Software Engineering I (02161)

Week 4

Assoc. Prof. Hubert Baumeister

DTU Compute
Technical University of Denmark

Spring 2018

Contents

Basic Activities in Software Development

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

Requirements Analysis

Requirements Analysis

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

- *external behaviour* and *not* how it is realised
- Techniques used
  - Interviews
  - Business Processes
  - Domain model
  - Use Cases / User Stories: document functionality

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

    $\rightarrow$ more detailed than user requirements

    $\rightarrow$ 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

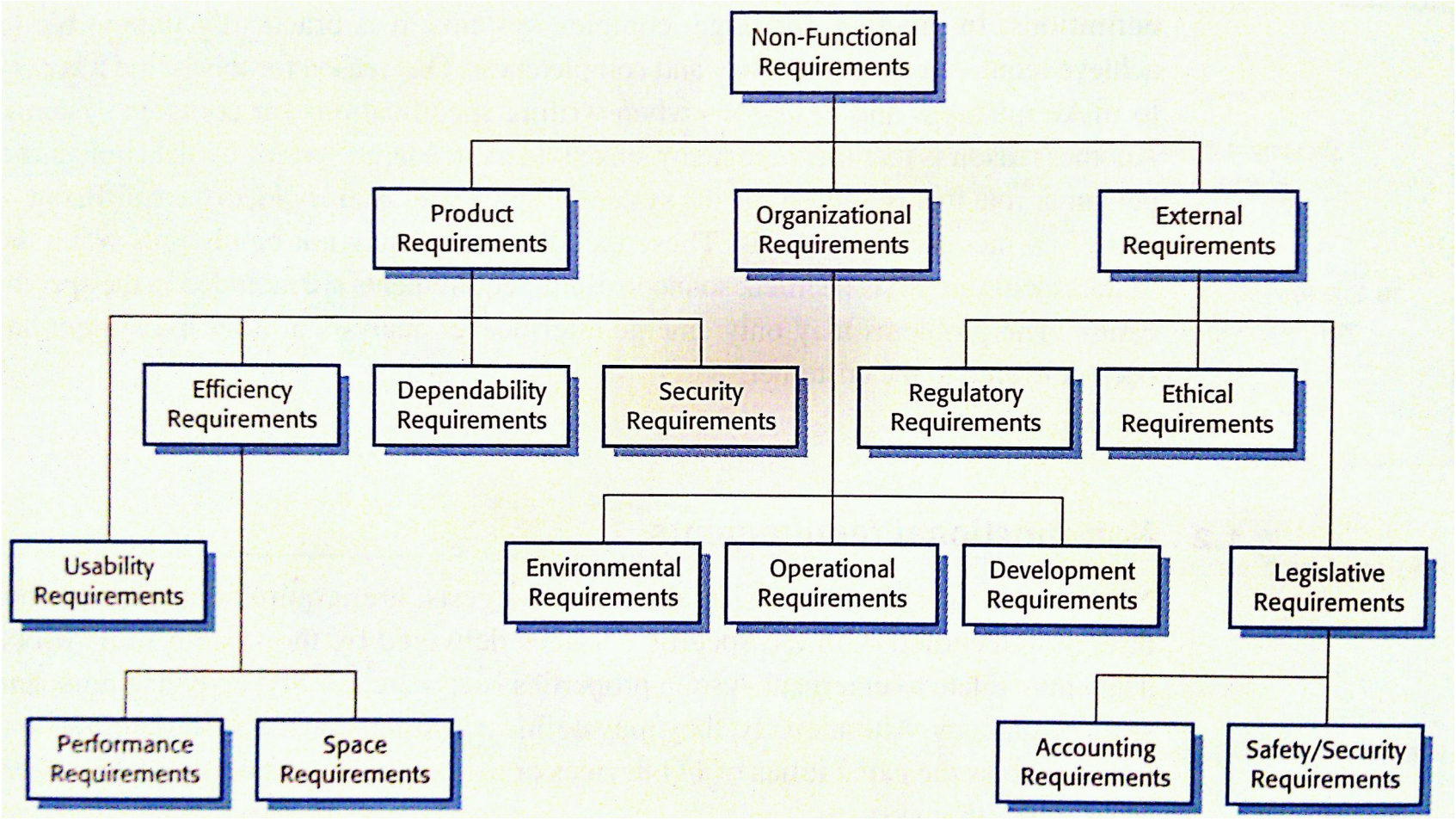
  "TravelGood wants to offer a trip-planning and booking application to its customers. The application should allow the customer to plan trips consisting of flights and hotels. First the customer should be able to assemble the trip, before he then books all the flights and hotels in on step. The user should be able to plan several trips. Furthermore it should be possible to cancel already booked trips. The application should be a user friendly Web application and should use the latest Java technology running on WildFly 10"

→ 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 WildFly 10”
  - ”It should use Java 9”
  - ”The system should be user friendly”

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

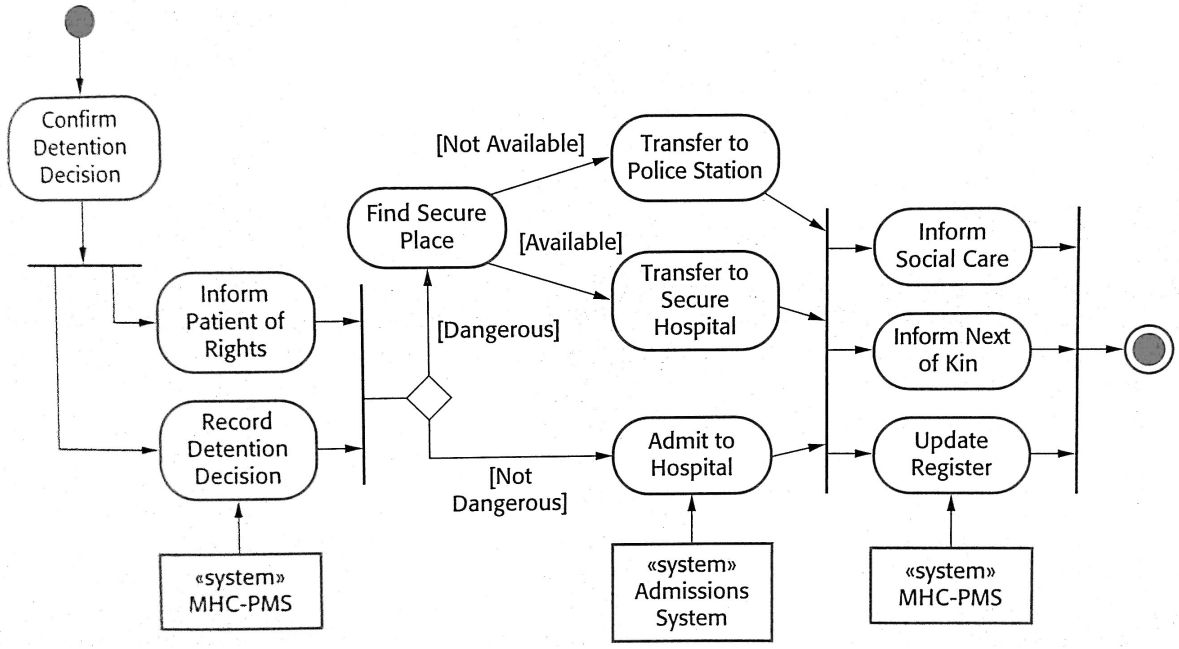
- Testability
  - → manual/automatic acceptance tests
- Measurable
  - Not measurable: *The system should be easy to use by medical staff and should be organised in such a way that user errors are minimised*
  - Measurable: *Medical staff shall be able to use all the system functions after four hours 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

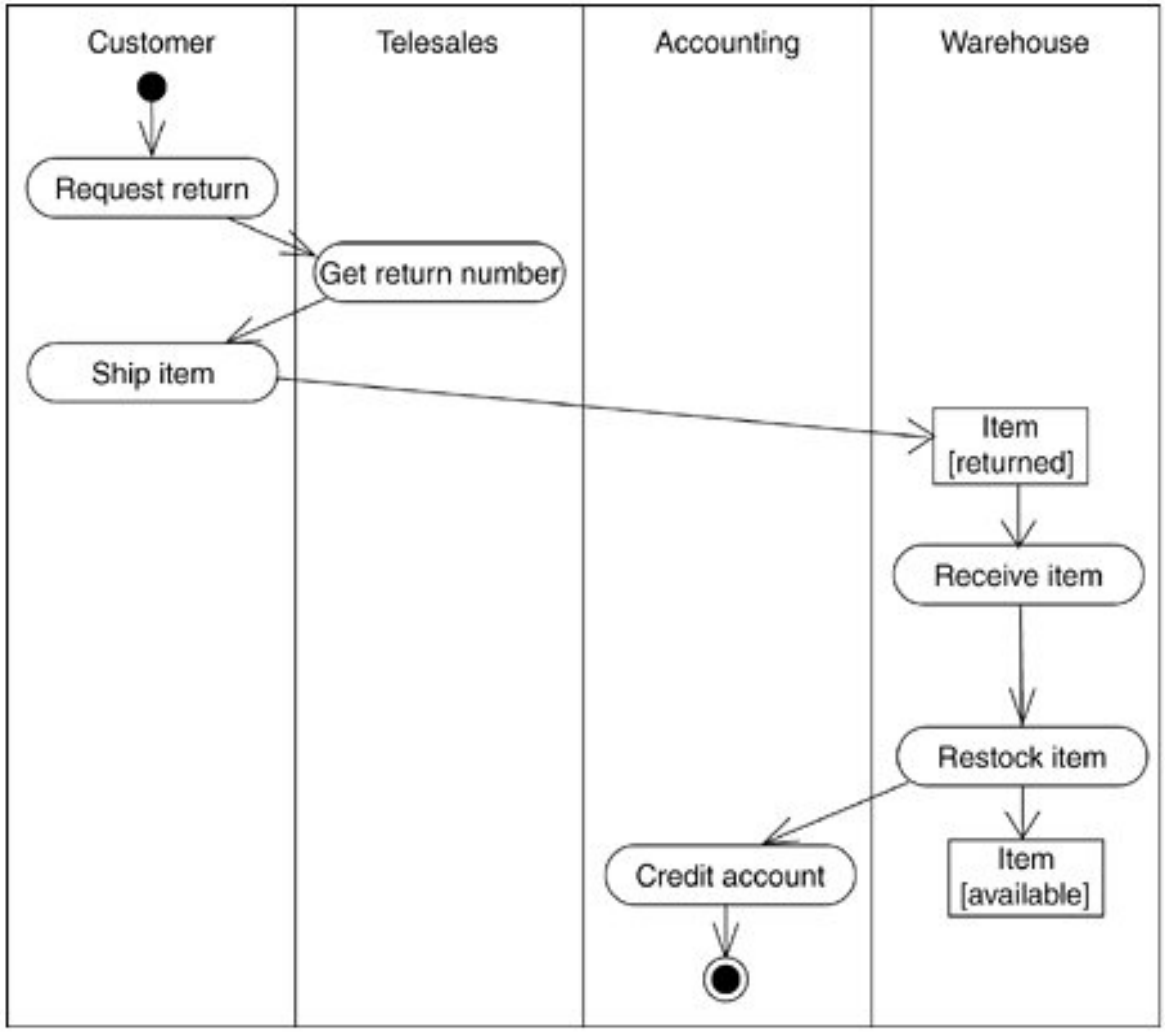
Contents

Activity Diagram: Business Processes

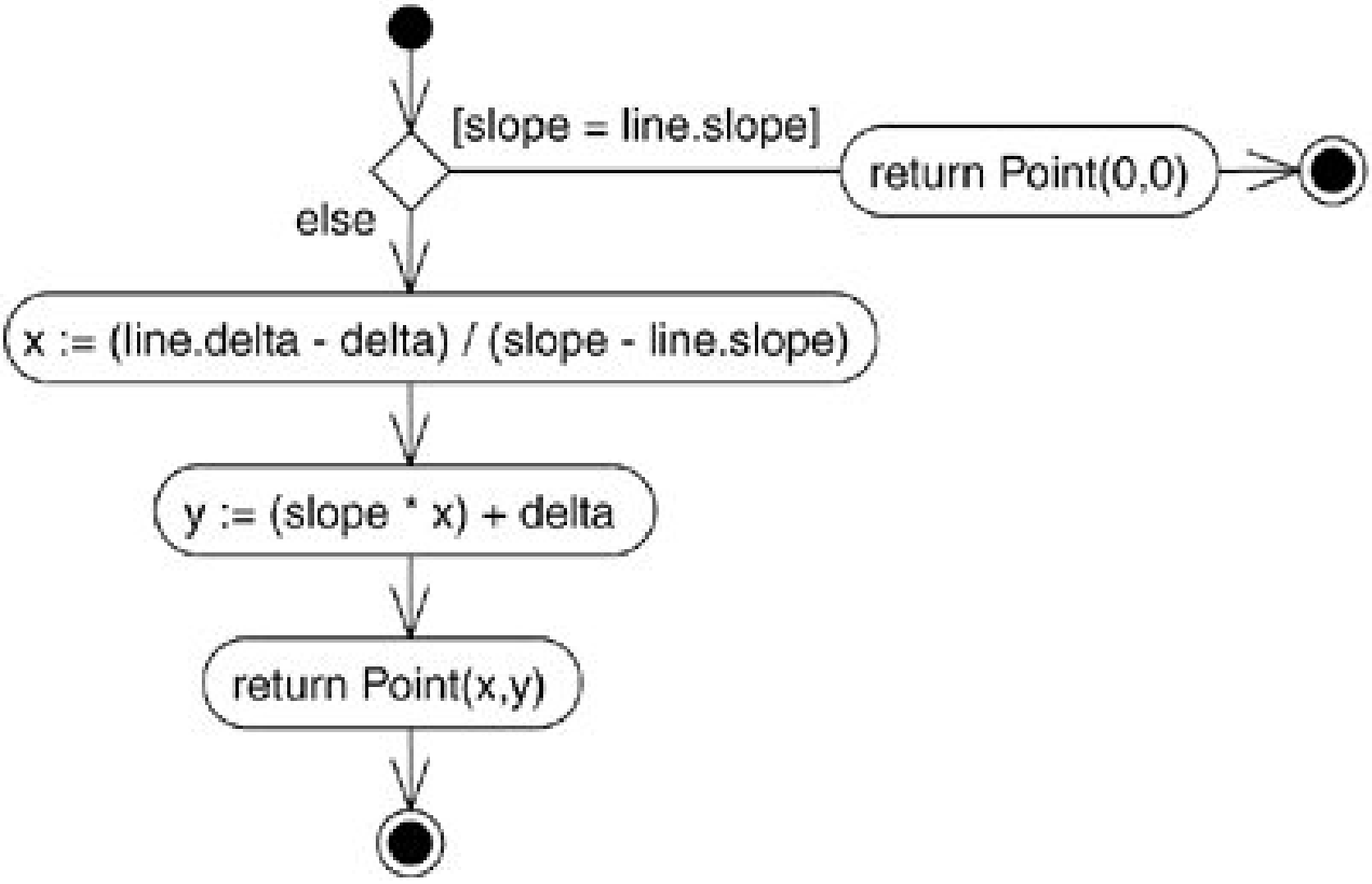


- Describe the *context* of the system
- Helps find the requirements of a system
  - What does the user do?
  - How does he interact with what kind of systems?
  - Ideally, software systems need to fit in into existing business processes

Activity Diagram Example Workflow



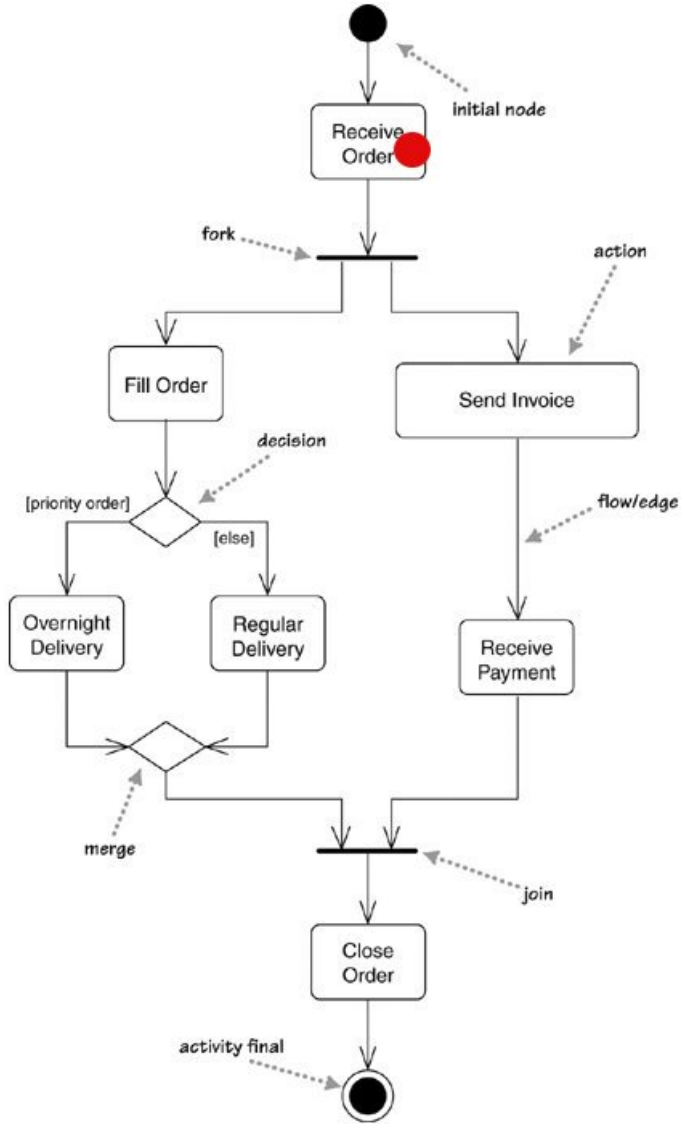
Activity Diagram Example Operation



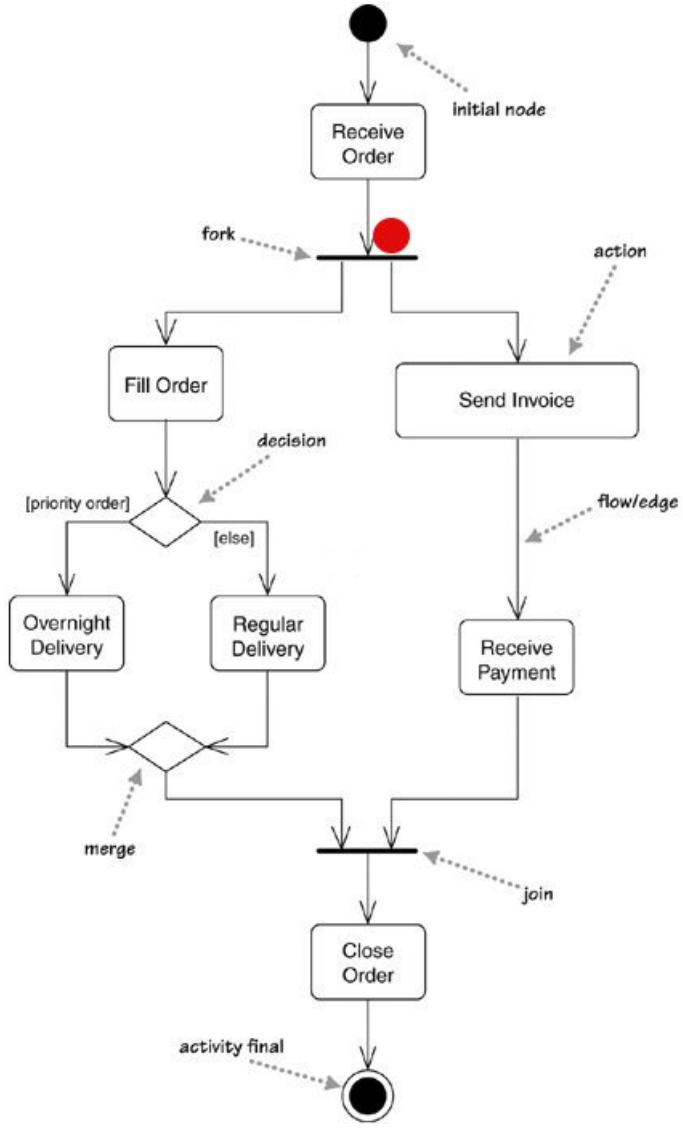
UML Activity Diagrams

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

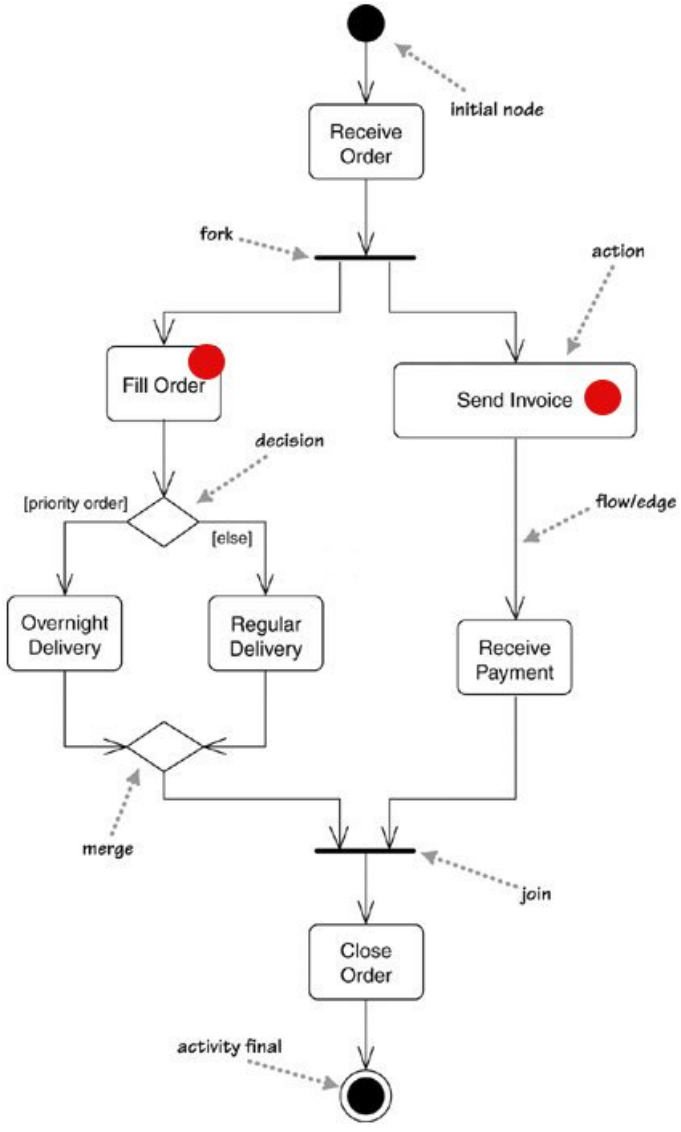
Activity Diagrams Execution



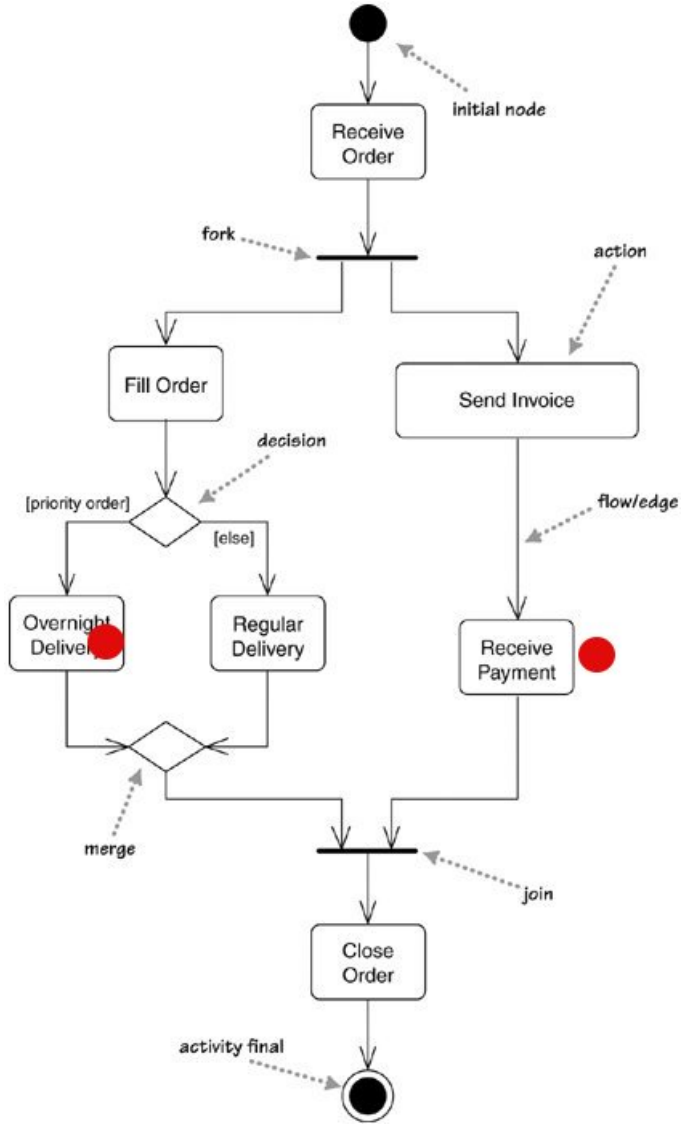
Activity Diagrams Execution



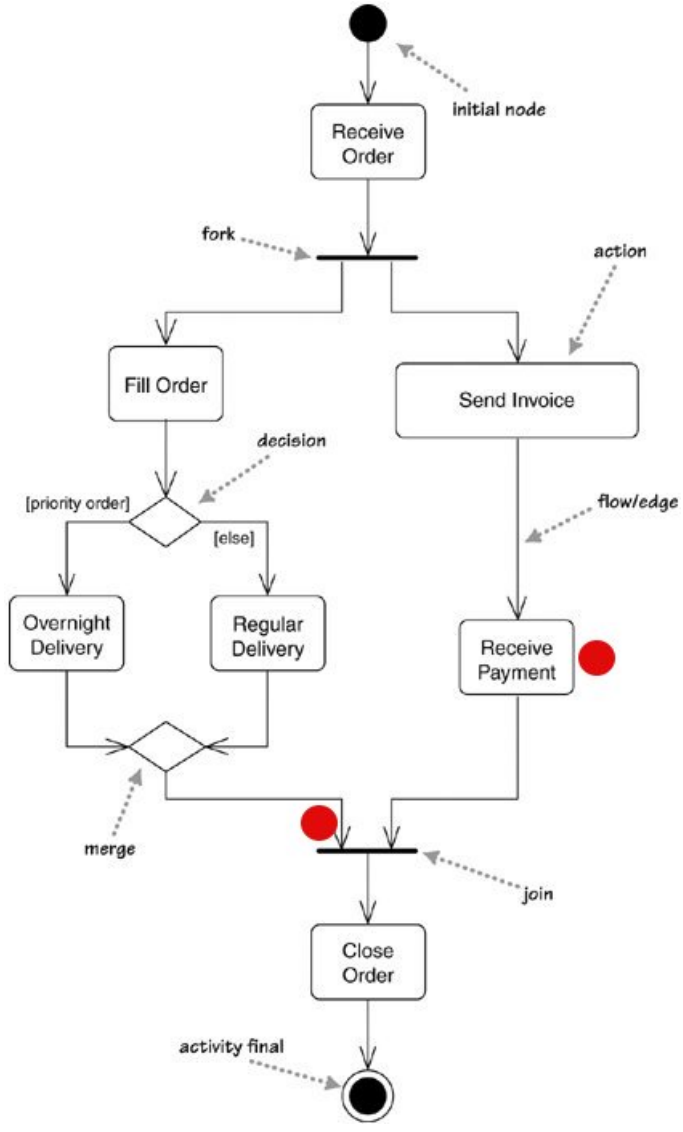
Activity Diagrams Execution



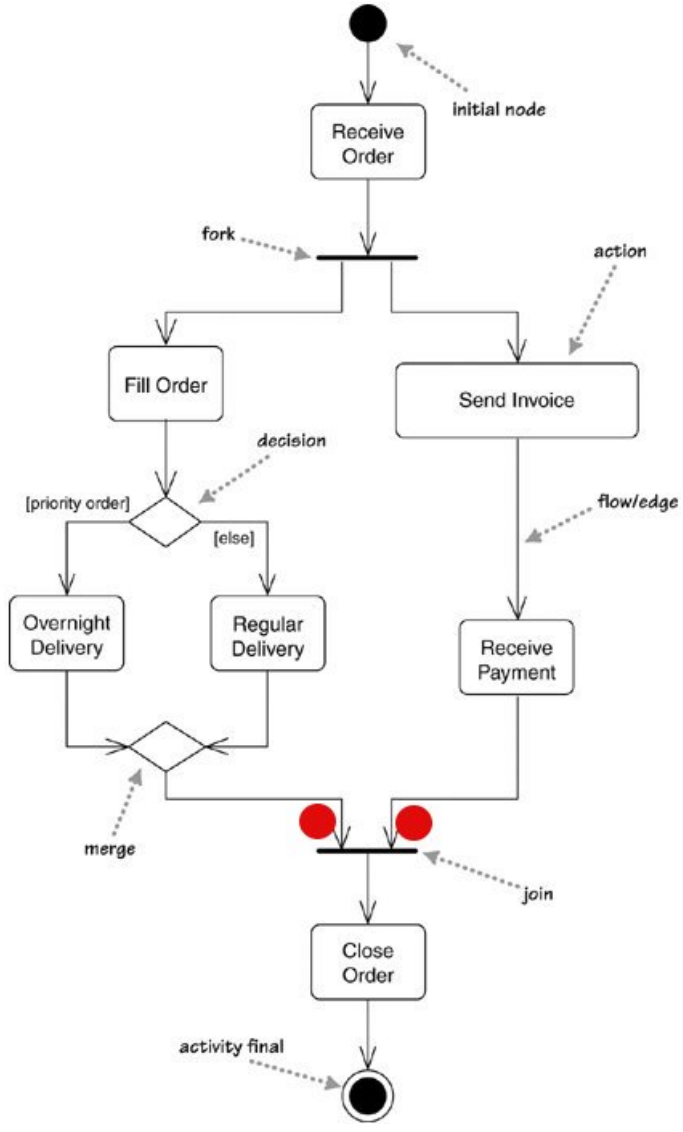
Activity Diagrams Execution



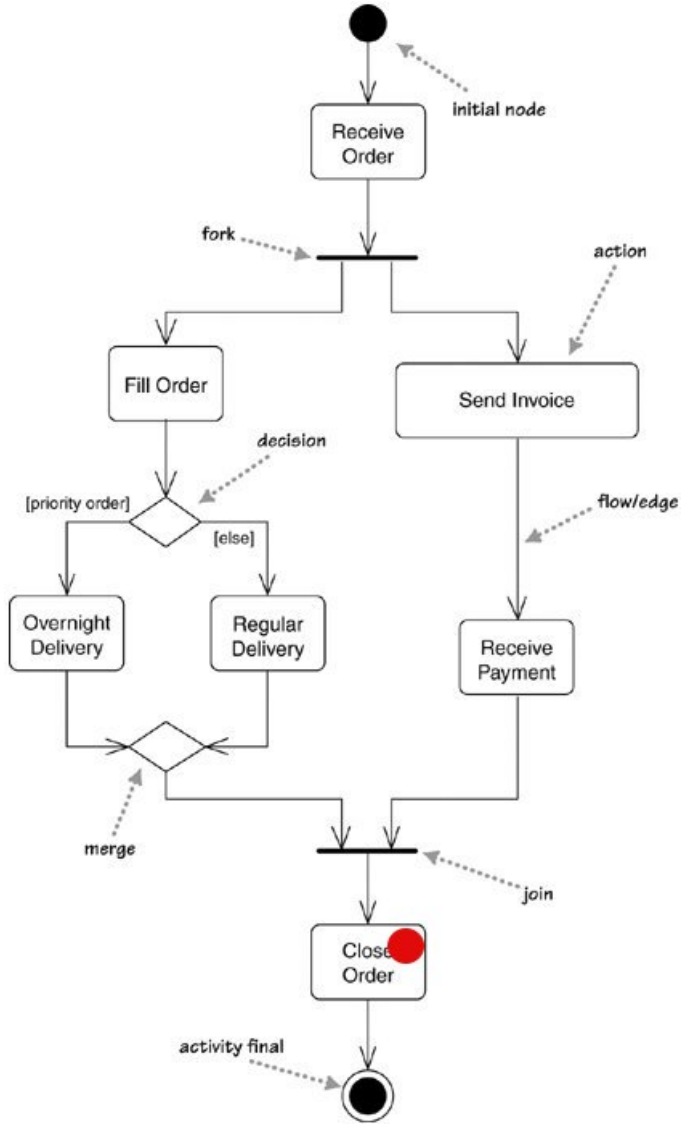
Activity Diagrams Execution



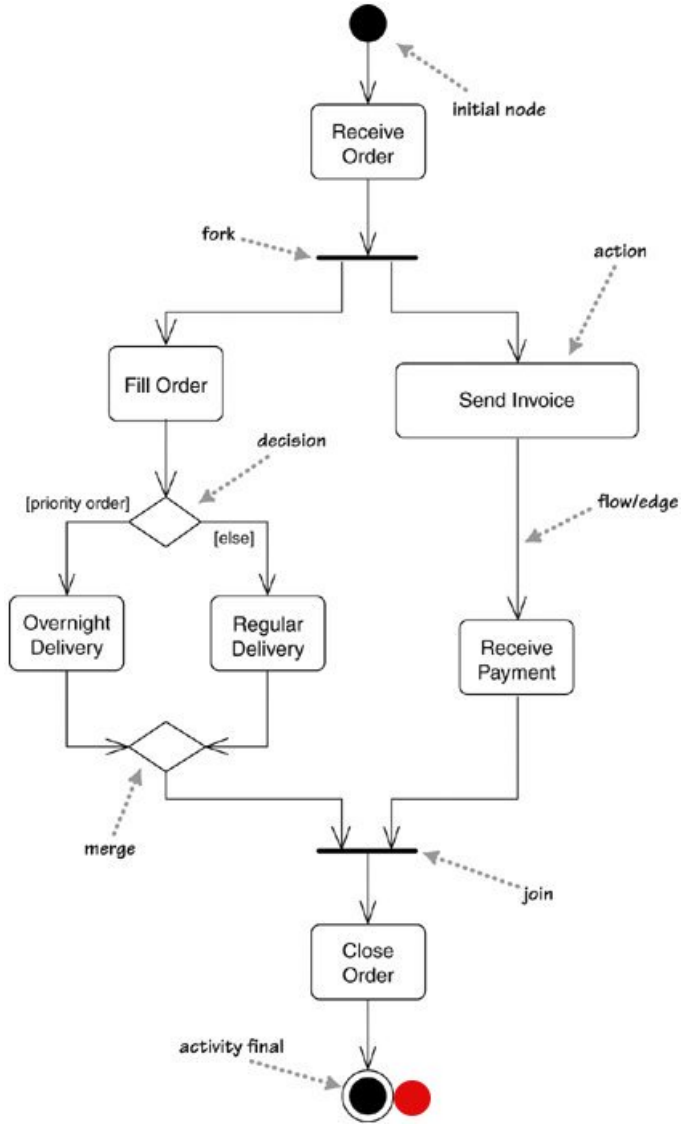
Activity Diagrams Execution



Activity Diagrams Execution

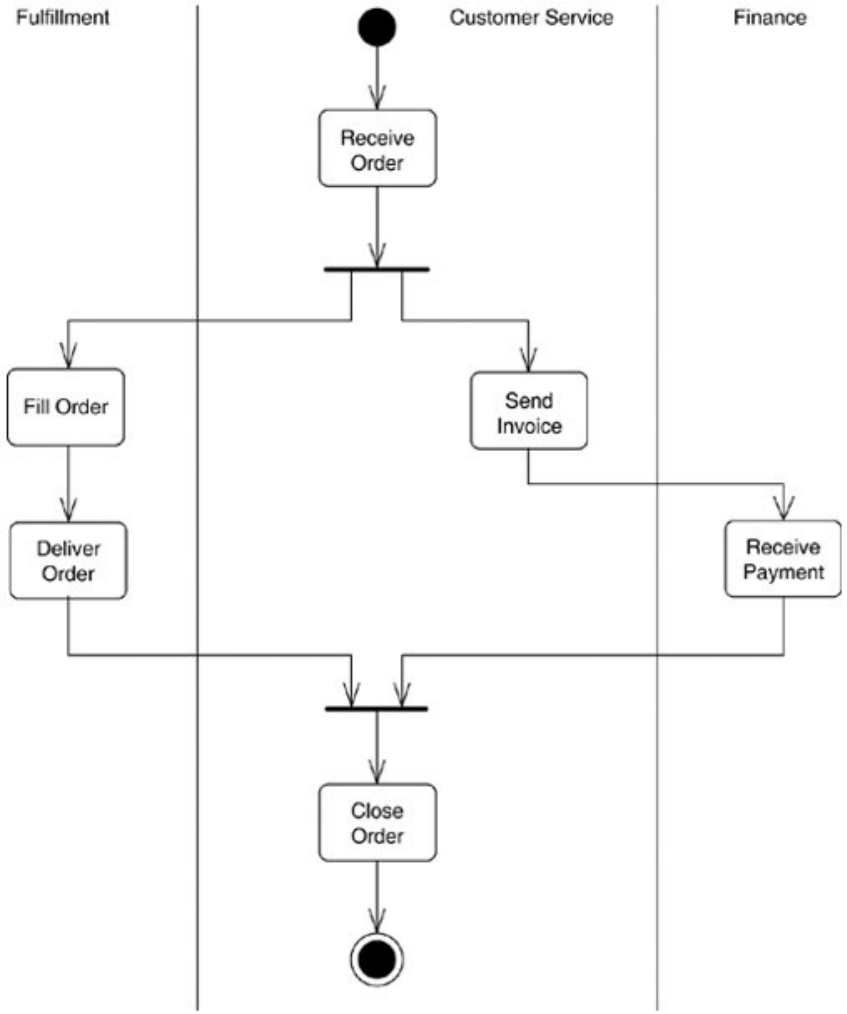


Activity Diagrams Execution

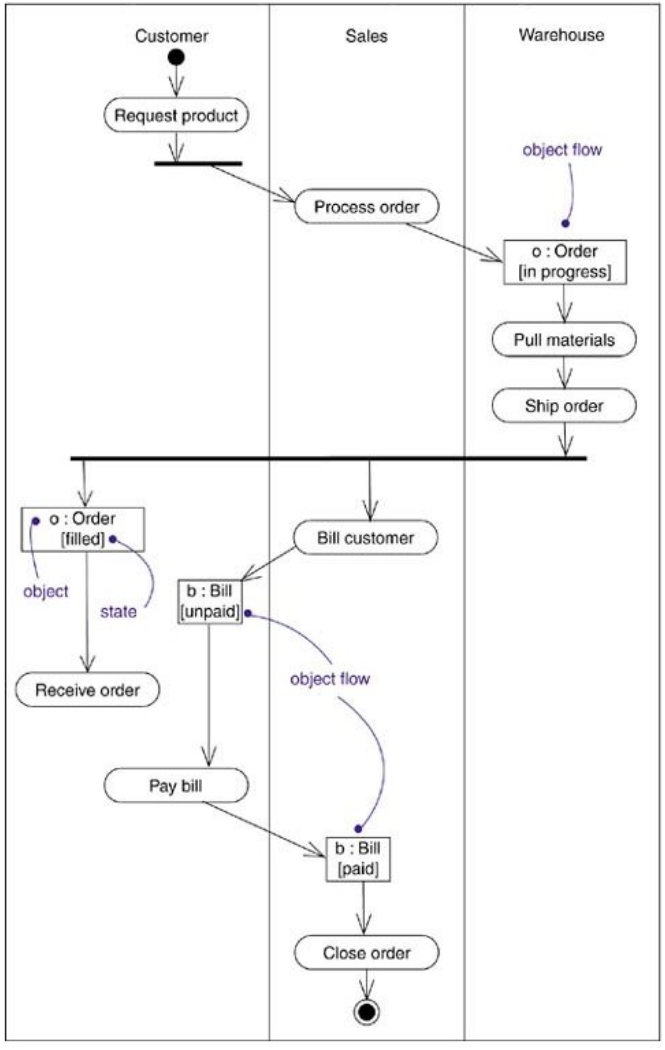


Swimlanes / Partitions

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

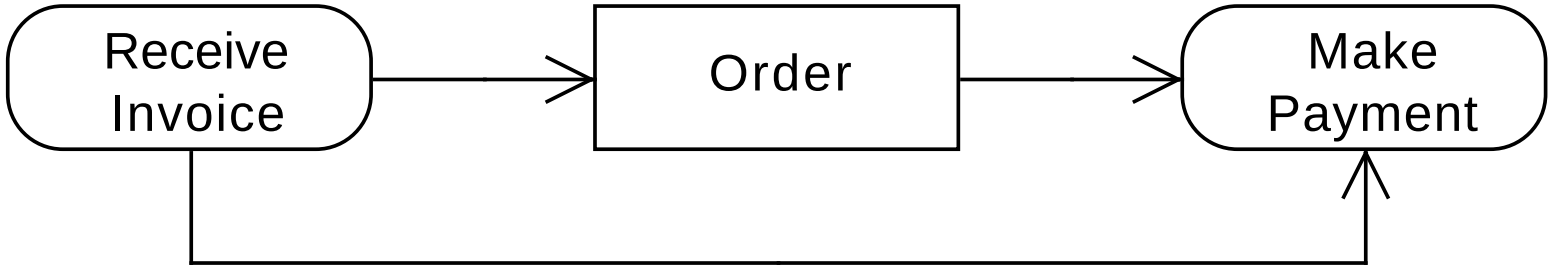


Objectflow example



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 of activities
- 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

Domain model

- Capture *customer's knowledge* for *system builders*
  - → *ubiquitous language* = speak the *same language*
- Define the **terminology**
  - → *Glossary*
- Define **Relationships** between concepts
  - → Class diagram
  - → Related to *knowledge representation* and *ontology*
- Make *business processes* visible

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.
- *Warning*
  - Don’t try to capture all possible knowledge
  - But don’t forget important concepts
- Glossaries develop over time

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

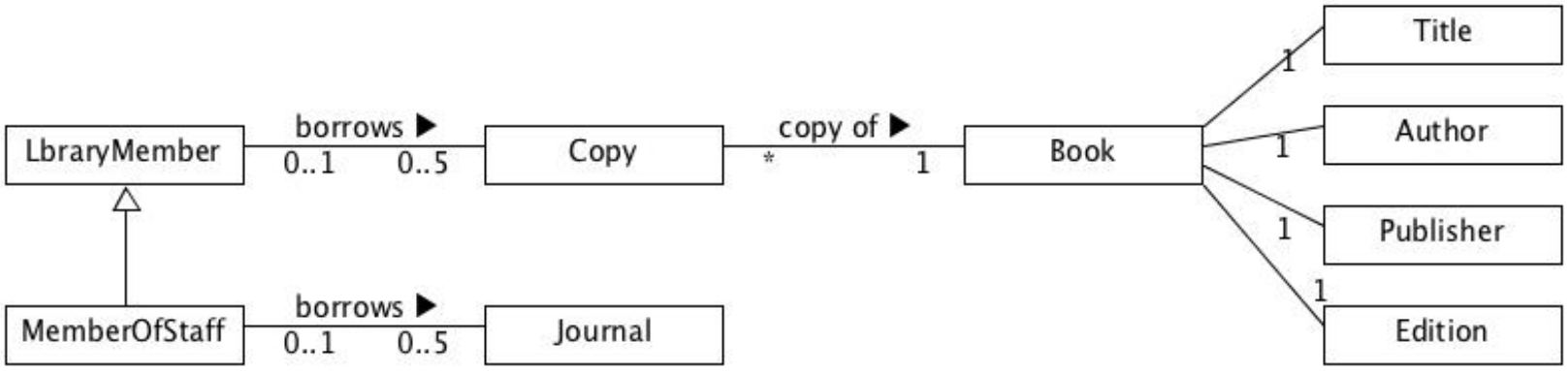
- A copy is a physical copy of a particular book. For example, the library has three copies of the book ”Using UML” by Perdiate Stevens. . . .

. . .

Terms and their relations

- Class diagrams
- Usually
  - Classes (for nouns)
  - Associations: for static relationships
  - Generalizations
  - Avoid use of *attributes*
  - Avoid use of *operations*

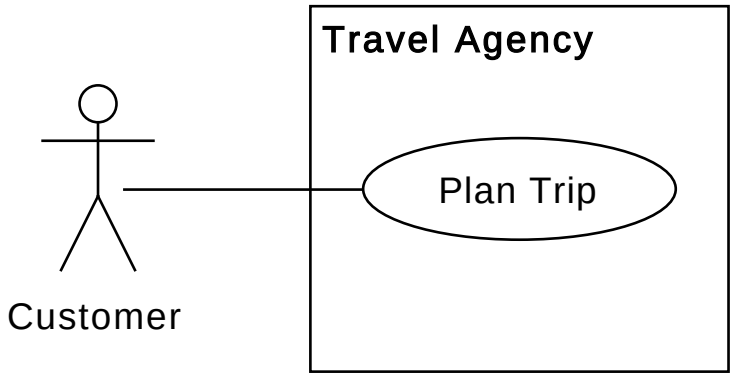
Domain model (terms and their relations)



Contents

Definition Use Case

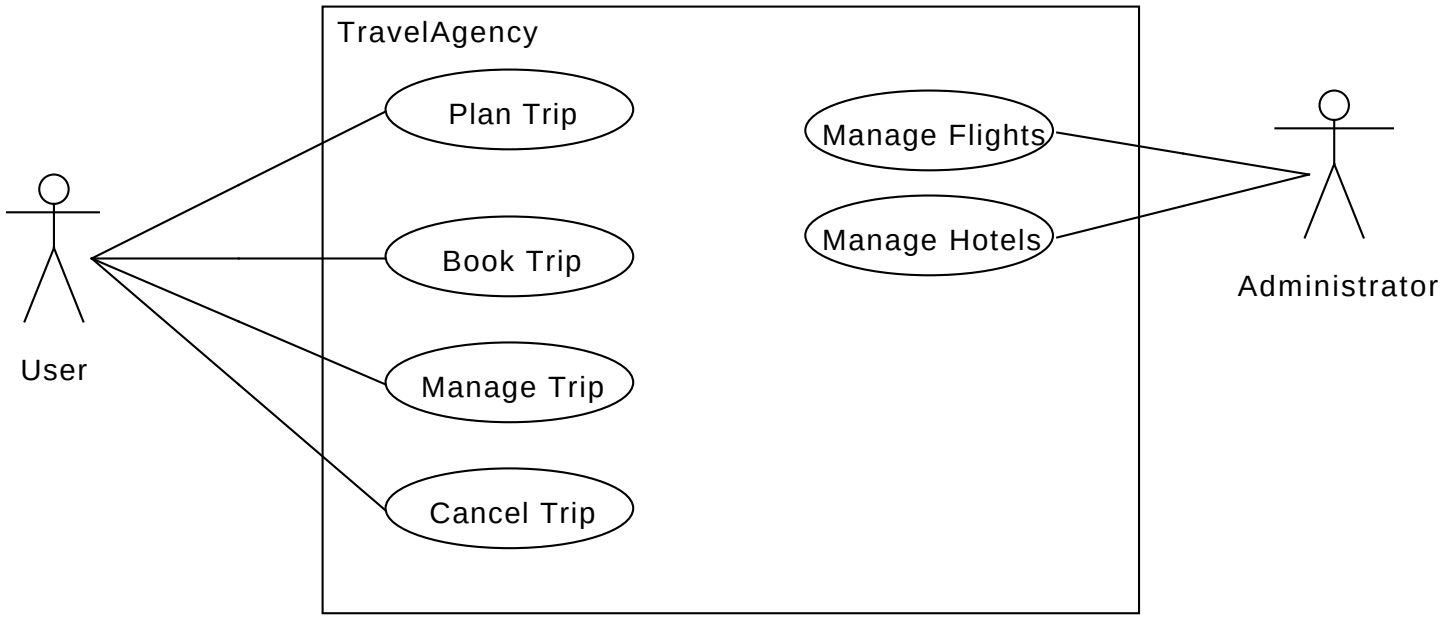
- Use cases discover and document functional requirements
  - → Naming convention: ”Do something” (= functionality)



- Set of interactions: Actor $\leftrightarrow$ System
  - Anything the *actor* does with the system
  - System responses
  - Effects visible/important to the customer
  - Common goal
- Two ways of documenting
  - Use case diagram
  - Detailed use case

Use Case Diagram

Purpose: Overview

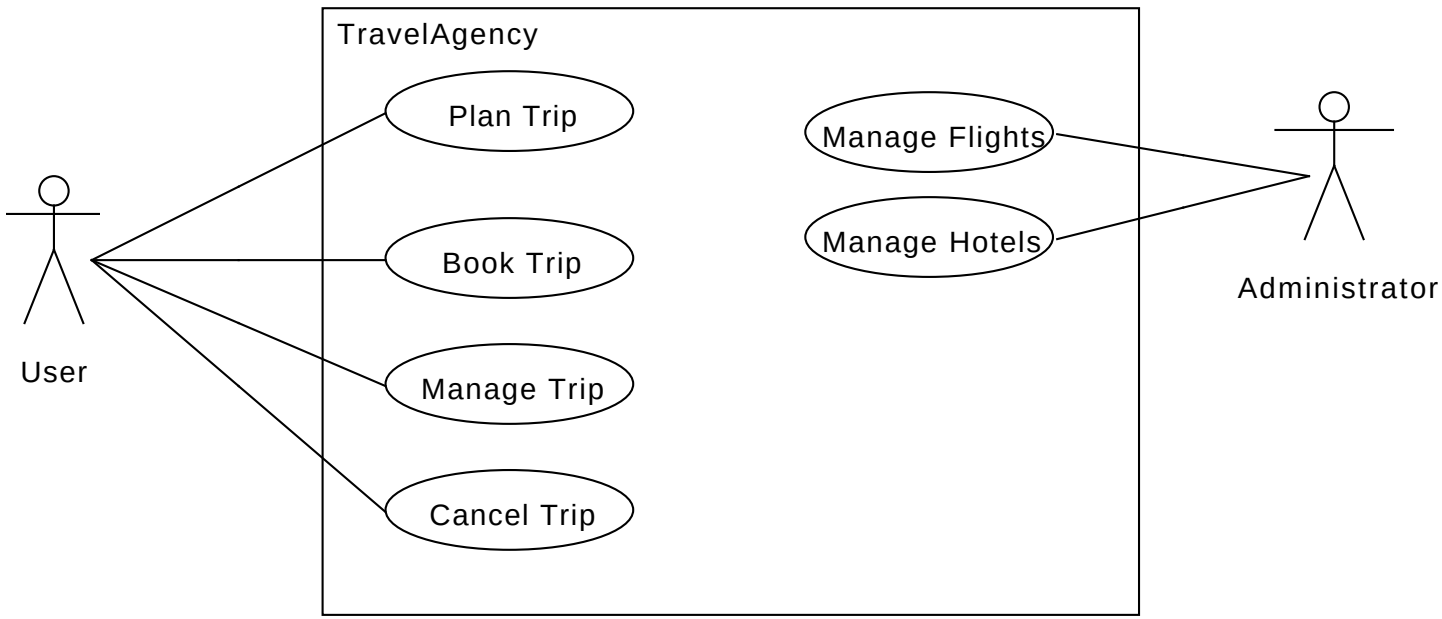


Levels of use case diagrams

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)

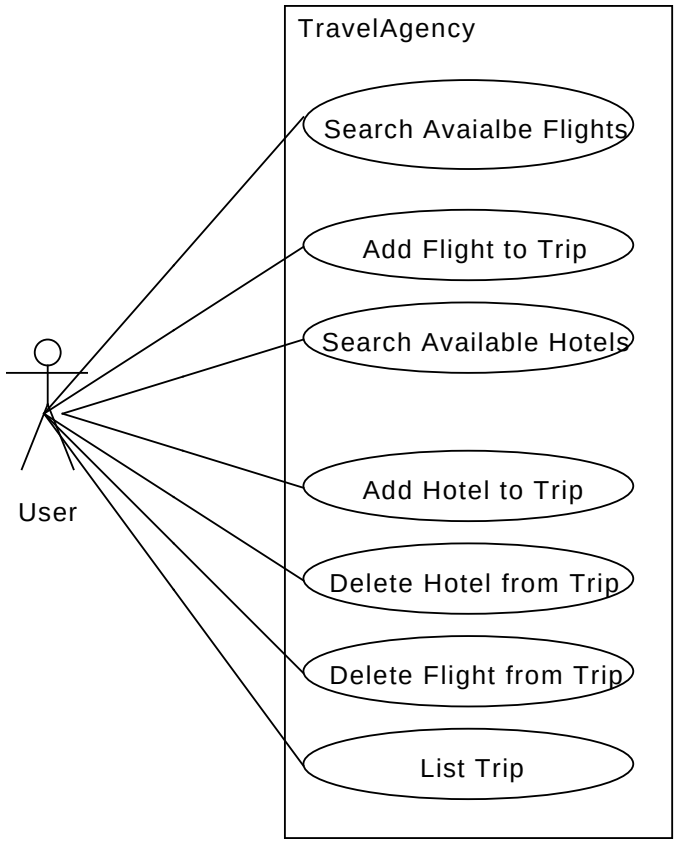
UML Destilled, Martin Fowler

Example Business Use Cases

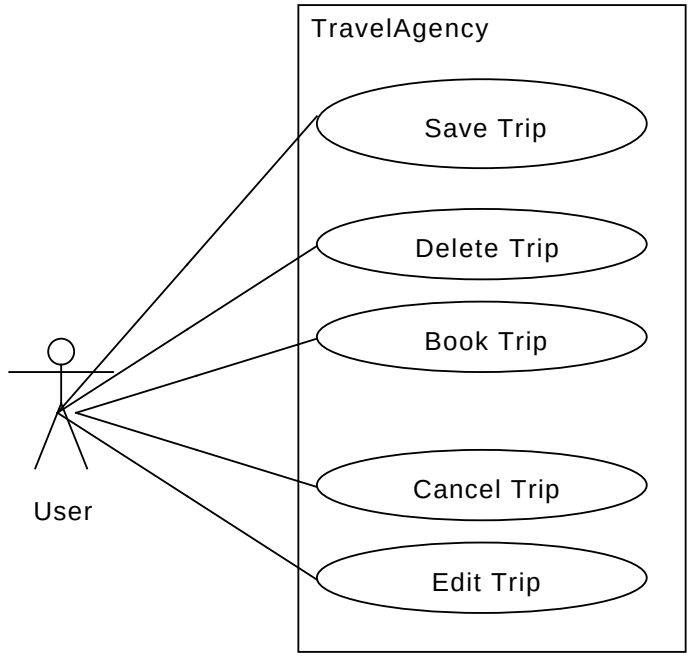


Example System Use Cases

Plan trip use cases



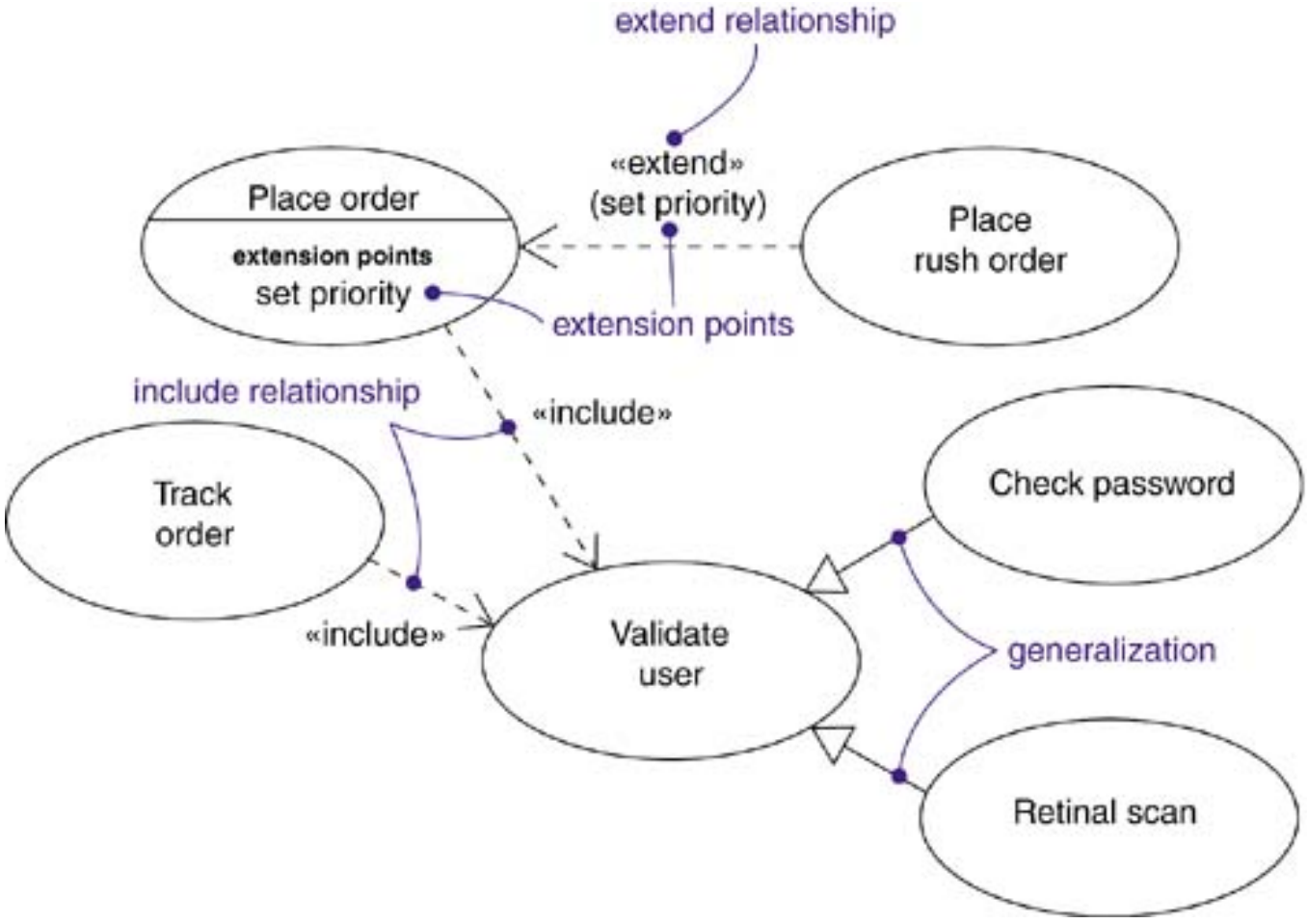
Manage trip use cases



Relations between use cases

extends: used to extract variant behaviour

includes: used to factor common behaviour of use cases

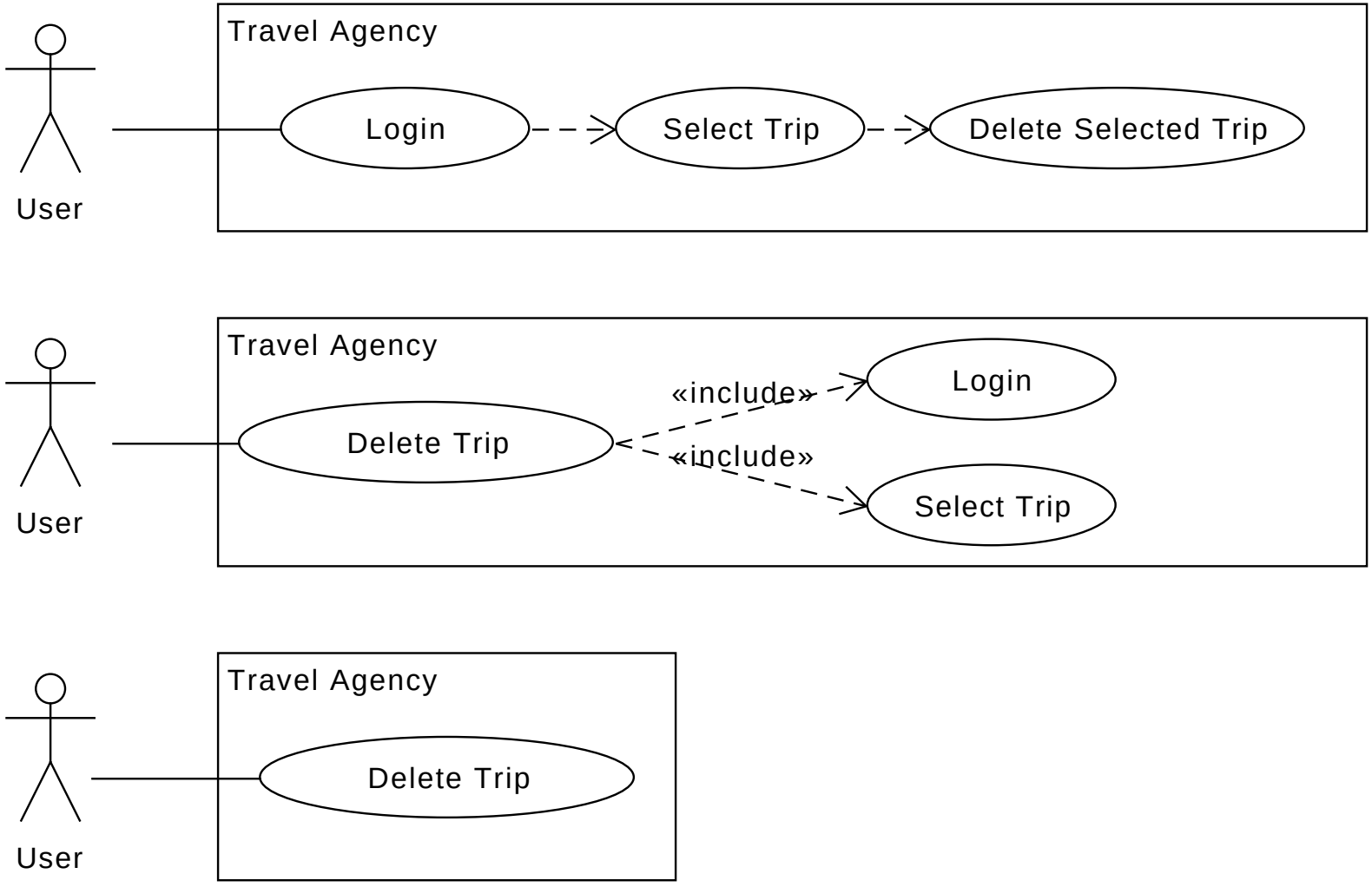


UML User Guide, Grady Booch et al

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 Case: *search available flights*

name: search available flights

description: the user checks for available flights

actor: user

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: Cucumber feature

```
Feature: <Name of the use case/feature>
   Description: <Short description>
   Actor: <One or more actors>

# Main scenario
Scenario: <Name>
   Given <establishing a starting state>
   And <possible precondition> # optional
   When <an action occurs>
   Then <then there is a resulting state>
   And <possible postcondition> # optional

# Alternative scenarios
Scenario: <Name>
...
```

Detailed use case *search available flights*

```
Feature: Search available flights one way
   Description: the user checks for available flights one way
   Actors: User

# Main scenario
Scenario: Search for available flight
   Given the user provides information about the city
         the user starts his travel
   And about the city the user travels to
   And about the departure date
   When the user searches for flights
   Then the system provides a list of available flights
        with prices and booking number

# Fail scenario one
Scenario: No flights when the departure date
           has already past
   Given the user provides information about the city
         the user starts his travel
   And about the city the user travels to
   And about the departure date, which is in the past
   When the user searches for flights
   Then the system provides an error message that
        the departure date needs to be in the future

# More scenarios for other error conditions
```

Step definitions

```
@Given("^the user provides information about the city the user starts
public void theUserProvidesInformationAboutTheCityTheUserStartsHisTrav
   from = "Copenhagen";
}

@Given("^about the city the user travels to$")
public void aboutTheCityTheUserTravelsTo() throws Exception {
   to = "Paris";
}

@Given("^about the departure date$")
public void aboutTheDepartureDate() throws Exception {
   departureDate = new GregorianCalendar(2018,1,10);
}

@When("^the user searches for flights$")
public void theUserSearchesForFlights() throws Exception {
   result = travelAgency.searchFlights(from,to,departureDate);
}

@Then("^the system provides a list of available flights with prices an
public void theSystemProvidesAListOfAvailableFlightsWithPricesAndBooki
   assertEquals(3,result.size());
   /* Check that the correct flight information is returned. */
}
```

Alternate use of Cucumber

```
Feature: Search available flights
   Description: the user checks for available flights
   Actors: User

# Main scenario
Scenario: Search for available flights
   Given the user wants to fly from "Copenhagen"
         to "Paris" on "10.2.2018"
   When the user searches for flights
   Then these flights are returned
    | CPH | CDG | 10.2.2018 | LH 1023  | 8:30  | 1:30 | 1000 | BK01 |
    | CPH | CDG | 10.2.2018 | AF 4201  | 9:00  | 1:30 | 2000 | BK02 |
    | CPH | CDG | 10.2.2018 | SAS 0022 | 13:00 | 1:30 | 1500 | BK03 |

# Fail scenario one
Scenario: No flights when the departure date
          has already past
   Given that today is "1.2.2018"
   Given the user wants to fly from "Copenhagen"
         to "Paris" on "31.1.2018"
   When the user searches for flights
   Then the system provides the error message
        that the departure date needs to be in the future

# More scenarios for the other fail scenarios
```

Step definitions

```
@Given("^the user wants to fly from \"([^\"]*)\" to \"([^\"]*)\"
         on \"([^\"]*)\"$")
public void theUserWantsToFlyFromToOn(String from, String to,
                                      String date) throws Exception {
   this.from = from;
   this.to = to;
   this.date = convertDateFromString(date);
}

@When("^the user searches for flights$")
public void theUserSearchesForFlights() throws Exception {
   result = travelAgency.searchFlights(from,to,date);
}

@Then("^these flights are returned$")
public void theseFlightsAreReturned(List<Map<String,String>> flights)
   assertEquals(flights.size(),result.size());
   /* check that the values of result are correct reg. flights */
}
```

Detailed use case *cancel trip*

```
Feature: Cancel Trip
   Description: the user cancels a trip that was booked
   Actors: User

Scenario: cancel trip
   Given that the user has booked a trip
   And that the earliest date of a booking
       belonging to that trip has not yet passed
   When the user cancels the trip
   Then the system shows the user the cost of cancellation
   When the user confirms cancellation
   Then the trip is cancled

Scenario: started trip cannot be cancelled
   Given that the user has booked a trip
   And that the earliest date of a booking
       belonging to that trip has has passed
   When the user cancels the trip
   Then the system provides an error message
        that the trip can't be cancelled
        because it has already started
   And the trip is not cancled
```

Pre-condition

```
Feature: Delete trip
   Description: the user deletes a trip
   Actors: User
```

```
Scenario: The user deletes a trip
   Given that the user
         has planned a trip
   When the user logs-in
   And the user deletes the trip
   Then the trip is deleted
```

Pre-condition

```
Feature: Delete trip
   Description: the user deletes a trip
   Actors: User
```

```
Scenario: The user deletes a trip
   Given that the user
         has planned a trip
   When the user logs-in
   And the user deletes the trip
   Then the trip is deleted
```

- Issue: The user has to login each time

Pre-condition

```
Feature: Delete trip
   Description: the user deletes a trip
   Actors: User
```

```
Scenario: The user deletes a trip
   Given that the user
         has planned a trip
   When the user logs-in
   And the user deletes the trip
   Then the trip is deleted
```

- Issue: The user has to login each time

```
Scenario: The user deletes a trip
   Given that the user
         has planned a trip
   And that the user is logged-in
   When the user deletes the trip
   Then the trip is deleted
```

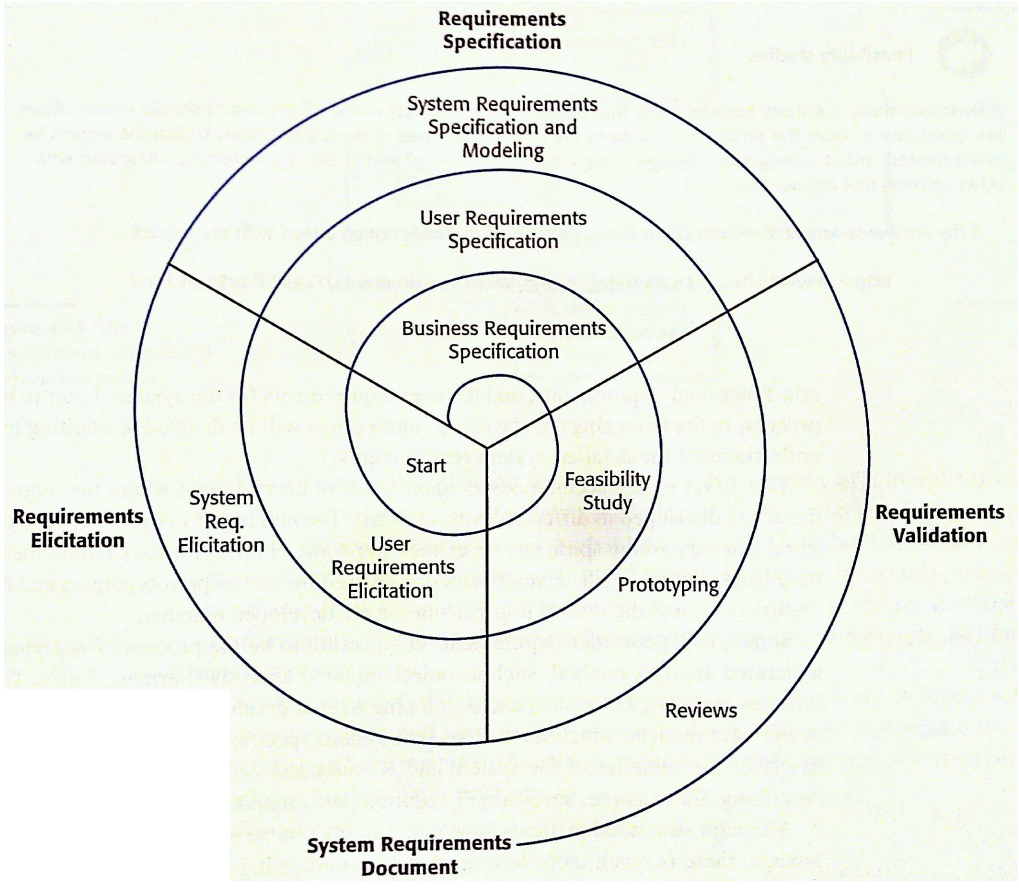
- Now the user has to be logged in, but does not have to login each time

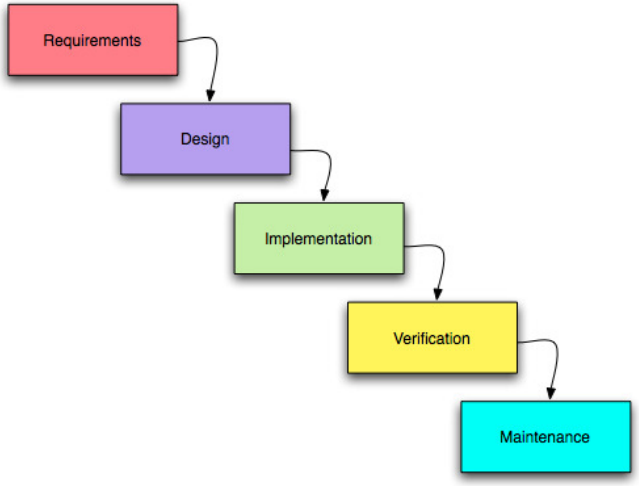
Contents

Requirements engineering process

p-type/e-type projects:

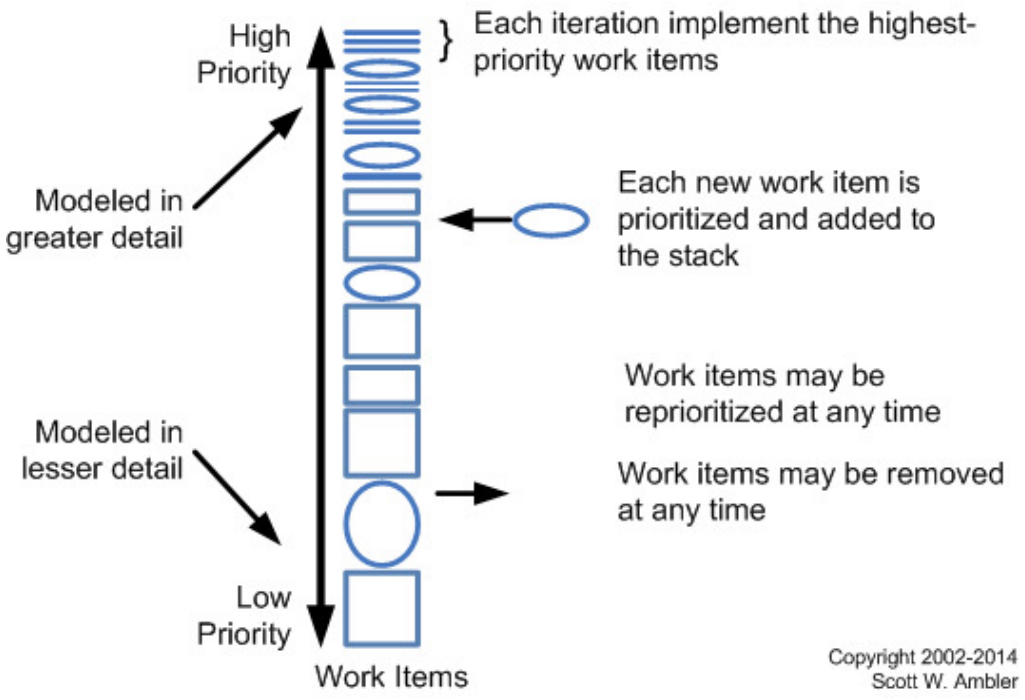
- impossible to get requirements right the first time





Requirements engineering process: Agile Methods

- Pareto principle 80/20
- Feedback: requirements change



Scott Ambler 2003–2014

`http://www.agilemodeling.com/artifacts/userStory.htm`

1. Business Processes
2. Domain model
3. Use case diagram
4. Take most important use case scenario
5. Detail use case / write Cucumber scenario
6. Implement
7. Feedback: Adapt requirements
8. Repeat from step 4

- Lookout for *Minimal Viable Product*