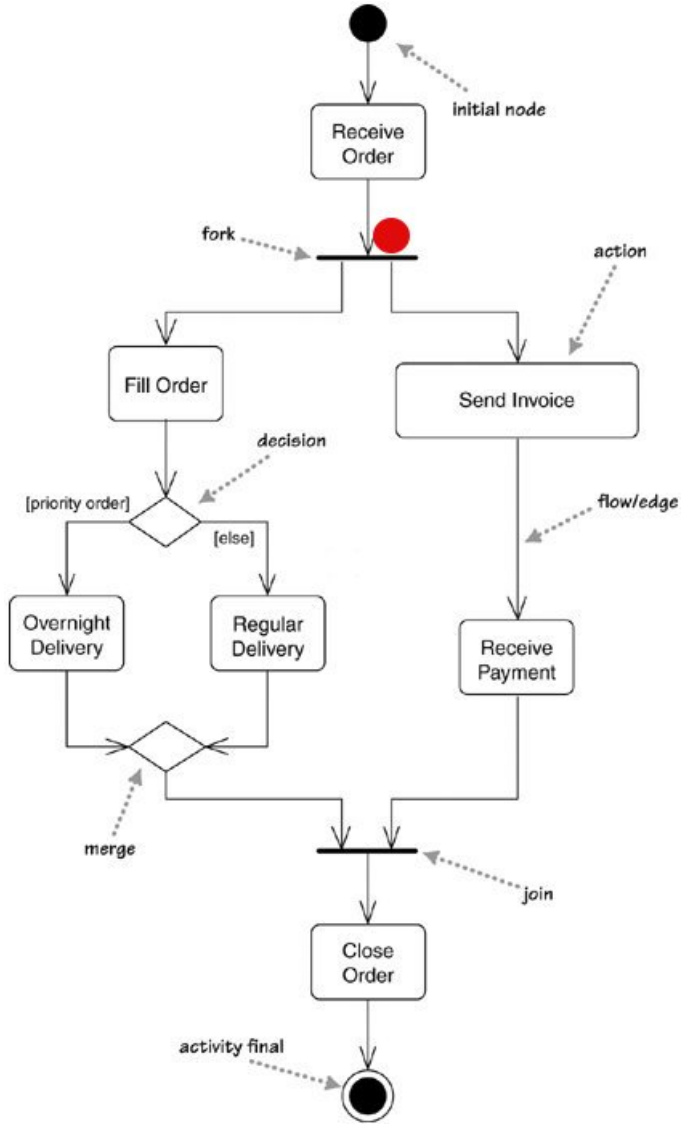
box with rounded edges
  - Are atomic
  - E.g Sending a message, doing some computation, raising an exception, . . .
    - UML has approx. 45 Action types
- *Concurrency*
  - Fork: Creates concurrent flows 
    - Can be true concurrency
    - Can be interleaving
  - Join: Synchronisation of concurrent activities 
    - Wait for all concurrent activities to finish (based on token semantics)
- *Decisions* 

  - Notation: Diamond with conditions on outgoing transitions
  - `else` denotes the transition to take if no other condition is satisfied

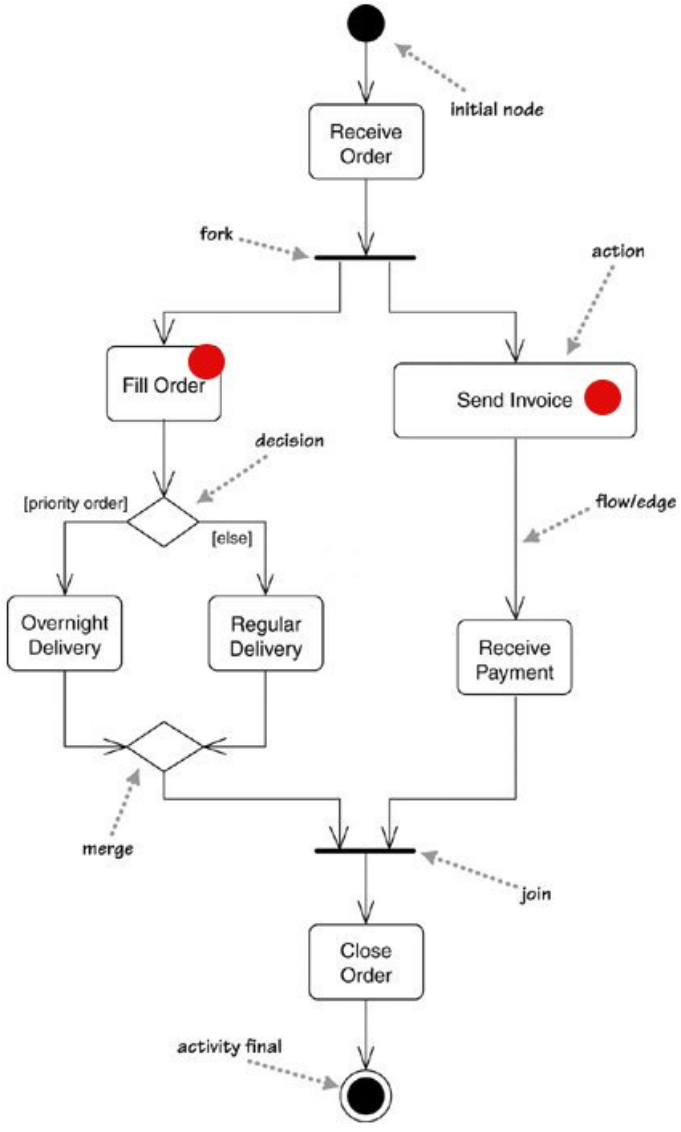
Activity Diagrams Execution



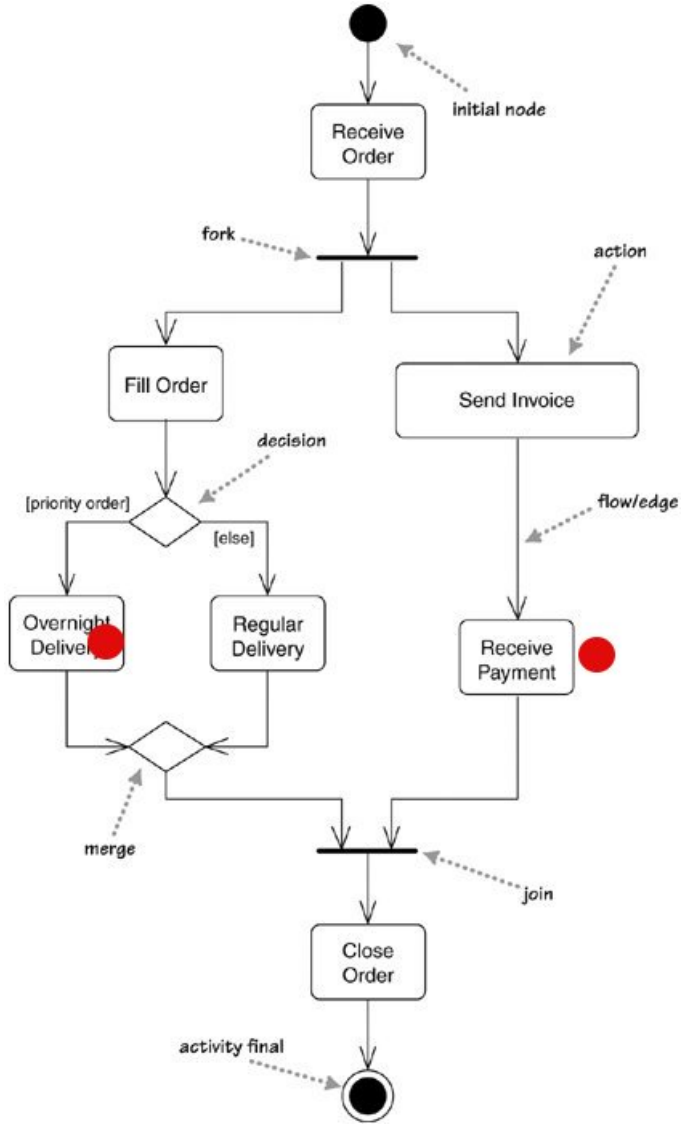
Activity Diagrams Execution



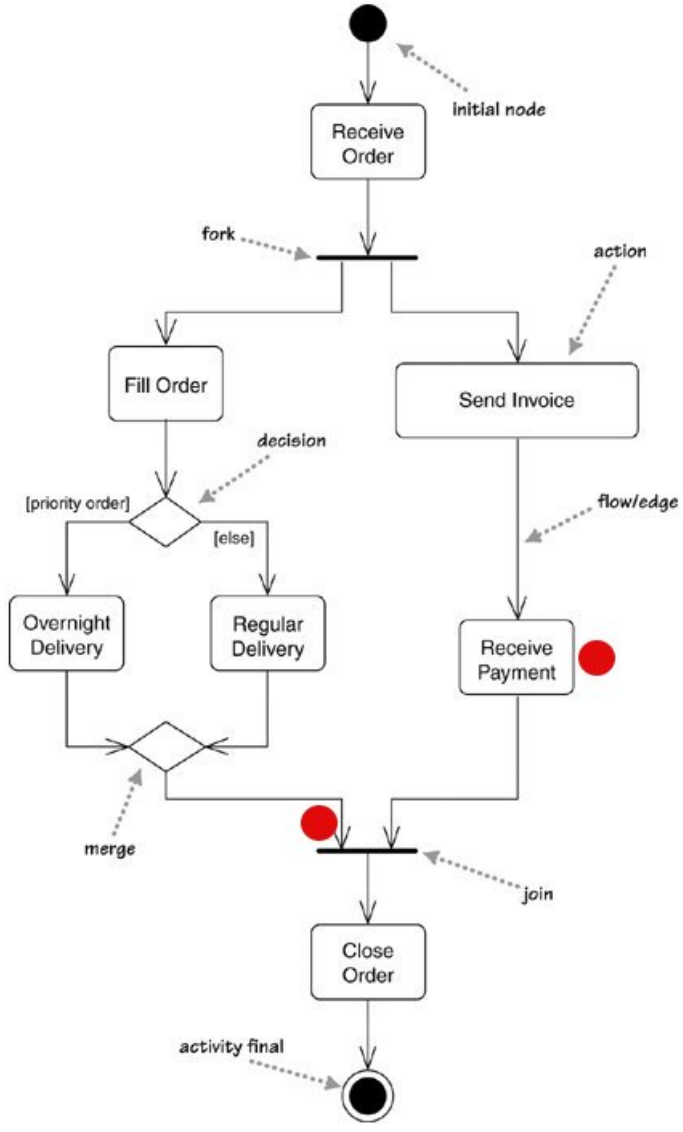
Activity Diagrams Execution



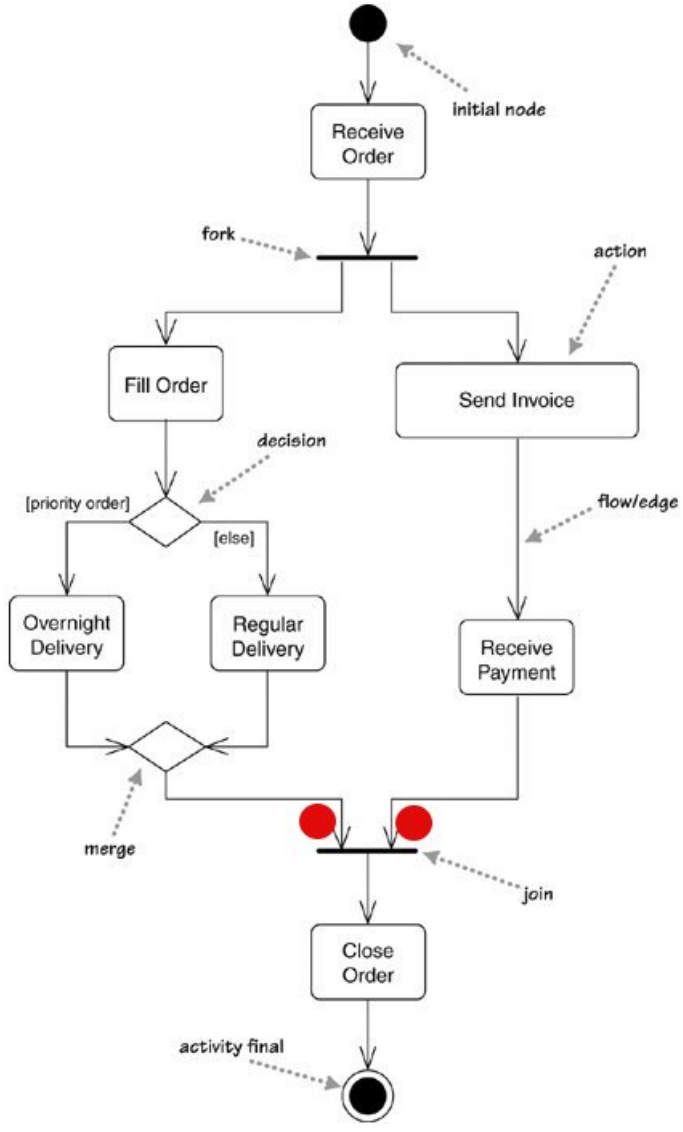
Activity Diagrams Execution



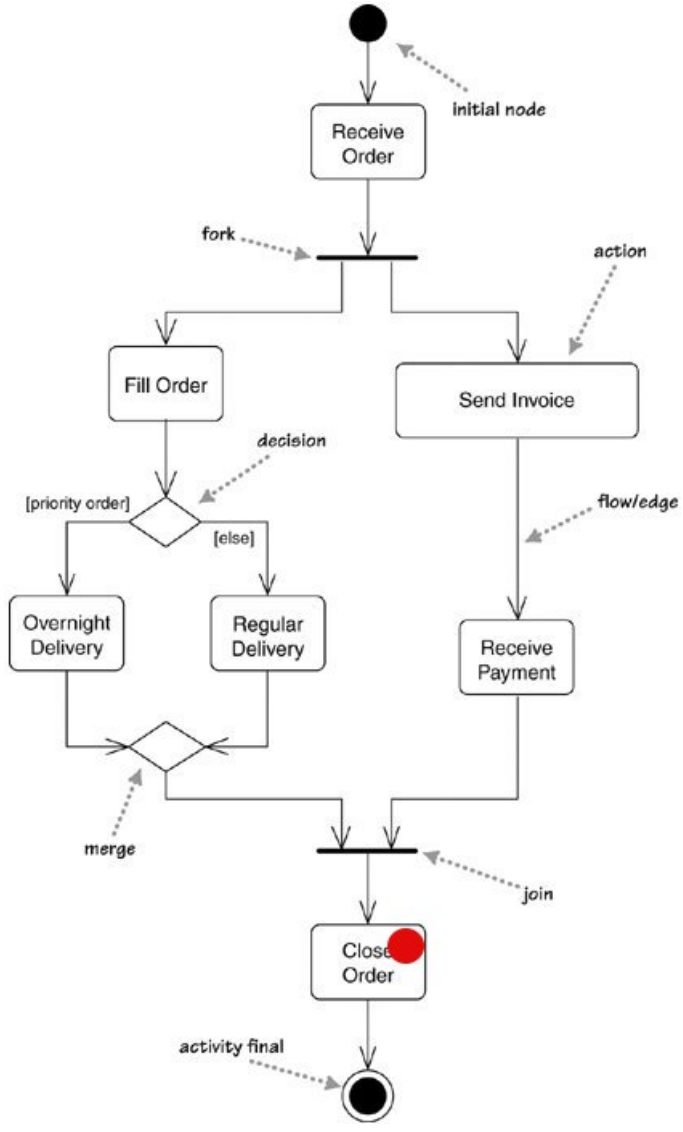
Activity Diagrams Execution



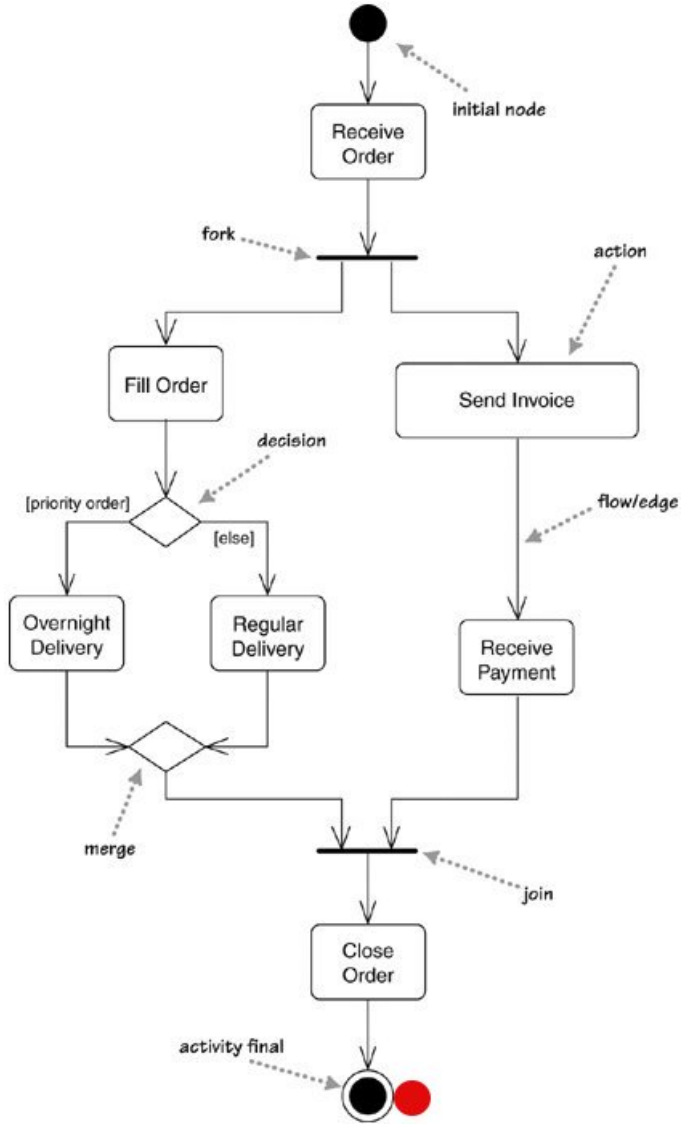
Activity Diagrams Execution



Activity Diagrams Execution

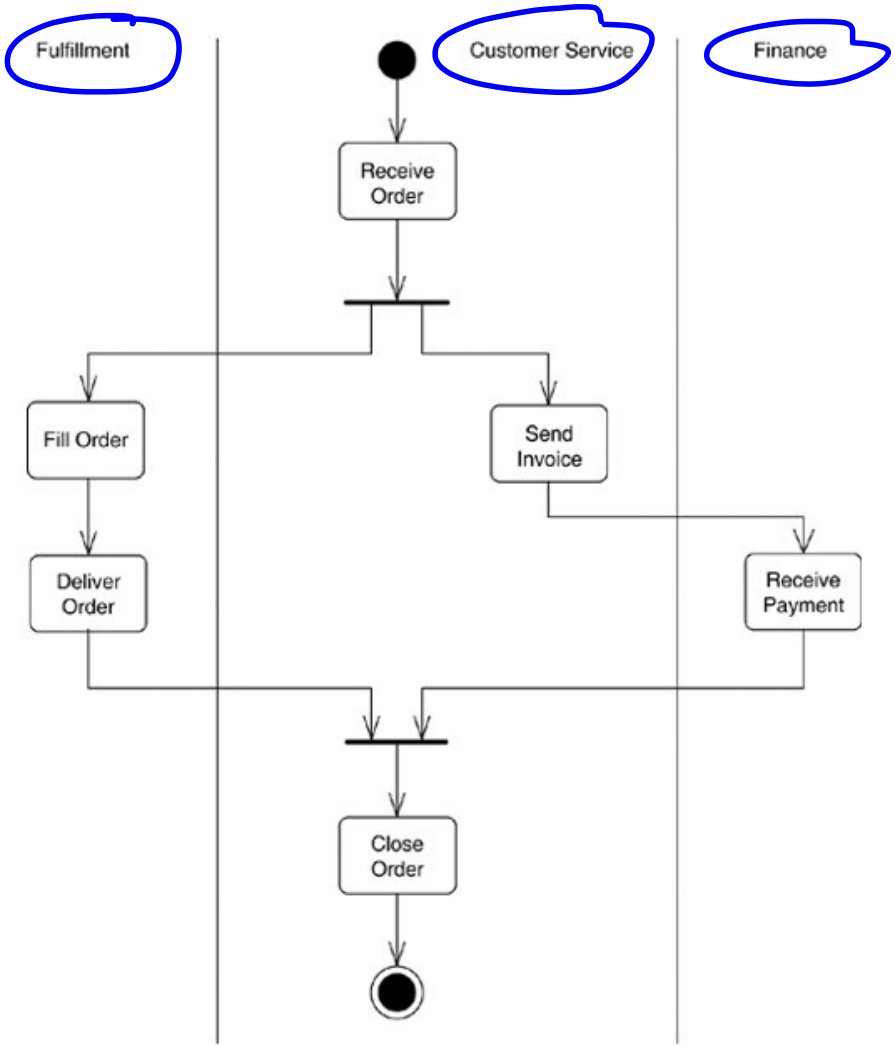


Activity Diagrams Execution

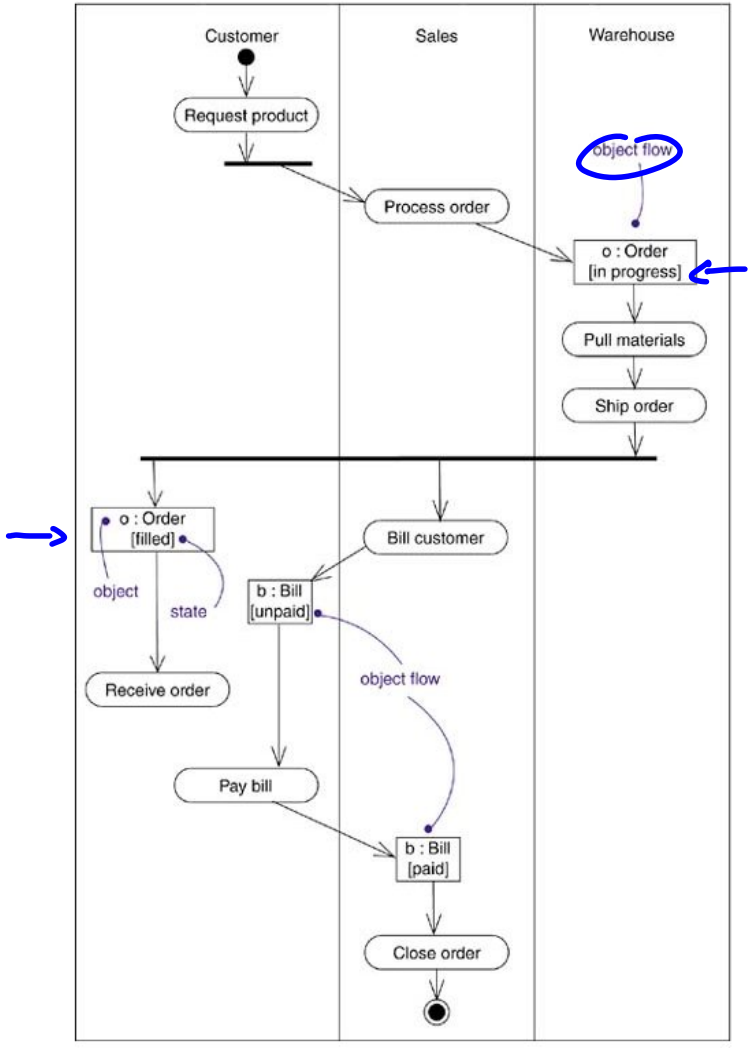


Swimlanes / Partitions

- Swimlanes show **who** is performing an activity

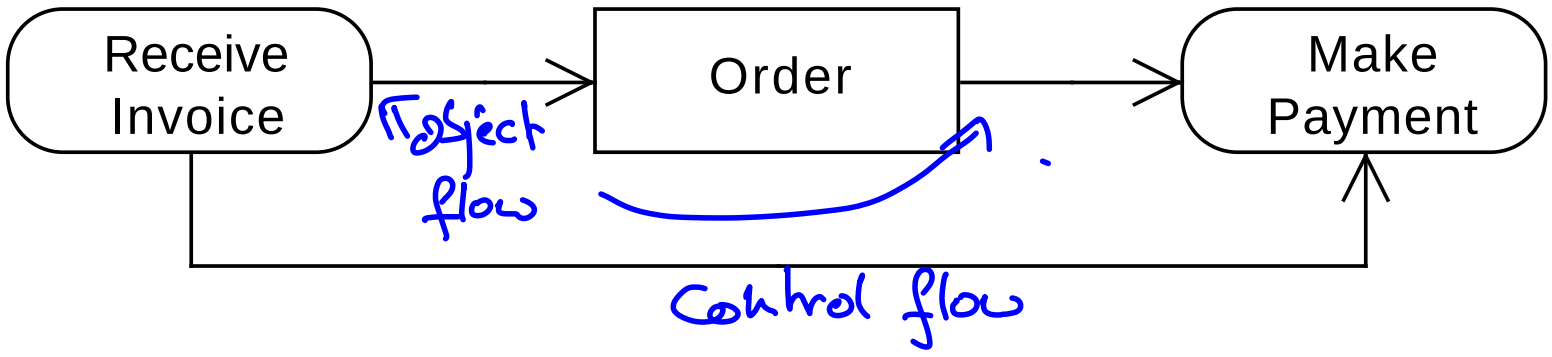


Objectflow example



Data flow and Control flow

- *Data flow* and *control flow* are shown:



- Control flow can be omitted if implied by the data flow:

Use of Activity Diagrams

- Focus on concurrent/parallel execution
- Requirements phase
  - To model business processes / workflows to be automated
- Design phase
  - Show the semantics of one operation
    - Close to a graphic programming language

Contents

Use Case

Use cases discover and document functional requirements

→ Naming convention: ”Do something” (= functionality): ”verb + noun”

Use Case

A Use Case is a set of interaction scenarios of one or several actors with the system serving a common goal.

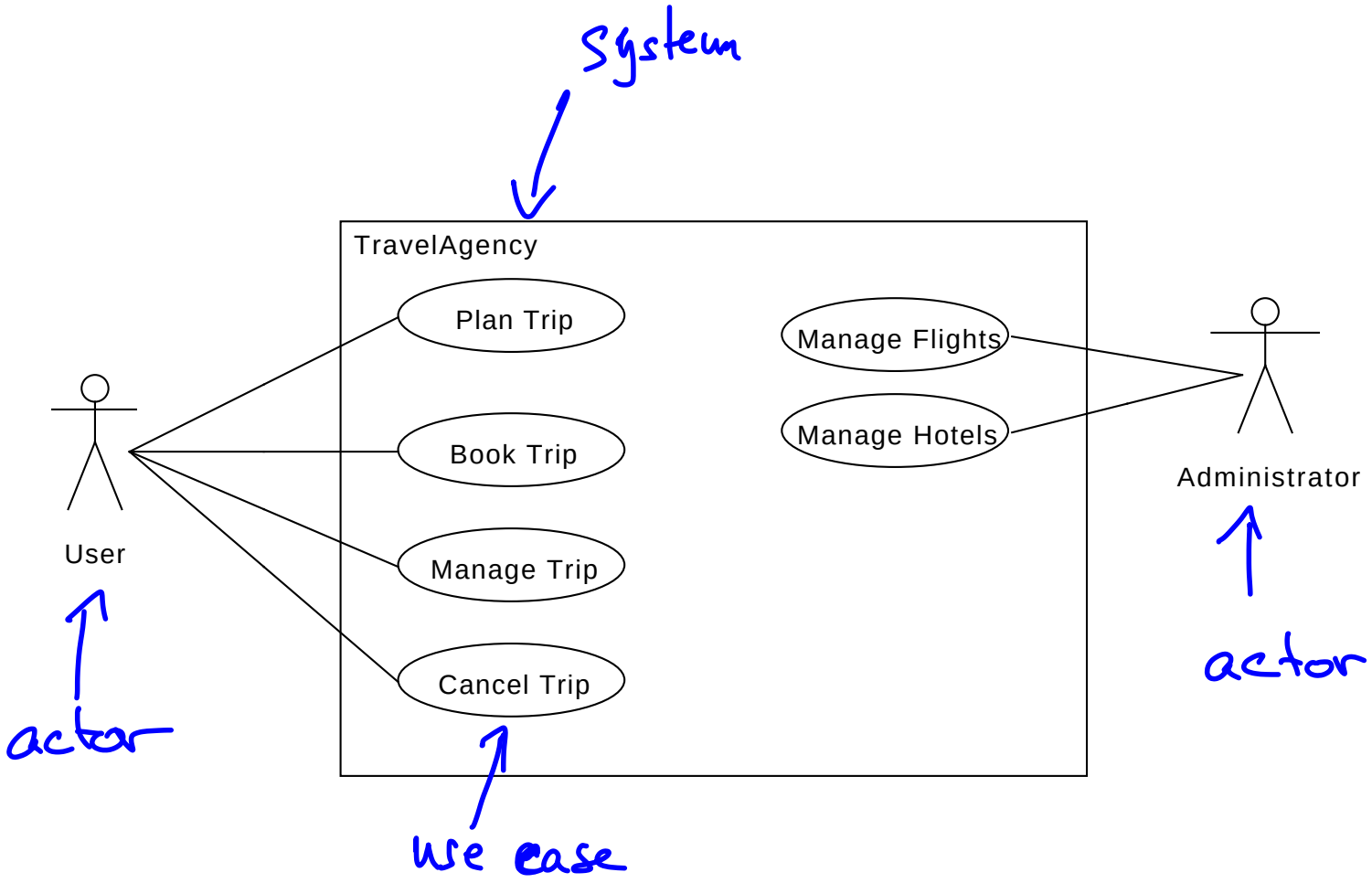
Use Case Diagram

A use case diagram provides and overview over the use cases of a *system* and *who* is using the functionality.

Detailed Use Case description

A detailed use case description describes the interaction between the user and the system as a set of *scenarios*

Use Case Diagram

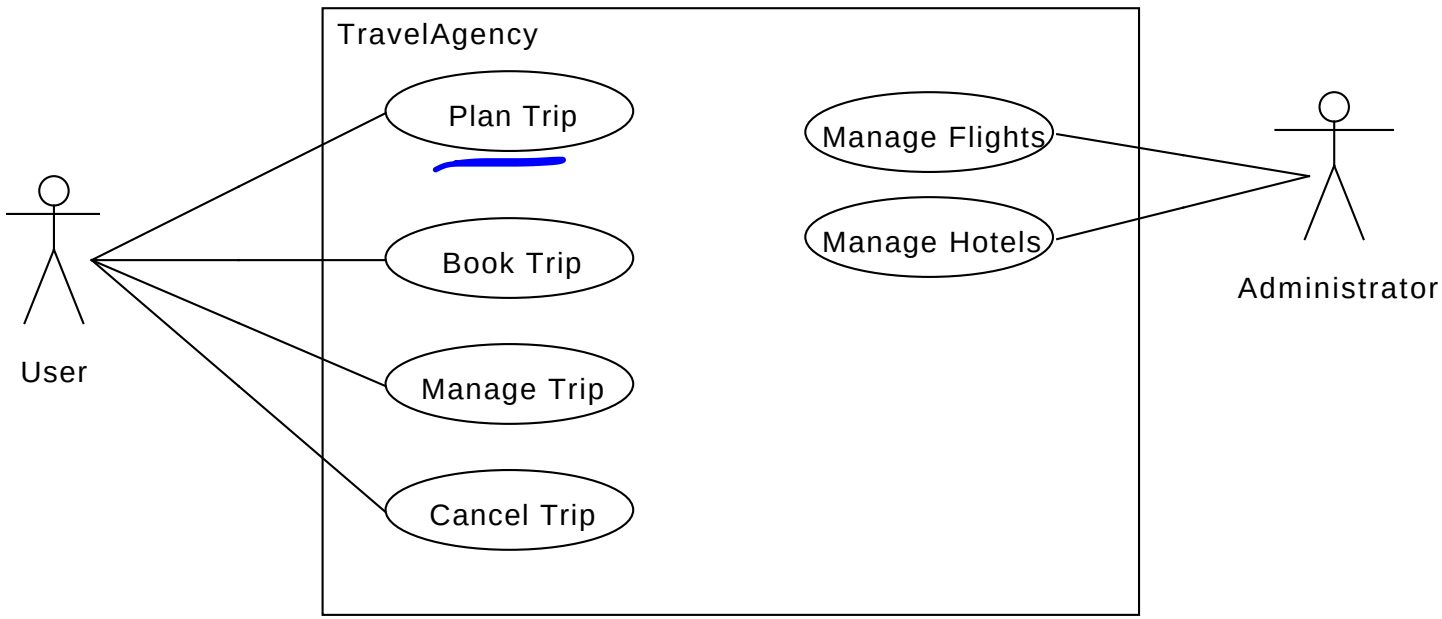


Types of use case diagrams

a) *Business use cases* (*kite level* use cases (from Alistair Cockburn))
b) *System use cases* / *sea level* use cases (*fish level* use cases (from Alistair Cockburn)

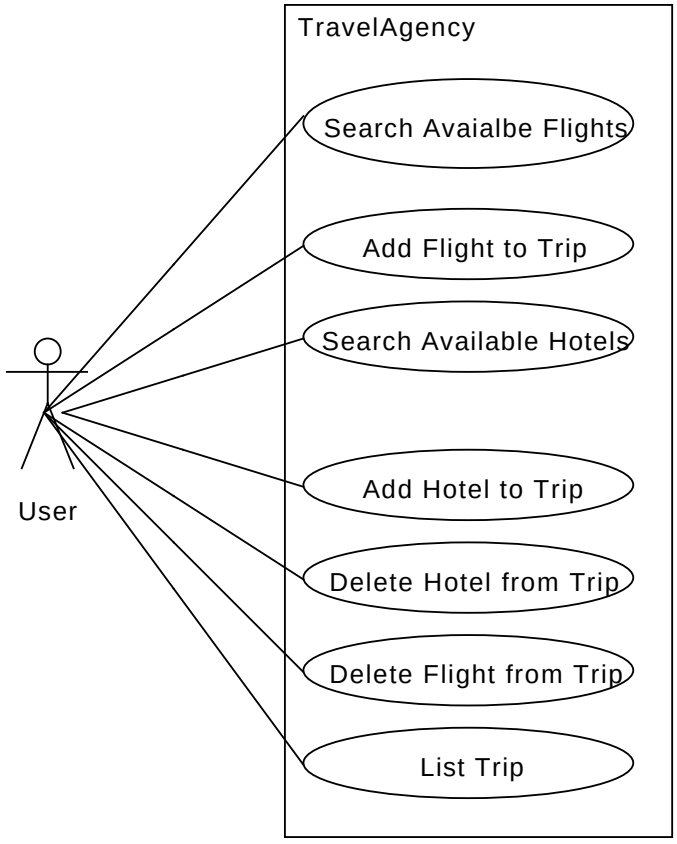
UML Destilled, Martin Fowler

Example Business Use Cases

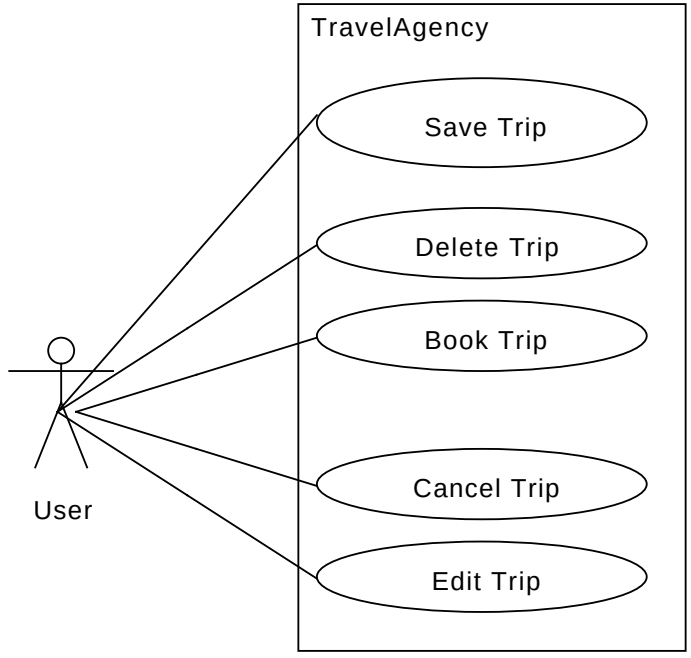


Example System Use Cases

Plan trip use cases



Manage trip use cases



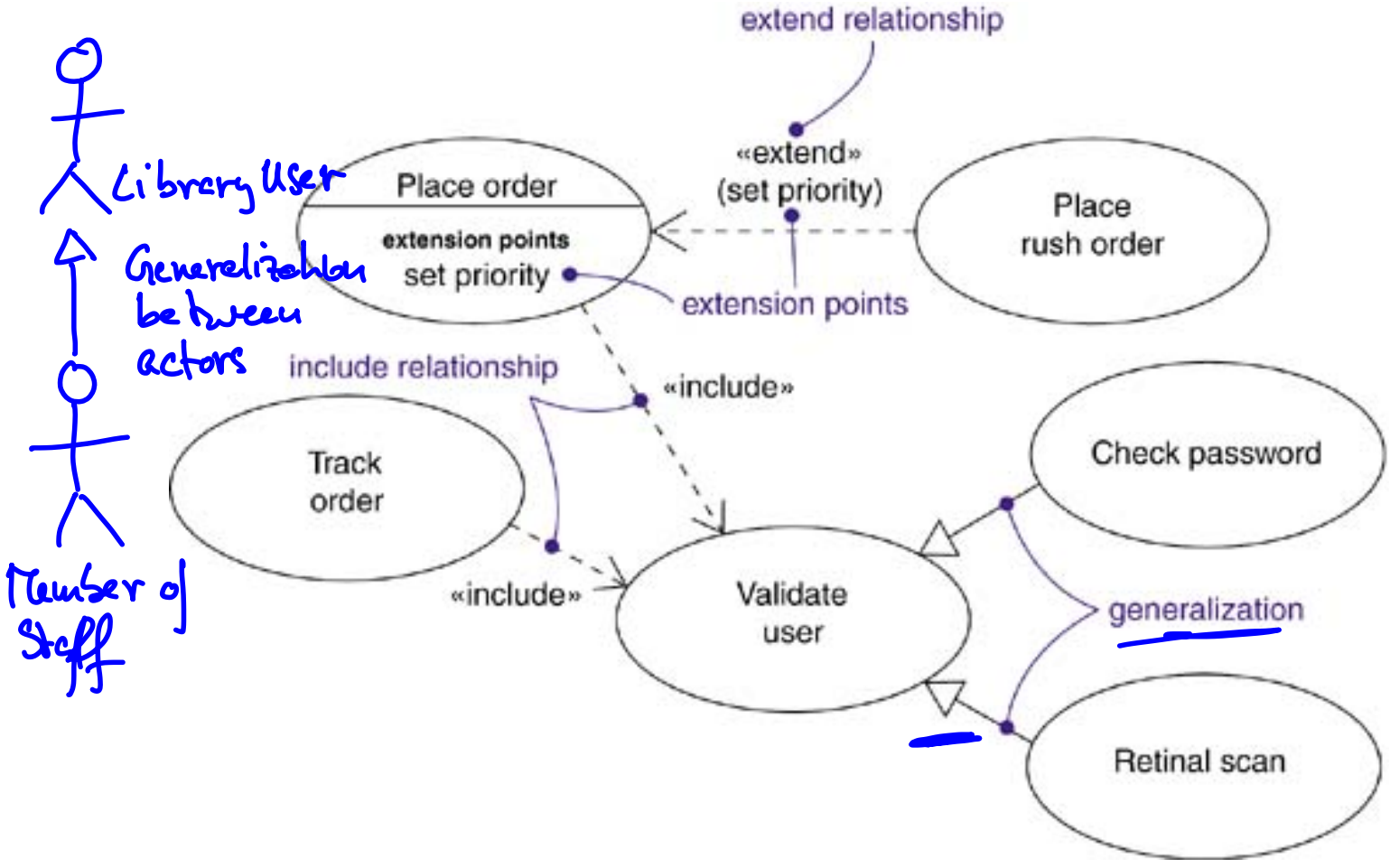
Business Use Cases vs. System Use Cases

- Choose the appropriate detail level and stick with it
- Start with business use cases
  - Overview over system functionality is more important than detail
- Add system level use cases as needed
- If needed use several diagrams

Relations between use cases

extends: used to extract variant behaviour

includes: used to factor common behaviour of use cases

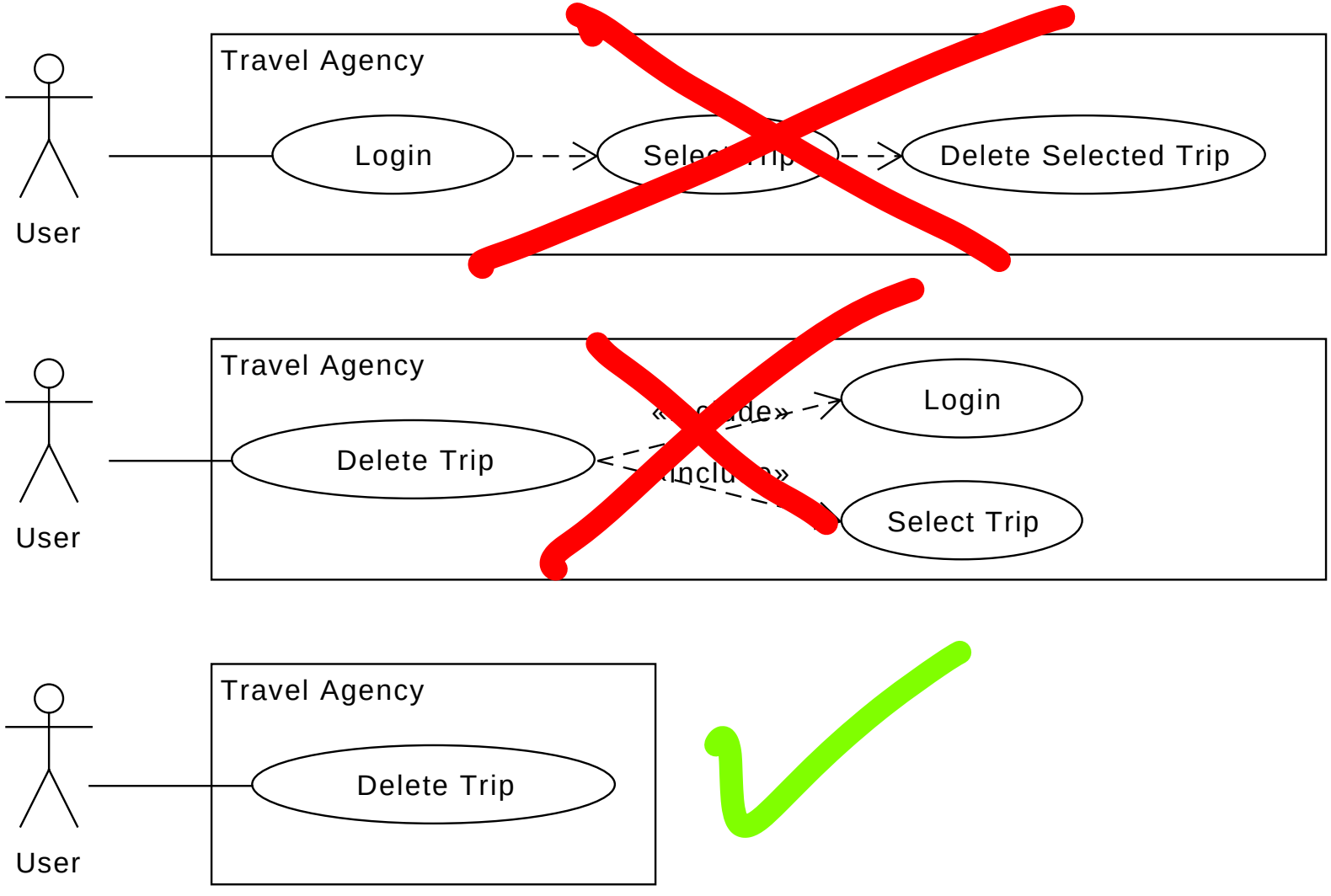


UML User Guide, Grady Booch et al

Use extend and include sparingly

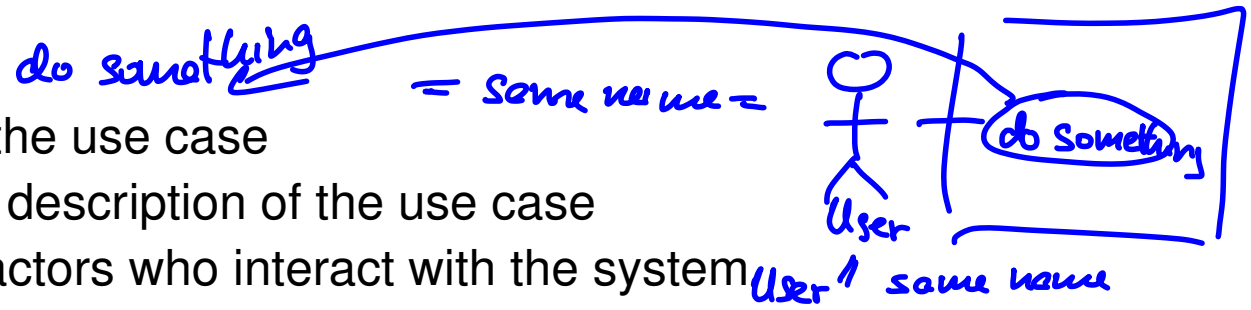
Don’ts of Use case diagrams

- Use case diagrams don’t explain how a use case works, this is part of the detailed use case description



Detailed use cases: Template

Template *to be used in this course* for detailed use case descriptions



name: The name of the use case

description: A short description of the use case

actor: One or more actors who interact with the system

precondition: Possible assumptions on the system state to enable the use case (optional)

happy scenario

main scenario: A description of the main interaction between user and system

→ Note: should *only* explain what the system does from the *user's* perspective

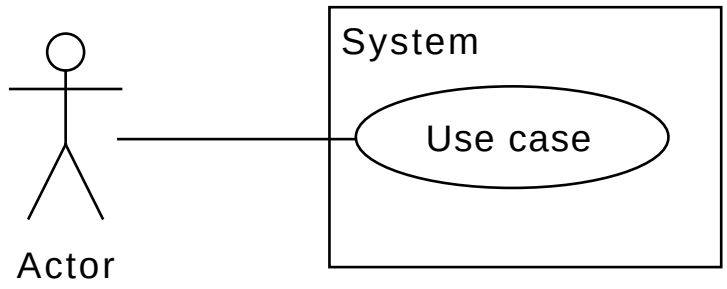
something went wrong (exceptional / alternative scenarios / fail scenarios)

alternative scenarios:

note: Used for everything that does not fit in the above categories

→ To be used in the examination report

Use case scenarios



- Use case scenarios = interaction between an actor and the system
  - Anything the actor does with the system
  - System responses
  - Effects visible/important to the customer
- Not part of the interaction: What the system internally does

not part of interaction

Detailed use case *search available flights*

name: search available flights

description: the user checks for available flights

actor: user

no GUI/UI

main scenario:

1. The user provides information about the city to travel to and the arrival and departure dates
2. The system provides a list of available flights with prices and booking number

alternative scenario:

1a. The input data is not correct (see below)

   2. The system notifies the user of that fact and terminates and starts the use case from the beginning

2a. There are no flights matching the users data

   3. The use case starts from the beginning

note: The input data is correct, if the city exists (e.g. is correctly spelled), the arrival date and the departure date are both dates, the arrival date is before the departure date, arrival date is 2 days in the future, and the departure date is not more then one year in the future

Detailed use case *cancel trip*

name: cancel trip

description: cancels a trip that was booked

actor: user

precondition:

- the trip must have been booked
- the first date for a hotel or flight booking must be one day in the future

main scenario:

1. user selects trip for cancellation
2. the system shows how much it will cost to cancel the trip
3. selected trip will be cancelled after a confirmation

alt. ~~sen~~ scenario

1| trip is not booked
Show an error message

Pre-condition

name: Delete trip

actor: User

scenario

1. login user
2. select trip
3. delete selected trip

Pre-condition

name: Delete trip

actor: User

scenario

1. login user
2. select trip
3. delete selected trip

- Issue: The user has to login each time

Pre-condition

name: Delete trip

actor: User

scenario

1. login user
2. select trip
3. delete selected trip

- Issue: The user has to login each time

name: Delete trip

actor: User

precondition: user is logged in

scenario

1. ~~login user~~
2. select trip
3. delete selected trip

- Now has be logged in, but does not have to login each time

Contents

Summary

- Requirements:
  - functional vs non-functional; user– vs system requirements
- Requirements process:
  - discover, document, validate
- Discovering requirements
  - Create a domain model: Glossary + Business Process
- Discovering and documenting requirements
  - Use cases: Use case diagram + detailed use cases
- Next week
  - User stories
  - Changing requirements