Software Engineering I (02161)

Week 2

Assoc. Prof. Hubert Baumeister

DTU Compute
Technical University of Denmark

Spring 2013

Contents

- Java
  - Collections
  - User-defined Exceptions
  - Delegation
- What are software requirements?
- Requirements Engineering Process
- Use Cases
- Glossary
- Summary

Lists (Collections)

- Interface: java.util.List<T>
  → `http://docs.oracle.com/javase/1.4.2/docs/api/java/util/List.html`
- Classes implementing the List interface:
  - java.util.ArrayList<T>, java.util.Vector<T> (among others)

→ Use java.util.List<T> in all methods and as the type of the instance variable

→ Information hiding

- decoupling implementation from usage

```
List<Book> books = new ArrayList<Book>()
```

Creating a List

generic type
argument

- Instance variable containing a list

```
List<Book> books = new ArrayList<Book>();
```

- Alternative (not so good)

```
ArrayList<Book> books = new ArrayList<Book>();
```

Iterating over a list

Maybe 1?

<= ?

- Variant a)

```
for (int i = 0; i < books.size(); i++) {
   Book book = books.get(i);
   // do something with book
}
```

++

Iterating over a list

- Variant a)

```
for (int i = 0; i < books.size(); i++) {
   Book book = books.get(i);
   // do something with book
}
```

- Variant b)

```
for (Iterator it = books.iterator(); ~~books~~ it.hasNext(); ) {
   Book book = it.next();
   // do something with book
}
```

end condition

initial value

next element

Iterating over a list

- Variant a)

```
for (int i = 0; i < books.size(); i++) {
   Book book = books.get(i);
   // do something with book
}
```

- Variant b)

```
for (Iterator it = books.iterator(); books.hasNext(); ) {
   Book book = it.next();
   // do something with book
}
```

- Variant c) *recommended* way

```
for (Book book : books) {
   // do something with book
}
```

User-defined Exceptions

- Purpose: To notify the caller about some exceptional or error state of the method

```
public void addBook(Book book)
  throws OperationNotAllowedException {
  if (!adminLoggedIn())
    throw new OperationNotAllowedException(...);
  ...
}
```

adminLoggedIn == false

- Creating a user defined exception

Error

```
public class OperationNotAllowedException extends Exception {
  public OperationNotAllowedException(String errorMsg) {
    super(errorMsg);
  }
}
```

- Throwing a user-defined exception

```
throw new OperationNotAllowedException("some error message");
```

Create exception

Checked vs. unchecked Exceptions

- Checked Exception

```
public class MyCheckedException extends Exception {...}
```

→ Methods which throw MyCheckedException must have throws MyCheckedException in the signature, e.g.

```
public void m() throws MyCheckedException {...}
```

- Unchecked Exception

```
public class MyUncheckedException extends Error {...}
```

→ Methods don’t need the throw clause

User-defined Exceptions: Example

- Catching an user-defined exception

```
try {
   libApp.addBook(book1);
} catch (OperationNotAllowedException e) {
 // Error handling code
}
```

Compiler error: Unreachable catch block

- Code added by Eclipse

  ```
  public void addBook(Book book) { }
  ```

- Test code

  ```
  try {
    libApp.addBook(book1);
    fail();
  } catch (OperationNotAllowedException e) { .. }
  ```

- Compiler error: ”Unreachable catch block for OperationNotAllowedException. This exception is never thrown from the try statement body”

Compiler error: Unreachable catch block

- Code added by Eclipse

  ```
  public void addBook(Book book) { }
  ```

- Test code

  ```
  try {
    libApp.addBook(book1);
    fail();
  } catch (OperationNotAllowedException e) { .. }
  ```

- Compiler error: ”Unreachable catch block for OperationNotAllowedException. This exception is never thrown from the try statement body”
- Solution

  ```
  public void addBook(Book book)
        throws OperationNotAllowedException { }
  ```

- Problem only occurs with *checked exceptions*

Testing and exceptions

- Test for the presence of an exception

```
@Test
public void testSomething() {
  ...
  try {
    // Some code that is expected to
    // throw OperationNotAllowedException
    assertFalse(libApp.adminLoggedIn());
    libApp.addBook(b);
    fail("Expected OperationNotAllowedException to be thrown");
  } catch (OperationNotAllowedException e) {
    // Check, e.g., that the error message is correctly set
    assertEquals(expected, e.getMessage());
  }
}
```

- Alternative test

```
@Test(expected=OperationNotAllowedException.class)
public void testSomething() {...}
```

- No try-catch if you don't test for an exception: JUnit knows best how to handle not expected exceptions

Delegate Responsibility

- Original

```
public List<Book> search(String string) {
   List<Book> booksFound = new ArrayList<Book>();
   for (Book book : books) {
      if (book.getSignature().contains(string) ||
         book.getTitle().contains(string) ||
         book.getAuthor().contains(string)) {
         booksFound.add(book);
      }
   }
   return booksFound;
}
```

a | b   or

a || b   seq. or

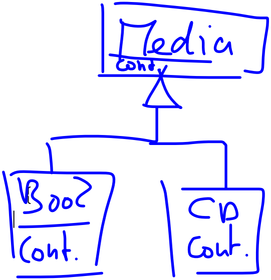
Better: With Delegation

- In class LibraryApp

```
public List<Book> search(String string) {
   List<Book> booksFound = new ArrayList<Book>();
   for (Book book : books) {
      if (book.contains(string)) {
         booksFound.add(book);
      }
   }
   return booksFound;
}
```

- In class Book

```
public boolean contains(String string) {
   return signature.contains(string) ||
          title.contains(string) ||
          author.contains(string)
}
```



CD

contains(____) {
ret titleList.contains(str)

Contents

Java

What are software requirements?

Requirements Engineering Process

Use Cases

Glossary

Summary

Basic Activities in Software Development

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

Requirements Engineering

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
  - Use Cases
  - Glossary
  - User Stories → later

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.

user requirements $\xrightarrow{\text{refined}}$ system req.

Types of Requirements

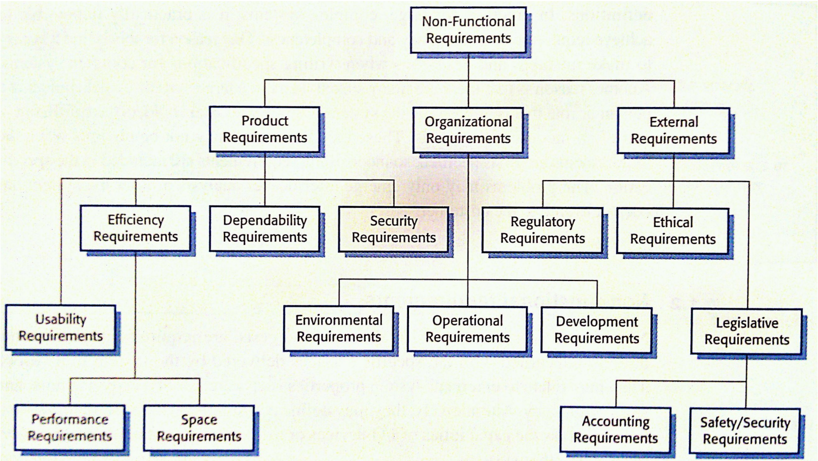
- User requirements
  - The requirements the user has
- System requirements
  - The requirements for the software development team
- Functional Requirements
  - E.g. the user should be able to plan and book a trip
- Non-functional Requirements
  - All requirements that are not functional
  - E.g.
    - Where should the software run
    - What kind of UI the user prefers

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)”

unclear

Categories of non-functional requirements



Ian Sommerville, Software Engineering - 9

Characteristics of good requirements

- Testability

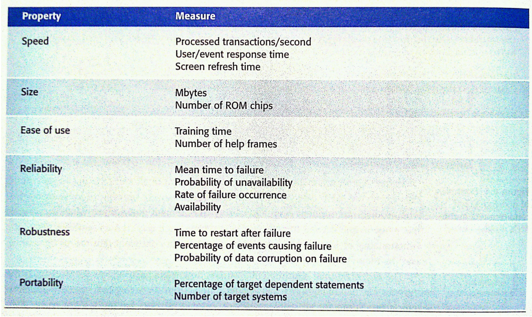
  → manual/automatic acceptance tests
- Measurable
  - Not measurable: *The system should be easy to use by medical staff and should be organised in such a way that user errors are minimised*

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

Ian Sommerville, Software Engineering - 9

Contents

Java

What are software requirements?

Requirements Engineering Process

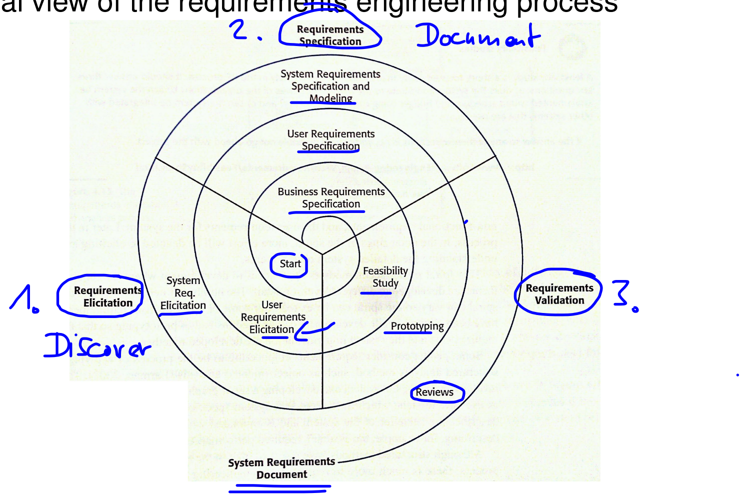
Use Cases

Glossary

Summary

Requirements engineering process

A spiral view of the requirements engineering process


2.
Requirements Specification
Document
System Requirements Specification and Modeling
User Requirements Specification
Business Requirements Specification
Start
Feasibility Study
1.
Requirements Elicitation
System Req. Elicitation
User Requirements Elicitation
Discover
Prototyping
Requirements Validation
3.
Reviews
System Requirements Document

Ian Sommerville, Software Engineering - 9

Requirements Engineering Process: Techniques

- Elicitation
  - Interviews
  - Glossary ← Understanding problem domain
  - Use Cases / User Stories
- Specification
  - Glossary
  - Use Cases / User Stories
- Validation
  - Inspection
    - Validity, Consistent, Complete, Realistic, . . .
  - Creation of tests

Contents

Java

What are software requirements?

Requirements Engineering Process

Use Cases

Glossary

Summary

Use Case

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 → graphical

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

Detailed Use Case description → textual

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

Use Case Example: Travel Agency use case *list available flights*

name: list available flights

description: the user checks for available flights = goal

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)

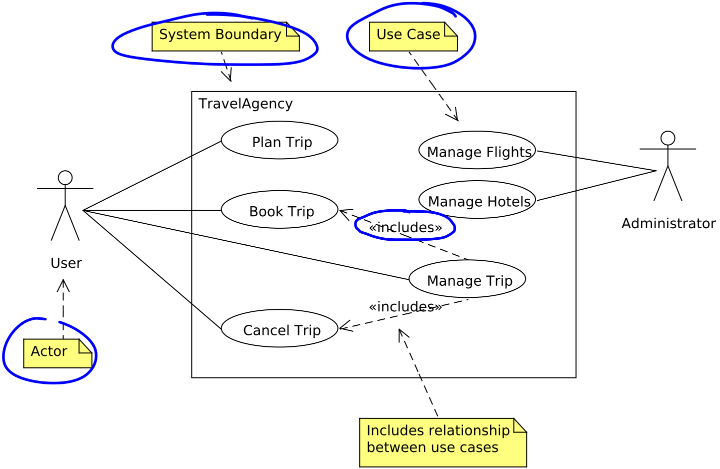
   2. The system notifies the user of that fact and terminates and starts the use case from the beginning

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

Use Case Diagram



Types of use case diagrams

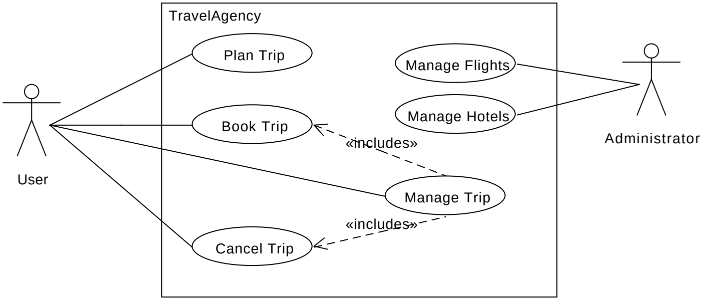
a) *Business use cases* (*kite level* use case (from Alistair Cockburn))

b) *System use cases* / *sea level* use case

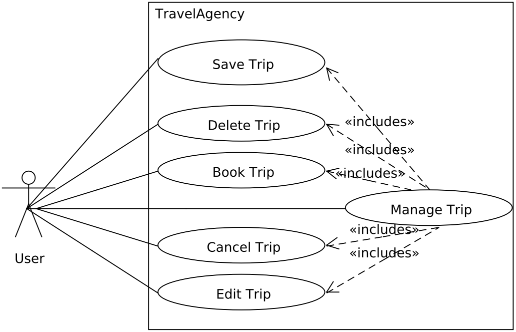
c) Use cases included in *sea level* use cases are called *fish* level use cases by Alistair Cockburn

UML Destilled, Martin Fowler

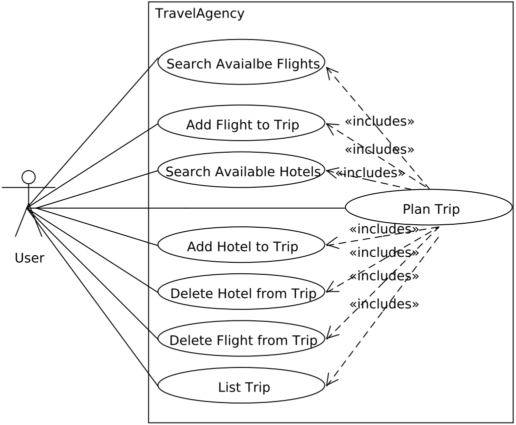
Travel Agency functional requirements: Business use cases



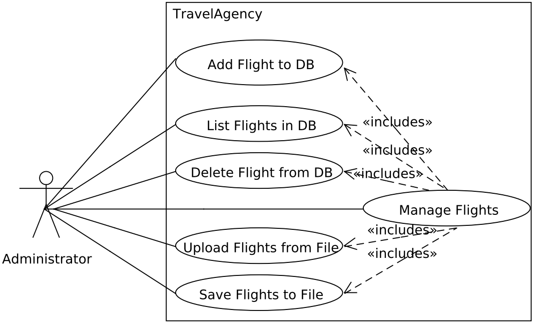
Travel Agency functional requirements: System use cases Part I: *manage trip*



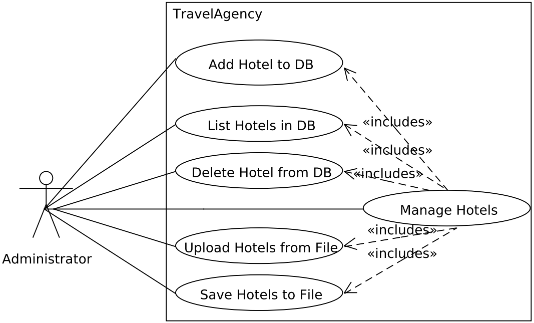
Travel Agency functional requirements: System use cases Part II: *plan trip*



Travel Agency functional requirements: System use cases Part III: *manage flights*



Travel Agency functional requirements: System use cases Part IV: *manage hotels*



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 -> goal

actor: One or more actors who interact with the system

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

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

alternative scenarios: can include fail scenarios

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

Travel Agency: detailed use case *list available flights*

name: list 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)

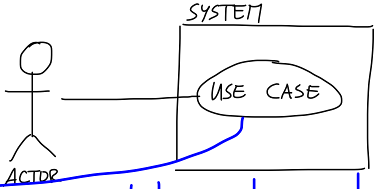
  2. The system notifies the user of that fact and terminates and starts the use case from the beginning

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

Scenarios



encapsulate to achieve a goal

- Interaction between an actor and the system
  - Anything the user does with the system 1)
  - System responses 2)
  - Effects visible/important to the customer 3)
- Not part of the interaction: What the system internally does

Travel Agency: 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

Travel Agency: detailed use case *plan trip*

This use case *includes* other use cases

name: plan trip

description: The user plans a trip consisting of hotels and flights

actor: user

main scenario:

repeat any of the following operations in any order until finished

1. list available flights (use case)
2. add flight to trip (use case)
3. list available hotels (use case)
4. add hotel to trip (use case)
5. list trip (use case)
6. delete hotel from trip (use case)
7. delete flight from tip (use case)

Note: the trip being planned is referred to as the current trip

Travel Agency: detailed use case *save trip*

name: save trip

description: provides the current trip with a name and saves it for later retrieval

actor: user

precondition: the current trip is not empty

main scenario:

1. user provides a name for the trip

alternative scenarios:

1: the name is not valid

  2: notify the user of the fact and end the use case

1: a trip with the name already exists

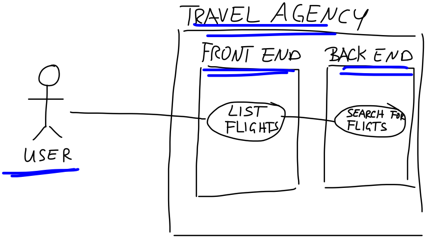
  2: ask the user if the trip should overwrite the stored trip

  3a: If yes, overwrite the stored trip

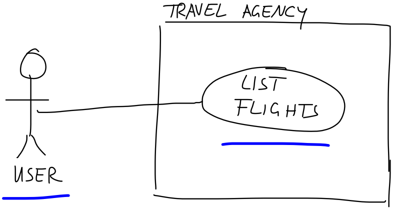
  3b: If no, end the use case

Use cases and system boundary

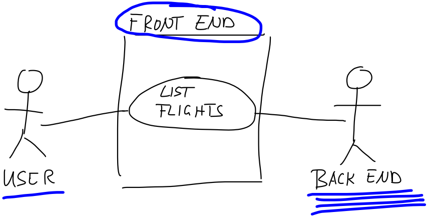
Actors and use cases depend on the system boundary



- System Boundary: Travel Agency



- System Boundary: Front end of the travel agency



Contents

Java

What are software requirements?

Requirements Engineering Process

Use Cases

Glossary

Summary

Glossary

problem

- Purpose: capture the **customer’s knowledge** of the domain so that the **system builders** have the *same knowledge*

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.

Example

Part of a glossary for the travel agency

User: The person who is using the travel agency

Trip: A trip is a collection of hotel and flight informations. A trip can be booked and, if booked, cancelled.

Booking a trip: A trip is booked by making a hotel reservation for the hotels on the trip and a flight booking for the flights of the trip

Flight booking: The flight reservation is booked with the flight agency and is payed.

Reserving a hotel: A hotel is reserved if the hotel informed that a guest will be arriving for a certain amount of time. It is possible that the hotel reservation requires a credit card guarantee.

. . .

2. Sources: i) problem descr.
ii) Use cases

- *Warning*
  - Capture only knowledge relevant for the application
  - Don't try to capture all possible knowledge

Contents

Java

What are software requirements?

Requirements Engineering Process

Use Cases

Glossary

Summary

Summary

- Requirements analysis is about finding out *what* the software should be able to do, not *how*
- Types: *functional* and *non-functional* requirements
- Qualities: *testable* and *measurable*
- Process: *Discover* (Elicitation), *Document* (Specification), *Validate* (Validation)
- Use cases
  - Used for both *finding* and *documenting* the requirements
  - What are the *functions* the user can perform with the software?
    - Detailed use cases: *Detailed* (textual) description of what the software should do
    - Use case diagram: *Graphical overview* over the functionality of the software
- Glossary: Document *domain knowledge* and define a *common language* between customer and software developer

Exercises

- For this week
  - `http://www2.imm.dtu.dk/courses/02161/2013/slides/exercise02.pdf`
- Still ongoing: programming exercises
  - `http://www2.imm.dtu.dk/courses/02161/2013/index2.html`