Introduction to Software Engineering
(2+1 SWS)
Winter Term 2009 / 2010

Dr. Michael Eichberg
Vertretungsprofessur Software Engineering
Department of Computer Science
Technische Universität Darmstadt
Statement:
Mary had a little lamb.

Meaning:
to have
1. to hold in possession as a property
2. to trick or fool someone (been had by a partner)
3. to beget or bear (have a baby)
4. to partake (have as a dinner)
5. ....

Statement:
Mary had a little lamb.

Meaning:
a lamb

1. a young sheep less than one year ...
2. the young of various other animals (antelope etc.)
3. a person as gentle or weak as a lamb
4. a person easily cheated or deceived
5. the flesh of lamb used as food
6. ...

Statement: Mary had a little lamb.

Meaning: ?
Statement:
Mary had a little lamb.

Meaning:

<table>
<thead>
<tr>
<th>have</th>
<th>lamb</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>Mary owned a little sheep under one year...</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>Mary gave birth to an antelope.</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
Requirements are the descriptions of the services provided by the system and the operational constraints.

Requirements are described in the system requirements specification.
W.r.t. the level of description, we can distinguish two types of requirements: (A) User and (B) System requirements.

- **User requirements**
  They state - in natural languages plus diagrams - what services the system is expected to provide and the constraints under which it must operate. Usually written by the customer.
  (dt. Grundlage für das „Lastenheft“)

- …
W.r.t. the level of description, we can distinguish two types of requirements: (A) User and (B) System requirements.

- System requirements (dt. Systemanforderungen)
  Set out the system’s functions, services and operational constraints in detail. The system requirements document (also called a functional specification) should be precise and is often part of the contract. Written by the software developer. (dt. Grundlage für das „Pflichtenheft“)
User and System Requirements Exemplified

• User requirements definition
  • The Library system shall keep track of all data required by copyright licensing agencies in the UK and elsewhere.

• System requirements specification ...
User and System Requirements Exemplified

• User requirements definition ...

• System requirements specification
  • On making a request for a document from the library system, the requestor shall be presented with a form that records details of the user and the request made.
  • The library system’s request forms shall be stored on the system for five years from the date of the request.
  • All library system request forms must be indexed by user, by the name of the material requested and by the supplier of the request.
  • The library system shall maintain a log of all requests that have been made to the system.
  • ...
From the point-of-view of a developer, we can distinguish two types of requirements: (A) Functional and (B) Non-functional requirements.

- **Functional requirements**
  They specify the services that the system should provide, how the system should react to particular inputs and how the system should behave in particular situations.
  To fulfill these requirements it is usually necessary to “write code”.

Requirements Engineering - Introduction | 12
From the point-of-view of a developer, we can distinguish two types of requirements: (A) Functional and (B) Non-functional requirements.

**Non-functional requirements**
Constraints on the services or functions offered by the system; including: timing constraints, constraints on the development process and standards. They often apply to the system as a whole. It is usually not possible to write some well-defined piece of code to fulfill these requirements.
The boundaries between functional and non-functional requirements are often not clear-cut and if you take a more detailed look on a non-functional requirement (e.g. “the system has to be secure”) it might result in the identification of functional requirements.
“Some” Types of Non-functional Requirements

Product requirements
- Portability requirements
- Reliability requirements
- Efficiency requirements
- Usability requirements

Organizational requirements
- Delivery requirements
- Implementation requirements
- Standards requirements

External requirements
- Interoperability requirements
- Ethical requirements
- Legislative requirements
- Safety requirements
- Privacy requirements

Non-functional requirements

Ian Sommerville - Software Engineering 8; Addison Wesley 2007
Often non-functional requirements are more critical than individual functional requirements. (E.g. a banking system that does not support the export of the bank statement as PDF is probably still useable; if the system is not secure, it is worthless).
Often non-functional requirements are more critical than individual functional requirements. (E.g. a banking system that does not support the export of the bank statement as PDF is probably still useable; if the system is not secure, it is worthless).

But, how to evaluate if a non-functional requirement is met?
(Let's assume that we are going to develop a new web shop)
How about the following non-functional requirement:

The user interface should be easy to use.

not very useful
How about the following non-functional requirement:

The user interface should be easy to use.
(Let's assume that we are going to develop a new web shop)

How about the following non-functional requirement:
- The number of forms that fail server-side validation due to input errors should be less than $Y$ percent.
- An average user should be able to make an order in less than $X$ minutes.
- The number of not completed transactions that should not exceed $Z$ percent.
- After one day of training an agent should be able to handle twice as many orders.

These requirements could be indirect measures of a user interface’s quality.
Domain requirements are derived from the application domain of the system rather than from the needs of the system users.

- Usually **expressed using domain-specific terminology**; hard to understand by software engineers.
- May **not be explicitly stated** since they are obvious to the domain expert.
- Can be functional or non-functional.
The “Software Requirements Document” states what the developers should implement. (Software Requirements Document ~dt. Pflichtenheft)

• The document has a diverse set of users:
  • System customers,
  • Managers,
  • System engineers,
  • System test engineers,
  • System maintenance engineers.

• ...
The “Software Requirements Document” states what the developers should implement.

(Software Requirements Document ~dt. Pflichtenheft)

- The level of detail depends on:
  - The type of system.
  - The development process that is used.
  - Where the system is build: external contractor or in-house.

1. **Introduction**
   a. Purpose of the requirements document
   b. Scope of the product
   c. Definitions, acronyms and abbreviations
   d. References
   e. Overview

2. **General description**
   a. Product perspective
   b. Product functions
   c. User characteristics
   d. General constraints
   e. Assumptions and dependencies

3. **Specific requirement**

4. **Appendices**

5. **Index**
Requirements are the descriptions of the services provided by the system and the operational constraints.

Requirements are described in the system requirements specification.

Requirements engineering is the process of:
- finding out,
- analyzing,
- documenting and
- checking these services and constraints.

(Requirements Engineering dt. Anforderungsanalyse)
The goal of the requirements engineering process is to **create and maintain a system requirements document**.
The Requirements Engineering Process

Feasibility study

Requirements elicitation and analysis

Requirements specification

Requirements validation

System models

User and system requirements

Requirements document

Feasibility report
The result of a feasibility study is a report that recommends whether or not it is worth carrying on with the requirements engineering and system development process.

• Input to the study:
  • Preliminary business requirements.
  • Outline description of the system.
  • How the system is intended to support business processes.
The result of a feasibility study is a report that recommends whether or not it is worth carrying on with the requirements engineering and system development process.

- The report is a short document that should answer the following questions:
  - Does the system contribute to the overall objectives of the organization?
  - Can the system be implemented using current technology and within given cost and schedule constraints?
  - Can the system be integrated with other systems which are already in place?
The Requirements Elicitation and Analysis Process

(dt. Anforderungsermittlung)

- Requirements discovery
- Requirements classification and organization
- Requirements prioritization and negotiation
- Requirements documentation
Requirements discovery is the process of interacting with stakeholders in the system to collect their requirements.

- Major problem: How to systematically discover requirements?

- An approach: Viewpoint-oriented approaches.
• Generic types of viewpoints:

  • **Interactor viewpoints**
    People that will interact with the system.

  • **Indirect viewpoints**
    Stakeholders that influence the requirements, but who will not directly use the system.

  • **Domain viewpoints**
    Domain characteristics and constraints that influence the system requirements.
Viewpoint-oriented approaches

• During the elicitation you should try to identify more specific viewpoints.
• After discovering the most important viewpoints start with them to discover the requirements.
Techniques for Requirements Elicitation

• Interviews
Interviews should only be used alongside other requirements techniques, because interviewees use domain knowledge the interviewer may not be familiar with, are reluctant to reveal the actual structure, may work against the project if (they think) their job is at stake.

Types of interviews:
• Closed interviews where the stakeholder answers a predefined set of questions
• Open interviews with no predefined agenda
Techniques for Requirements Elicitation

• Scenarios
Scenarios cover possible interactions with the system. The interactions are roughly outlined at the beginning and are detailed during the elicitation. Most people can understand and critique a scenario of how they might interact with the system. Scenarios are particularly useful for adding detail to an outline requirements description.

• Use cases (will be covered later on)

• ...
Requirements classification and organization

Given the unstructured collection of requirements, the requirements are grouped and organized into coherent clusters.
Requirements prioritization and negotiation

The requirements are prioritized and conflicts are found and resolved through negotiation.
Requirements documentation

The requirements are documented and used as input for the next round in the spiral (the produced documents may be formal or informal).
Use cases are text stories, widely used to discover and record requirements.

A use case encapsulates a set of actions that are executed in a well defined order.

Use cases influence many other artifacts, e.g. analysis, design, implementation and test artifacts.

Remember, use cases can not be used for all kinds of requirements!
Use cases are text stories, widely used to discover and record requirements.

**Process Sale**: A customer arrives at a checkout with items to purchase. The cashier uses the POS system to record each item. The system presents a running total and line-item details. The customer enters payment information, which the system validates and records. The system updates inventory. The customer receives a receipt from the system and then leaves with the items.
Use Cases - some Definitions

• Actor
  ... something with behavior; e.g. a person, computer system or organization.

• Scenario
  [Also known as a “use case instance”.
  ... a specific sequence of actions and interactions between actors and the system. It is one particular story using a system, or one path through the use cases.

• Use case
  ... is a collection of related success and failure scenarios that describe an actor using a system to support a goal.
Use Case Formats

• “Brief”
  (dt. kurz)
  Terse one-paragraph summary, usually of the main success scenario (as seen above).

• “Casual”
  (dt. ungezwungen, zwanglos)
  Informal paragraph format. Multiple paragraphs that cover various scenarios.

• “Fully dressed”
  (dt. vollständig bearbeitet)
  All steps and variations are written in detail, and there are supporting sections, such as preconditions and success guarantees.
Focus on the accuracy of use cases before you focus on the precision.

- First, identify all currently relevant use cases at a very high level
  (low precision, high accuracy)
- Second, work out the details.
  (add precision)
# A Template for Fully Dressed Use Cases

(Created by Alistair Cockburn, alistair.cockburn.us)

<table>
<thead>
<tr>
<th>Use Case Section</th>
<th>Purpose / Guidelines</th>
</tr>
</thead>
<tbody>
<tr>
<td>Use Case Name</td>
<td>Start with a verb.</td>
</tr>
<tr>
<td>Scope</td>
<td>The system under design.</td>
</tr>
<tr>
<td>Level</td>
<td>&quot;summary goals&quot; → &quot;user goals&quot; → &quot;subfunction&quot;</td>
</tr>
<tr>
<td>Primary Actor</td>
<td>Calls on the system to deliver its services.</td>
</tr>
<tr>
<td>Stakeholders and Interests</td>
<td>Who cares about this use case, and what do they want?</td>
</tr>
<tr>
<td>Preconditions</td>
<td>What must be true on start, and worth telling the reader?</td>
</tr>
<tr>
<td>Minimal Guarantees</td>
<td>The fewest promises the system makes to the stakeholders.</td>
</tr>
<tr>
<td>Success Guarantee</td>
<td>What must be true on successful completion, and worth telling the reader?</td>
</tr>
<tr>
<td>Main Success Scenario</td>
<td>A typical path scenario of success.</td>
</tr>
<tr>
<td>Extensions</td>
<td>Alternate scenarios of success or failure.</td>
</tr>
<tr>
<td>Special Requirements</td>
<td>Related non-functional requirements.</td>
</tr>
<tr>
<td>Technology and Data Variation List</td>
<td>Varying I/O methods and data formats.</td>
</tr>
<tr>
<td>Frequency of Occurrence</td>
<td>Influences investigation, testing and timing of implementation.</td>
</tr>
<tr>
<td>Miscellaneous</td>
<td>Such as open issues.</td>
</tr>
<tr>
<td>Use Case Section</td>
<td>Explanation</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Stakeholders and Interests</td>
<td>Someone or something with an interest in the behavior of the system under discussion. E.g. company stakeholders, customers, vendors, and government regulatory agencies,…</td>
</tr>
<tr>
<td>Preconditions</td>
<td>It announces what the system will ensure is true before letting the use case start. Since it is enforced by the system and known to be true, it will not be checked again during the use case execution; e.g. user has logged in.</td>
</tr>
<tr>
<td>Minimal Guarantees</td>
<td>The fewest promises the system makes to the stakeholders, particularly when the primary actor’s goal cannot be delivered. They hold when the goal is delivered, but they are more important when the main goal is abandoned. E.g. the system logged all performed steps.</td>
</tr>
<tr>
<td>Success Guarantee</td>
<td>States what interests of the stakeholders are satisfied after a successful conclusion of the use case, either at the end of the main success scenario or at the end of a successful alternative path. The success guarantees are written additionally to the minimal guarantees.</td>
</tr>
</tbody>
</table>
Excerpt of a Fully Dressed Use Cases

<table>
<thead>
<tr>
<th>Name:</th>
<th>Buy Stocks over the Web</th>
</tr>
</thead>
<tbody>
<tr>
<td>Primary Actor:</td>
<td>Purchaser</td>
</tr>
<tr>
<td>Scope:</td>
<td>Finance Package (PAF)</td>
</tr>
<tr>
<td>Level:</td>
<td>User goal</td>
</tr>
</tbody>
</table>

Stakeholders and Interests:

- Purchaser - wants to buy stocks and get them added to the portfolio. 
- Stock agency - wants full purchase information.

Precondition:

- User is logged in.

Minimal guarantee:

- Sufficient logging information will exist so that the PAF can detect that something went wrong and ask the user to provide details.

Success guarantee:

- Web site has acknowledged the purchase; the logs and the user’s portfolio are updated.
Excerpt of a Fully Dressed Use Cases

Name: Buy Stocks over the Web
Scope: Finance Package (PAF)

Main Success Scenario:

1. Purchaser selects to buy stocks over the web
2. PAF gets name of web site to use (A, B,...) from user
3. PAF opens web connection to site, retaining control
4. Purchaser browses and buys stock from the web site
5. PAF intercepts responses from the web site and updates purchaser’s portfolio
6. PAF shows the user the new portfolio standing
**Name:** Buy Stocks over the Web  
**Scope:** Finance Package (PAF)

...  

**Extensions:**

2a. Purchaser wants a web site PAF does not support  
   2a1. System gets new suggestion from purchaser, with option to cancel  

4a. Web site does not acknowledge purchase, but puts it on delay  
   4a1. PAF logs the delay, sets a timer to ask the purchaser about the outcome  

...
UML use case diagrams provide a notation to illustrate the names of use cases and actors, and the relationship between them.

(Use-case diagram = dt. Anwendungsfall-Diagramm)
UML use case diagrams provide a notation to illustrate the names of use cases and actors, and the relationship between them.
The **UML use case diagram** is trivial to learn, but identifying and writing good use cases may take weeks. Organizing use cases into relationships has no impact on the behavior or requirements of the system. **It is an organization method to improve communication and comprehension of the use cases and to reduce duplication of text.**
UML use case diagram provide a black-box view on a system.

They are particularly useful during the early phases of a software project.
For partial behavior that is common across several use cases (e.g. “Pay by Credit” occurs in “Process Sale”, “Process Rental”,...) it is desirable to separate it into its own sub-function use case and indicate its inclusion.

The primary purpose is to avoid repetition or to decompose extremely long use cases to make them comprehensible.
The extend relationship can be used to describe where and under what condition an extending or additional use case extends the behavior of some base use case.

- **Process Sale**
  - **Extension Points:**
    - Payment
    - VIP Customer
  - «extend» Payment, if Customer presents a gift certificate
The Inheritance Relationship in Use Case Diagrams

The inheriting use case replaces one or more of the courses of action of the inherited use case. The inheriting use case overrides the behavior of the inherited use case.

Inheritance between use cases is not very common.