Softwaretechnik II
Requirements Engineering

Prof. Dr. Ralf H. Reussner
Motivation

- 60% of software defects coming from wrong requirements (Boehm 1981)
  - Requirements are missing
  - Requirements are wrong

- Solving these problems during development phase is very expensive

⇒ Requirements have to be considered well
⇒ Some aspects already in SWT I
What are Requirements

- Something required, wanted or needed

IEEE 610.12-1990 definition:

Requirement:

(1) A condition or capability needed by a user to solve a problem or achieve an objective.

(2) A condition or capability that must be met or possessed by a system or system component to satisfy a contract, standard, specification, or other formally imposed documents.

(3) A documented representation of a condition or capability as in (1) or (2).
Requirements

- **Functional Requirement:**
  - Defines a functionality that has to be fulfilled by the system or component

- **Non-Functional Requirement:**
  - Quality attributes like performance, security, reliability

- **Constraints:**
  - Organisational or technical specifications leading to implicit constraints (e.g., laws, regulations, ...)

Prof. Ralf Reussner – Softwaretechnik II
The RE Process

- Cooperative, iterative and incremental process
- Determine and comprehend all relevant requirements in necessary degree of detail
- Acknowledgement of requirements by involved stakeholders
- Document and specify requirements conforming to document and specification rules
The RE Process – Main Tasks

- Requirements Elicitation
  - Documentation of Goals
  - Scenarios
  - Problem Analysis
- Requirements Documentation
  - Create Software Requirements Specification (SRS)
- Cross cutting actions:
  - Requirements Validation
  - Requirements Management
The RE Process in Software Development

Two Kinds:

- Single phase -> heavy weight
  - Waterfall
- Multi phase incremental -> light weight
  - Agile development
  - Prototype and incremental processes
Stakeholder

- A person or organisation that (in)directly influences the requirements of a system
  - Users of the system
  - Operator of the system
  - Purchaser
  - Software Developer
  - Software Architects
  - Tester

- Identification of relevant Stakeholders critical task in the RE process
  - Missing Stakeholders lead to missing requirements
  - Use checklist for stakeholders
The Requirements Engineer

- Role in the development process
  - Perceives domain knowledge
  - Understands all stakeholder -> translates

- Has methodological skills
  - Thinks analytically
  - Has empathy
  - Has communication skills
  - Is conflict solving
  - Has discussion moderation skills
  - Is Selfconclous
  - Is Cogent (Überzeugend)
The Requirements Engineer

- Documents the Stakeholders
  - Name, Role, Availability, Knowledge area, Aims and Interests
- Stays in touch with Stakeholders
  - Keeps them informed
  - Perceives domain knowledge
- Maintains the requirements document
System Context and Scope

System Context:
- Part of the environment of a system relevant for definition and understanding of requirements

System Border (Scope):
- Border between editable system and noneditable environment
- Interface of the system to the environment
  - Information sources and sinks
  - Hardware and software

Context Border:
- Separates System Context from irrelevant Environment
Illustration System Context and Scope

- **Context Border**
- **Environment**
- **System Context**
- **System**
- **Hardware**
- **Document**
- **Interface**
- **Software A**
- **Software B**
- **System Border (Scope)**
System Context and Scope (cont‘d)

- Borders are usually unsharp in beginning
  - Unclear functionalities, environment, etc. lead to gray areas
  - Shifting of borders commonly occur
  - Esp. Scope has to be fixed during the RE process

- System Context and Scope have to be documented
  - Essential for requirements acquisition
  - Use-Case and data-flow diagrams usually used
Sources of Requirements

- Stakeholders
  - Domain knowledge

- Documents
  - Norms, standards and laws
  - Domain specific rules

- Other systems
  - Legacy systems
  - Concurrent systems
Prioritisation of Requirements

- Stakeholders have to rank requirements
- Priority scale has to be documented
- 4 priorities commonly used (MoSCoW):
  - Must – essential feature
  - Should – highly desirable feature
  - Can – desirable feature
  - Won’t – not essential / stakeholders agreed feature will not be implemented (in this release)
The Kano model [Kano84]

- Model for categorisation of requirements
  - Excitement (Attractive Quality)
    - Features that make a system attractive to use
    - Unaware requirements
  - Performance (One-dimensional Quality)
    - Critical key functionality
    - Explicit requirements
  - Threshold (Must-be Quality)
    - Taken for granted basic attributes
    - Often implicit requirements

- Shows impact of requirements on customer satisfaction
The Kano model

- Absent
- Low
- Performance
- Threshold
- Excitement
- Fully Implemented
- High

Customer Satisfaction vs. Product Function
Requirements Elicitation Techniques

I'll need to know your requirements before I start to design the software.

First of all, what are you trying to accomplish?

I'm trying to make you design my software.

I mean what are you trying to accomplish with the software?

I won't know what I can accomplish until you tell me what the software can do.

Try to get this concept through your thick skull: the software can do whatever I design it to do!

Can you design it to tell you my requirements?
Requirements Elicitation Techniques

- Questioning techniques
  - Interviews, Questionnaire, On-Site-Customer
- Creativity techniques
  - Brainstorming, Changing the Perspective, Analogy
- Retrospective techniques
  - System archaeology, Reuse
- Observation techniques
  - Field observation, Apprenticing
- Supporting actions
  - Mind Maps, Workshops, CRC-Cards, Audio and video recording, Use Case Modeling, Prototypes
Goal-oriented Requirements Engineering

Goal:

- A goal is an objective the system under consideration should achieve.
- Goal formulations refer to intended properties to be ensured
- Optative statements

Can be at different level of abstraction

- Strategic vs. low-level, technical

Different types

- Functional vs. non-functional concerns
Why are Goals needed

- Goals used for:
  - Requirements Acquisition
  - Documenting intention of diff. Stakeholders explicitly
  - Finding dependencies and conflicts between goals
  - Identifying and evaluating alternatives
  - Finding irrelevant requirements
  - Explaining requirements to stakeholders
Modeling Goals

- Goals can be refined / decomposed
  - Goal is refined by a set of subgoals
  - AND-refinement
    - All subgoals have to be satisfied
  - OR-refinement
    - Min. one subgoal has to be satisfied

- Goal dependencies
  - Supporting goals
  - Conflicting goals
  - Goal equivalence

- Can be visualised as graph
Specifying Goals

Seven Rules for Goals formulation:

- Short and concise description
- Use active speech
- Describe verifiable goals
- Refine non verifiable goals
- Document benefits of goals
- Document rational behind goal
- No solutions
Example - And-Or Trees

AND

Dynamic adaption to traffic conditions

Comfortable navigation to destination

Comfortable input of destination

Comfortable routing

OR

Manual input of traffic information

Automatic refreshment of traffic information

...

Prof. Ralf Reussner – Softwaretechnik II
Scenarios

Recap. SWT I Scenarios:
- Description of events, sequence of actions or events
- Description of system usage
- UML use case diagrams
  - Actors
  - Use-Cases
  - Use-Case models

A scenario describes a concrete example of (not) reaching one or several goals

Scenarios are used as Middle Level Abstraction
Scenario as Middle Level Abstraction

Conceptual Model

Scenario

Reality

High

Level of abstraction

Low
Relation between Goals and Scenarios

Goals

Scenarios

initiate and influence definition of

classify

concretise

lead to refinement of

lead to identification of

lead to revision of
Example: Use-Case Diagram

- **User**
  - Input Destination
  - Calculate Route (includes)
  - Calculate Position (includes)
  - Show Location (extends)
  - Input Traffic Information manually (extends)
  - Input Traffic Information
  - Manual input
  - Positioning System
  - Traffic Provider
Problem Analysis (PA)

- Refines goals and scenarios
- Learning about the problem to be solved
- Understanding needs, constraints, who is the user
  - Product-space, all possible software solutions
  - What does the user really need vs. What he wants
- Customer interests
  - Functions, performance, development time, cost, maintainability, modifiability and reliability
Three traditional perspectives of PA

- Three perspectives of problem analysis
  - Structural / data perspective
    - Static structure of data
  - Functional perspective
    - Manipulation of data by functions of the system
    - Transformation of input to output data
  - Behavioral perspective
    - Describes behavior of the system
    - Reaction to external stimuli
      - States
      - State changes
      - Output
Objects, Functions and States

- **Object:**
  - Any real-world entity, important to the discussion of requirements, with crisply defined boundary
  - Often grouped in classes

- **Function:**
  - Task, service, process, math. function
    - Being performed in real world
    - To be performed by a system
  - Often grouped hierarchically
Objects, Functions and States

- **State:**
  - Condition of some ‘thing’, captures history
  - Determines behavior in specific circumstances
  - ‘Thing‘ can be system, object or function

- **Requirements:**
  - Define objects, functions or states
  - Limit or control actions on objects, functions or states
  - Define relationships between objects, functions or states
  - Can be defined at diff. level of detail / abstraction
Knowledge Tree

- Capture structural relationships between objects, functions and states
  - Partitioning: aggregation / part of
  - Abstraction: general / specific, example of, instance of
  - Projection: view of

```
<table>
<thead>
<tr>
<th>Navigation System</th>
<th>Navigate</th>
<th>Destination</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input of destination</td>
<td>Navigate on road</td>
<td>Address</td>
</tr>
<tr>
<td>Navigation in progress</td>
<td>Navigate off road</td>
<td>GPS Position</td>
</tr>
<tr>
<td>Destination reached</td>
<td>Navigate in tunnel</td>
<td>Point of interest</td>
</tr>
</tbody>
</table>
```

State | Function | Object
Rule of Thumbs for PA

- Avoid doing „Software Design“
  - Define what and not how
  - Borderline between what and how depends on view

- Different Problems require different Solutions
  - Give some one a hammer and every problem starts to look like a nail
PA Approaches Overview

- Structural models
  - ER
- Function models
  - SADT
  - SSA
- Behavioral models
  - State machines
  - Statecharts
- Object-oriented models
Example: Object Oriented PA

- Recap. SWT I 2.3 - Objektmodellierung
  - Find and choose object classes
    - Attributes
  - Find associations
  - Inheritance

- Level of detail differs from design model to design model

- Entity Relationship Modeling
  - Model logical structure of data
  - Usually used for defining database requirements
Function-Oriented PA

- Hierarchy of functions are created
  - Leaf: Most detailed function
  - Root: Most abstract function

- Data-flow diagrams (DFDs) (Recap. SWT I)
- Data Dictionary (DD)
- Structured Requirements Definition (SRD)
- Structured Analysis and Design technique (SADT)
- Structured analysis and system specification (SASS)
Documentation of Requirements

- Requirements have to be documented
  - Used for planning
  - Basis for software development
  - Legal basis
  - Accessible by all stakeholders
Software Requirements Specification

- Complete description of the external behavior of the software
- Documentation of all interfaces between software and its environment

- Is what and not how
  - But: Design decisions taken by constraints can be included (e.g., platform)

- Can be textual, model based or even mixed
What should be included

- *Complete yet concise description of external interfaces of the system with its environment*
  - Other software
  - Communication ports
  - Hardware
  - Human users

⇒ Functional requirements
⇒ Nonfunctional requirements
What should not be included

- Project requirements
  - Staffing, Schedules
  - Costs
  - Milestones, activities, phases, etc.

- Product assurance plans
  - Configuration management plans
  - Verification and validation plans
  - Test plans, Quality assurance plans

- Software Designs
Quality Criteria for SRS

- Correct
- Unambiguous
- Complete
- Verifiable
- Consistent
- Understandable by customer

- Modifiable
- Traced
- Traceable
- Design independent
- Annotated
- Concise
- Organized
Requirements Document

- Often based on prosaic text
  - Can lead to ambiguities
    - Specify requirement more formal
  - Can include references to more formal models
    - E.g. includes UML diagrams as figures

- Usage of wordprocessing software
  - MS Word most used requirements software!
Requirements Document Structure

- Use a standardised document template
  - Easy to learn
  - Faster elicitation
  - Selective reading
  - Automatic checking
  - Reuse of requirements

- Template needs project specific adaptations

- Examples: Rational Unified Process (RUP), IEEE 830, V-Modell (Pflichten- und Lastenheft, SWT I), etc…
Basic Structure

- Introduction
  - Purpose, System Scope, Stakeholders, Definitions, Acronyms, Abbreviations, References, Overview

- Overall Description
  - System environment, architectural description, system functionality, users and audience, constraints, assumptions

- Requirements
  - Functional, quality, …

- Appendix

- Index
Example – RUP Requirements Document

1. Introduction
   1.1 Purpose
   1.2 Scope
   1.3 Definitions, Acronyms and Abbreviations
   1.4 References
   1.5 Overview

2. Overall Description

3. Specific Requirements
   3.1 Functionality
      3.1.1 <Functional Requirement One>
   3.2 Usability
      3.2.1 <Usability Requirement One>
   3.3 Reliability
      3.3.1 <Reliability Requirement One>
   3.4 Performance
      3.4.1 <Performance Requirement One>
   3.5 Supportability
      3.5.1 <Supportability Requirement One>
   3.6 Design Constraints
      3.6.1 <Design Constraint One>

3.7 Online User Documentation and Help System Requirements
3.8 Purchased Components
3.9 Interfaces
   3.9.1 User Interfaces
   3.9.2 Hardware Interfaces
   3.9.3 Software Interfaces
   3.9.4 Communications Interfaces
3.10 Licensing Requirements
3.11 Legal, Copyright and Other Notices
3.12 Applicable Standards

4. Supporting Information
Example - IEEE 830-1998

Table of Contents
- Introduction
  - Purpose
  - Scope
  - Definitions
  - References
  - Overview
- General Description
  - Product Perspective
  - Product Function
  - User Characteristics
  - General Constraints
  - Assumptions and Dependencies
- Specific Requirements
  - Functional Requirements
  - External Interface Requirements
  - Performance Requirements
  - Design Constraints
  - Attributes
  - Other Requirements
- Appendix
- Index
Writing Requirements

- Short sentences, one requirement per sentence
  - BAD: The Navigation systems navigates to a destination with comfortable usability.
  - GOOD: The navigation system allows all time and in all modes to set the destination

- Use terms conforming to the glossary
  - Glossary is maintained in parallel

- Use sentence templates
  - The initiator of action – *The navigation system*
  - Condition for action – *in on road mode*
  - Action – *must show*
  - Constraints for action
  - Objects of action - *distance to destination*
  - Refinement / source of object
  - Object – *on display*
  - Refinement / destination of action

![Sentence Templates Diagram]

Prof. Ralf Reussner – Softwaretechnik II

Software Design and Quality Group
Institute for Program Structures and Data Organization
Model based specification

- Target
- Use Cases, Scenarios
- System requirements

- Image of reality
- Abstraction with requirements focus
- For a purpose
- Languages
  - Informal, semiformal, formal
Requirements Validation

- Validation of requirements artefacts (output)
  - Ambiguities, incompleteness, inconsistencies
- Validate context aspects (input)
  - Wrong or missing context information
- Validate RE process
  - Missing steps, stakeholders
- Find errors in requirements as early as possible
- Without formal model, only reviews and prototypes help!
- Simulation and verification when formal model is available.
Validation vs. Verification

- **Validation:**
  - Case based
  - Am I building the right system?

- **Verification**
  - Correctness (relation between two documents)
  - Am I building the system right?
Requirements Validation - Techniques

- Inspection, Reviews, Walkthroughs
  - Find errors manually
- Model Checking
  - Formal verification of used models (e.g. FSMs)
- Simulation
  - Simulate aspects of a system
- Prototyping
  - Oriented to design model
  - Stakeholders test selected scenarios in a prototype
- Creation of system test cases
Validation

YOUR USER REQUIREMENTS INCLUDE FOUR HUNDRED FEATURES.

DO YOU REALIZE THAT NO HUMAN WOULD BE ABLE TO USE A PRODUCT WITH THAT LEVEL OF COMPLEXITY?

GOOD POINT. I'D BETTER ADD "EASY TO USE" TO THE LIST.
Requirements Management

- Maintain requirements in a central repository
  - Usage of databases
  - Special requirements software
- Link requirements
  - With documents and dependent requirements
- Use extended attributes
  - Id, Name, Description, Version, Author, Source
  - Stability, Criticality, Priority
  - Status, Type, Links, Release, Infos, Effort
  => Traceability
Traceability of Requirements

- Two kinds of traceability: horizontal and vertical

- Horizontal: Traceability to other requirements

- Vertical upwards: Traceability to sources of requirements

- Vertical downwards: Traceability to design entities realising the requirement
Tooling

- Specialised software helps managing requirements
  - Used for automatic generation of SRSs
  - Support the RE process
    - Guide the requirements engineer
  - Automatic validity and conflict checks
  - Versioning

- Usually central repository on dedicated server
  - Multi-user access
  - Web-Frontend for customer access
    - High client license fees
IBM Rational DOORS

- Formely know as Telelogic DOORS
- Dedicated to requirements change management
  - Supports custom requirements processes / workflows
- Server based
- Reviewer web client
- Integrated with many other development tools
  - Rational Requirements Composer etc.
- Can generate requirements documents
IBM Requisite Pro

- Very similar to Doors
- Both are #1 and #2 on the market
- Offers Full web client

The Jazz Team Server offers seamless ALM with shared resources – less reliance on point-to-point integrations.
Requirements Composer
SoftWiki

- Research project for ‘agile‘ RE
- Uses WIKI approach for documentation of requirements
  - Central server, web-based interface
- Linking of requirements is based on an ontology
  - Links to documents and other requirements
  - Relations have defined semantic
  - Allows for logic reasoning
Literature

- Davis, A. M. (1993), Software Requirements: Objects, Functions and States, Prentice Hall PTR, New Jersey
- Pohl, K. (2007), Requirements Engineering: Grundlagen, Prinzipien, Techniken, dpunkt, Heidelberg
- Lamsweerde, A. V. (2001), Goal-oriented requirements engineering: A guided tour