Chapter 4.1: Software Application Engineering – General Introduction
The goals of this chapter are to

- understand the application of engineering principles to specifying requirements for software systems (what do the principles in chapter 3 mean for software system requirements development & specific technologies?)
- be able to apply selected system specification (User Requirements) and acceptance testing approaches
- be able to understand the pros & cons of different user requirements notations & approaches
- understand how to bridge the gap between implicit user expectations and explicit user requirements
- be able to create & verify user requirements
- understand what documentation has to be stored together with a software system in order to make it reusable

Literature

Motivation

- Communication in RE and SE
- RE Problems
- System vs Software Model

Introduction and Principles

Problem Description

System Requirements
- Functional User Requirements
- Functional Developer Requirements
- Requirements Categories
- Cost Estimation
- Documentation

Requirements Specification
Communication in RE and SE

- Motivation
  - Communication in RE and SE
  - RE Problems
  - System vs Software Model

- Introduction and Principles

- Problem Description

- System Requirements
  - Functional User Requirements
  - Functional Developer Requirements
  - Requirements Categories
  - Cost Estimation
  - Documentation

- Requirements Specification

As proposed by the project sponsor
As specified in the project request
As designed by the senior analyst

As produced by the programmers
As installed at the user's site
What the user wanted
### Problems in RE

- **Effects of Inadequate RE – Airbus**
  - **Requirement:** "Reverse thrust may only be used, when the airplane is on the ground."
  - **Translation:** "On the ground” means “wheels are turning."
  - **Implementation:** "Reverse thrust may only be used when wheels are turning!"
  - **Situation:** Rainstorm – aquaplaning on the runway
  - **Result:** Crash due to overshoting the runway!

  ⇒ **Problem:** erroneous modeling (mapping of real-world concepts into computer-recognizable concepts) in the requirement phase

- **Typical Problems:**
  - **Imprecise requirements**
    - "Shut the system down if the water level is below X"
    - "Enable reverse thrust if the airplane is on the ground"
    - "Acceptable performance"
    - "Easily maintainable" etc.
  
  - **Software engineering solutions** include
    - Precise 1-1 translation of real-world concepts into computer concepts
    - Scenario based specification (modularization by scenarios) (e.g., SBS)
    - Mathematically precise specification (e.g., SBS)
    - Quantifiable non-functional requirements
System vs. Software Model

- Motivation
- Communication in RE and SE
- RE Problems

- Introduction and Principles

- Problem Description

- System Requirements
  - Functional User Requirements
  - Functional Developer Requirements
  - Requirements Categories
  - Cost Estimation
  - Documentation

- Requirements Specification

© Prof. Dr. Dr. h. c. Dieter Rombach, Fundamentals of Software Engineering, Winter Term 2011/12
Goals

- Derive adequate system requirements for the problem to be solved
- Check these for consistency with the problem description (verification)
- Validate the system in its target environment against these system requirements (validation)
Characteristics at the Requirements Level

- The starting point (problem description) is usually informal (text), incomplete and inconsistent

- Verification against the problem description is particularly difficult (! Inspections work, iff user understands requirements!)

- Interface between users/customers and software developers (computer scientists) creates problems
  - Language / terminology
  - Notational preferences
  - Conflicts of interest among stakeholders:
    → Customer ↔ Contractor

- Requirements documents often serve as legal contracts

- A “good” requirement document Should be
  - Easy to understand (customer, developer)
  - Easy to check (verification against problem description)
  - Easy to change (→ problem description initially incomplete and inconsistent)
  - Suitable as a reference document for V&V (verification of the system design, validation of the executable system)
Basic Principle 1: Understandability

- Understandable presentation (e.g., SBS)
  - Appropriate notation
  - Appropriate structure
  - Complete/consistent

- Minimal content
  - Only the necessary requirements should be presented, not all possible requirements
Basic Principle 2: Structuring

- **Modular structure (divide and conquer)**
  - Requirements must be structured in a modular way to cope with complexity
    → Necessary for efficient testing and incremental development
  - Structuring allows local and aggregated understanding
  - E.g.
    → Standard sections
    → Individual numbering of requirements
Basic Principle 4: Horizontal Traceability

- Each description contains different views (e.g. structure, data flow, event flow)
- These must be inter-related and consistent (e.g. UML models)

Where A, B, C are examples of objects, methods, or attributes.
Basic Principle 5: Vertical Traceability (1/2)

- Different descriptions (problem description, user requirements, developer requirements and design) must be mutually consistent.
- Consistency means that every element of level (1, 2, 3) is traceably realized by elements at level i+1.
- User and developer requirements are distinguished less by their content than by their viewpoints as well as different user-oriented notations.
- Traceable realization means that the relationship between different abstractions is clearly and explicitly documented.

PD:  

C-Req:  

D-Req:  

Design  

Level 1  

Level 2  

Level 3  

Level 4
Basic Principle 5: Vertical Traceability (2/2)

C-requirements:

\[ a \xrightarrow{f_1} b \xrightarrow{f_2} c \}\)

- DFG
- Data dictionary

D-requirements:

Data types: (from C-requirements or refinement)

Functions: (from C-requirements or refinement)

\[ f_1: \text{input: } a \]
\[ \text{output: } b \]
\[ \text{transformation: } (a < 0 \rightarrow 0 \mid a^2) \]

\[ f_2: f_{21}: \text{input: } a \]
\[ \text{output: } b \]
\[ \text{transformation: } \ldots \]

\[ f_{22}: \ldots \]
Basic Principle 6: Explicit Documentation of the “is verified” Relation (1/2)

- Documentation of external consistency (Di+1 → Di)

**Diagram:**

- Problem
  - C-Requirements
  - D-Requirements
- System Design
- Component Design
- Component Code
- Executable Component(s)
- Executable System
- Usable System
- Deployed System

© Prof. Dr. Dr. h. c. Dieter Rombach, Fundamentals of Software Engineering, Winter Term 2011/12
Basic Principle 6: Explicit Documentation of the “is verified” Relation (2/2)

- The consistency check between two neighboring documents (i.e. correctness) must be traceably documented

- The following must be clearly documented:
  - Degree of verification
  - The results of the checking (faults)
  - The resulting changes

- This information is necessary for future changeability and reusability
### Basic Principle 7: Explicit Documentation of the “is validated” Relation

- **Documentation of external consistency** \( S_i \rightarrow D_i \)

- **The consistency between a system in execution and its description (i.e. reliability) must be traceable**

- **This requires the following to be clearly documented:**
  - Degree of validation
  - The results of the checking (faults)
  - The resulting changes

- **This information is necessary for future changeability and reusability**
Problem Description

- **Definition of the problem to be solved**

- **Can be based on:**
  - Market analysis
  - User interviews
  - Feedback from users
Problem Description - Structure

- **Motivation**
  - Communication in RE and SE
  - RE Problems
  - System vs Software Model

- **Introduction and Principles**

- **Problem Description**
  - **Mandatory**
    - Problem (What is the problem?)
      - Actual system (a)
      - Weak points, improvement wishes (functional and non-functional aspects)
  - **Optional**
    - Expected system functionality
      - New system (b)
      - Software sub-system
      - Contribution of sw sub-systems to the new system
      - Priorities (significance)
    - Environmental characteristics
      - Problem context (embedded into other systems or processes, legal / organizational constraints, ...)
      - System context (language, sub-system structure, ...)
      - Development context (process, reuse of components, experience, ...) (c)
    - Usage scenarios
      - User classes and User profiles

- **Requirements Specification**
  - Source of information
    - Users, customers, [legislators], documents....
Problem Description

(a) Actual system:
- Who are the persons affected?
- In which environment do these people work?
- What are improvement wishes / problems of these people?

(b) New system:
- What is to be achieved with the system?
- Which attributes must an acceptable solution definitely not have?
- Should a solution have a specific structure?
- How do the components of a solution belong together?
- When does a solution have to be ready in order to be acceptable?

(c) Development context:
- How does this problem differ from other systems known to us?
- Are there solutions that we can profit from?
- Is there commercial-off-the-shelf software that we can use?
- Are there partial solutions that make sense?
- Are there priorities from the customer view?
- Can I transform the problem into a known problem?
### User Requirements
- Definition of the functional and non-functional requirements from the perspective of the user
- Facilitate communication between users and developers
- Defines the behavior (dynamic, external) of the system/software and the interface to the users
- Verified against the problem description

### Developer Requirements
- Definition of the functional and non-functional features from the perspective of the designer and/or maintainer (static, external)
- Facilitates communication within the design and/or maintenance team
  - → often more formal than user requirements
- Typically derived from the user requirements in order to ensure traceability
- Verified against the user requirements
Functional User Requirements

- Describe the necessary (and only the necessary) requirements from the view of the user
- Should be minimal
  - Over specification should be avoided in order not to unnecessarily limit the options for solutions

Description views
- Interface (external, static)
- Behavior (external, dynamic)
- Structure (internal, static)
- Flow / Relations (internal, dynamic)
## Functional User Requirements

### Views versus Descriptions

<table>
<thead>
<tr>
<th></th>
<th>C-Requirements</th>
<th>D-Requirements</th>
<th>Design</th>
</tr>
</thead>
<tbody>
<tr>
<td>[System] Behavior</td>
<td>++</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>[System] Interfaces</td>
<td>+</td>
<td>++</td>
<td>0</td>
</tr>
<tr>
<td>[System] Structure</td>
<td>0</td>
<td>0</td>
<td>++</td>
</tr>
<tr>
<td>[System] Flows</td>
<td>0</td>
<td>0</td>
<td>++</td>
</tr>
</tbody>
</table>

### Emphasis of User Requirements

- Main emphasis on behavior
- Minor emphasis on
  - Interface
  - Structure/flow
  - Design decisions or guidance for operation definition
### Functional Developer Requirements

#### Emphasis of Developer Requirements
- Main emphasis on interface
- Minor emphasis on
  - Behavior
  - Structure/flow
  - Design decisions or guidance for operation definition

<table>
<thead>
<tr>
<th>Requirements</th>
<th>input / output</th>
<th>transformation</th>
<th>behavior</th>
</tr>
</thead>
<tbody>
<tr>
<td>natural language</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>EBNF (regular expressions)</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>conditional parallel instructions (Mills)</td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>decision tables</td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>STD (state transition diagrams)</td>
<td></td>
<td></td>
<td>X</td>
</tr>
</tbody>
</table>
**Requirement categories**

- Functional Requirements
- Non-Functional Requirements
- Inverse Requirements
- Design Decisions
- (Basic Cost Estimation)
All non-functional requirements (related to the entire system or to individual functions) are listed in tables, explained, and quantitatively defined.

Example non-functional requirements:

- **Reliability**
  \[ \rightarrow \text{e.g. maximum tolerable number of faults per time interval} \]

- **Performance**
  \[ \rightarrow \text{e.g. maximum tolerable response time} \]

- **Security**
  \[ \rightarrow \text{Data which must not be damaged through faulty use} \]

- **Portability**
  \[ \rightarrow \text{Possible changes to execution environment + costs} \]

- **Extensibility**
  \[ \rightarrow \text{Possible changes to the requirements + costs} \]
Requirement categories
- Inverse Requirements

- **Motivation**
  - Communication in RE and SE
  - RE Problems
  - System vs Software Model

- **Introduction and Principles**

- **Problem Description**

- **System Requirements**
  - Functional User Requirements
  - Functional Developer Requirements
  - Requirements Categories
  - Cost Estimation
  - Documentation

- **Requirements Specification**

- **Describe the situations which must never occur**

- **Examples (e.g., for a lecture planning system)**
  - IR1: Two lectures should never be scheduled for the same time
  - IR2: Graduate (Master) lectures must never take priority over undergraduate (Bachelor) lectures!

- **Inverse requirements require special attention during validation/verification**
- Design options which are no longer acceptable (i.e. have been ruled out)
  - Constraints on system architecture
  - Constraints on the development process
    → e.g. T/M/T’s to be used, organization standards
  - External influences
    → e.g. legal requirements, costs interoperability

- Examples
  - DD1: The system must be implemented in Java
  - DD2: The DBMS “Oracle” must be used

- Design decisions should only be made if absolutely necessary
Cost Estimation

- **Motivation**
  - Communication in RE and SE
  - RE Problems
  - System vs Software Model

- **Introduction and Principles**

- **Problem Description**

- **System Requirements**
  - Functional User Requirements
  - Functional Developer Requirements
  - Requirements Categories

- **Cost Estimation**

- **Documentation**

- **Requirements Specification**

---

We want to develop an understanding of the relative costs on the basis of the difficulties with realization.

- **For this, we assess the difficulty of each requirement on an ordinal scale**
  - E.g. simple [1], average [2], hard [3]

- **In the same way, their importance is assessed**
  - Desirable [1], required [2], absolutely required [3]
### Cost Estimation

<table>
<thead>
<tr>
<th>Importance to user</th>
<th>Requirements</th>
<th>Difficulty</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Functional Requirements</td>
<td>FR₁</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>FRₙ</td>
<td>3</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>Non-Functional Requirements</td>
<td>NFR₁</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>NFRₘ</td>
<td>2</td>
</tr>
<tr>
<td>...</td>
<td>Inverse Requirements</td>
<td>IR₁</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>IRₖ</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>Design Decisions</td>
<td>DD₁</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
<td>DDₜ</td>
</tr>
</tbody>
</table>

- **This table**
  - Must be maintained during the entire development project (like all other requirement artifacts)
  - Facilitates the use of so called cost prediction models
  - Provides information for validation activities
Motivation
Communication in RE and SE
RE Problems
System vs Software Model

Introduction and Principles

Problem Description

System Requirements
Functional User Requirements
Functional Developer Requirements
Requirements Categories
Cost Estimation Documentation

Requirements Specification

Problem-Solution Documentation

- Problem description
  - Problem
  - Expected system functionality
  - Environmental constraints

- User Requirements
  - User requirements
    - Tabular (Functional, Non-functional, Inverse, Development decisions)
    - UML (Use-cases, Use-case description, Scenarios)

- Developer requirements
  - Class diagrams
  - Data dictionary
  - Sequence diagrams
  - State chart diagrams

- Traceability Matrices

- Verification
  - Checklists
  - Fault classification
  - Results

- Validation
  - Test cases (acceptance)
  - Results of tests
  - Evaluation?
What are requirements?
- A statement of what is desired, or ought to be changed, in the environment/domain.

What are specifications of requirements?
- Specifications state the requirements for a system which, when correctly implemented, will achieve the desired change in the environment.
- Accurate description of what the system must do at its interface to its environment.

We often mean specifications when we talk of requirements!

Specifications must be
- Complete and Consistent
- Precise and accurate (i.e. understandable)
  - Unambiguous
  - Design flaws in software → Almost right means Wrong!
- Abstract (does not mean vague!)
  - Certain details omitted
    - Must be documented later/ elsewhere
    - Irrelevance of details justified
- Modularized and Traceable
- Verifiable (degree documented?)
- Validatable (degree documented?) → Correctness vs reliability?
## Requirements Specification
### - Specification Types

### Operational approaches (use of structure / flows for behavior description)
- DF diagrams (function-oriented description)
  - Functional requirements diagrams (or DFDs)
  - SADT
- State transition diagrams (control flow-oriented description)
  - SADT
  - FSMs ☼
  - State charts
  - Mode transition diagrams
  - Sequence Based Specification (SBS)
- Petri nets (control flow-oriented description, parallel)

### Descriptive approaches (behavior description via attributes)
- Entity relationship diagrams
- Relational approaches
  - PSL / PSA
- Logical approaches ☼
- Algebraic approaches ☼

(☼) Suitable primarily for developer-requirements or component requirements
### Requirements Specification

- **Motivation**
  - Communication in RE and SE
  - RE Problems
  - System vs Software Model

- **Introduction and Principles**

- **Problem Description**

- **System Requirements**
  - Functional User Requirements
  - Functional Developer Requirements
  - Requirements Categories
  - Cost Estimation
  - Documentation

- **Requirements Specification**

---

#### AE – General Introduction

**Requirements Specification**

- **Suitability for Different Problem Classes**

<table>
<thead>
<tr>
<th>Complexity focus</th>
<th>Type of System</th>
</tr>
</thead>
<tbody>
<tr>
<td>Functional</td>
<td>Event-controlled</td>
</tr>
<tr>
<td>Control flow</td>
<td>Information System (e.g. Reservation)</td>
</tr>
<tr>
<td>Data flow</td>
<td>Information System (e.g. Workflows)</td>
</tr>
<tr>
<td>Data structure</td>
<td>Information System (e.g. payroll)</td>
</tr>
</tbody>
</table>

© Prof. Dr. Dr. h. c. Dieter Rombach, Fundamentals of Software Engineering, Winter Term 2011/12
More about Requirements Engineering

- We offer every winter term a (master) lecture on Requirements Engineering

- Next in this lecture:
  - Requirement Engineering for Embedded Systems
  - Requirement Engineering for Information Systems
  - Requirement evolution and management