Requirements Engineering
Software Engineering Theory

Kristian Sandahl
Department of Computer and Information Science
2014-09-03
Requirements

System Design
(Architecture, High-level Design)

Module Design
(Program Design, Detailed Design)

Implementation of Units (classes, procedures, functions)

Acceptance Test
(Release testing)

System Testing
(Integration testing of modules)

Module Testing
(Integration testing of units)

Unit testing

Validate Requirements, Verify Specification

Verify System Design

Verify Module Design

Verify Implementation

Maintenance

Project Management, Software Quality Assurance (SQA), Supporting Tools, Education
What is a software requirement?

• “Software requirements express the needs and constraints placed on a software product that contribute to the solution of some real-world problems.”

(Kotonya and Sommerville, 2000)

• Example (web-shop):
When the user enters type of T-shirt, the system shall show a palette of available colors.
How do you write a natural language requirement?

• To avoid misunderstandings, always use a complete sentence.

• A sentence expresses a complete thought and consists of a subject and a predicate.

• Use modal verbs: ”shall”, ”will”, and ”must”

• Don’t use: ”should”, ”would”, or ”might”

• “Timely review should be done as soon as possible, and shall not exceed two working weeks.”
  (TS/ISO 16949)
Functional requirements

- Describe the functions that the software is to execute.
- Can be tested by giving input and checking the output.
- Example: “The user shall be able to add an item to the shopping basket.”
- Think of the mathematical definition:
  - a function is a relation between a set of inputs and a set of permissible outputs with the property that each input is related to exactly one output.
Non-functional requirements

• Design constraints, limiting the solution space
  • Example: “The system shall be implemented in php.”

• Quality requirements, possible to measure
  • Example: “The minimum response time is 2.0 seconds”
Features

• A distinguishing characteristic of a system item (includes both functional and nonfunctional attributes such as performance and reusability).

(IEEE Std 829)

Higher level stuff, used in advertising:

“The system shall have an SMS delivery notification service.”
Organize the Software Requirements Specification

• Example from IEEE Std 830-1998

3.2 Functional requirements

3.2.1 Show colors

When the user enters type of T-shirt, the system shall show a palette of available colours.

3.2.2 Add to shopping basket

The user shall be able to add an item to the shopping basket.

3.3 Performance requirements

3.3.1 Response time

The minimum response time is 2.0 seconds.

3.4 Design constraints

The system shall be implemented in php.
If a more thorough description is needed

3.2.1 Show colors

3.2.1.1 Input variables
   Type of T-shirt

3.2.1.2 Output variables
   Set of color codes

3.2.1.3 Processing
   1. Query database for available colors of Type of T-shirt.
   2. Create a set of color codes.
   3. Send color codes to palette viewer
Alternative ways of organizing requirements

3.2 System features
3.2.1 SMS notification

3.2.1.1 Purpose of SMS notification

The system shall have an SMS delivery notification service so customers can collect their delivery as fast as possible.

3.2.1.2 Stimulus/response sequence

When the delivery is has arrived to the pick-up point, notify the user.

3.2.1.3 Associated functional requirements

3.2.1.3.1 Number format

The user phone number shall be pretty-printed in the format: 07x – xxx xx xx

3.2.1.3.2 Change number

.....
1 Introduction
   1.1 Purpose
   1.2 Scope
   1.3 Definitions, acronyms and abbreviations
   1.4 References
   1.5 Overview

2 Overall description
   2.1 Product perspective
   2.2 Product functions
   2.3 User characteristics
   2.4 General constraints
   2.5 Assumptions and dependencies
   2.6 Lower ambition levels

3 Specific requirements
   3.1 Interface requirements
       3.1.1 User interfaces
       3.1.2 Hardware interfaces
       3.1.3 Software interfaces
       3.1.4 Communication interfaces
   3.2 Functional requirements
   3.3 Performance requirements
   3.4 Design constraints
   3.5 Software system attributes
   3.6 Other requirements

4 Supporting information
   4.1 Index
   4.2 Appendices
Pros and cons of IEEE 830

Pros

• Good checklist
• Many ways to adapt organization
• Many ways to detail requirements
• You don’t need to have everything in

Cons

• Takes some time to read and understand
• Hard to represent variation
• Is no guarantee for a good SRS
Requirements specification

Requirements in a good SRS are:

- Numbered
- Inspected
- Prioritised
- Unambiguous
- Testable
- Complete
- Consistent

- Traceable
- Feasible
- Modifiable
- Useful for:
  - operation
  - maintenance
  - customer
  - developer
  - ....

Who is the reader?
Iterative process

Collect user requirement
Elicitation

Check that it matches user/customer requirements
Validattion

Analysis

Speci-fication

Document/build

Understand
Elicitation

Purpose:

• Understand the true needs of the customer
• Trace future implementation to needs

Sources:

• Goals
• Domain knowledge
• Stakeholders
• Environment

Techniques:

• Interviews
• Scenarios
• Prototypes
• Facilitated meetings
• Observation
Interviews

Process:
- Start
- Q & A
- Summary teach-back
- Thank you!
- What’s next

Kinds:
- Structured
- Unstructured

Tips:
- Be 2 interviewers – shift roles
- Plan the interview
- Don’t stick to the plan – use feelings
- Let the customer talk
- Prepare ice-breakers
- Probe thinking
- Look for body language
- Think of human bias
- Why do you get the answers you get?
“If I’d asked my customers what they wanted, they’d have said a faster horse.”

Henry Ford
Requirements analysis

Goals

- Detect and resolve conflicts between requirements
- Discover bounds of software
- Define interaction with the environment
- Elaborate high-level requirements to derive detailed requirements
- Classify requirements for more effective management

Means:

- Often accomplished with conceptual modelling
Requirements classification

- Functional vs non-functional requirements
- Source
- Product or process requirements
- Priority
- Scope in terms of affected components
- Volatility vs stability
Requirements Abstraction Model

Organizational Strategies

Product Strategies

RAM - Abstraction Levels

- Product Level (goal)
- Feature Level (features)
- Function Level (functions/actions)
- Component Level (details - consists of)

Requirements

RAM - Action Steps

Specify (elicit)

Place (evaluate)

Abstraction (work-up)

Utilizing several levels of abstraction

A structured way of working

Source: Tony Gorschek, BTH
Up to product level and down to function level
Conceptual Modelling

• Representation in semi-formal notation
• Often diagrammatic representation
• Examples:
  • Object-orientation, use-cases, state-machines
  • Activity diagrams
  • Data flow diagrams
  • Entity-relationship models
A use-case is:

“... a particular form or pattern or exemplar of usage, a scenario that begins with some user of the system initiating some transaction of sequence of interrelated events.”

Jacobson, m fl 1992: Object-oriented software engineering. Addison-Wesley
Actor: a user of the system in a particular role. Can be human or system.

**CoffeeDrinker**

Use-case name (verb phrase)

Description of use-case

Buy a cup of coffee

A CoffeeDrinker approaches the machine with his cup and a coin of SEK 5. He places the cup on the shelf just under the pipe. He then inserts the coin, and presses the button for coffee to get coffee according to default settings. Optionally he might use other buttons to adjust the strength and decide to add sugar and/or whitener. The machine processes the coffee and bell when it is ready. The CoffeeDrinker takes his cup from the shelf.
Use-case diagram for the coffee-machine

Subject boundary

CoffeDrinker

TeaDrinker

Subject

CoffeeMachine

Buy a cup of coffee
Get coin in return
Pour hot water

Clean the Machine
Add substances
Collect coins
Brew a can of coffee

Subject name

Service

Porter

25
Relations between use-cases

- Extend loan
- Borrow copy of book
- Check for reservation
- Refuse loan

Stereotype: extended classification of meaning

”Separating scenarios”

”Reuse”
Identifying classes: noun analysis

A CoffeeDrinker approaches the machine with his cup and a coin of SEK 5. He places the cup on the shelf just under the pipe. He then inserts the coin, and press the button for coffee to get coffee according to default settings. Optionally he might use other buttons to adjust the strength and decide to add sugar and/or whitener. The machine processes the coffee and bell when it is ready. The CoffeeDrinker takes his cup from the shelf.
The coffee machine class model

- **CoffeeCustomer**
  - *buys* 0..1 *CoffeeCustomer*
  - *buys* 0..* *CanOfCoffee*

- **CupOfCoffee**
  - *makes* 0..* *CupOfCoffee*

- **CanOfCoffee**
  - *makes* 0..* *CanOfCoffee*

- **Porter**
  - *buys* 0..1 *Porter*

- **CoinHandler**
  - 1

- **Brewer**
  - 1

- **Interface**
  - 1

- *Machine*
  - 1
Data model: ER-diagram

```
Student
  Name
  Personal number
  Curriculum

Enrolled-in

Course
  Subject
  Course code
  Max-enrolment
```
There is no perfect specification, but you can write a good one
The RS, or SRS avoids many misunderstandings
The RS is of special importance in outsourcing programming

**Specification you already know**

- Carol, the customer
- Robert, the requirements engineer
- SRS
User stories – lightweight alternative

As a (role) I want (something) so that (benefit)

As a student I want to buy a parking card so that I can drive the car to school.

Priority: 3
Estimate: 4

Good for:
- Smaller (sub-)projects
- Frequent releases
- User feed-back
- Prototyping
- Elicitation
- Functional requirements

See more at:
http://www.agilemodeling.com/artifacts/userStory.htm
Means of validation

- Prototyping
- Simulation
- Software Reviews
- Model checking
- Formal proofs
- Acceptance testing
The role of requirements in the life-cycle

fuzziness

carol
the customer

elicitation

specification

modelling

formalisation

diana
the developer

tim
the tester

time

liu expanding reality
Quality factors

- Correctness
- Reliability
- Efficiency
- Usability
- Integrity
- Maintainability
- Flexibility
- Testability
- Security

- Portability
- Reusability
- Interoperability
- Survivability
- Safety
- Manageability
- Supportability
- Replaceability
- Functionality

Price?
Usability and Security

AND STARTING TODAY, ALL PASSWORDS MUST CONTAIN LETTERS, NUMBERS, DOODLES, SIGN LANGUAGE AND SQUIRREL NOISES.
Evaluate goals of

- Relevance
- Efficiency
- Attitude
- Learnability

- **Usability metrics**
Reliability

• The probability that the software executes with no failures during a specified time interval

• Approximation: MTTF/(1+MTTF)

• Example

• Easier to manage: Failure intensity, [failures / hours of execution time]

• Another approximation: $\lambda = (1-R)/t$

• Example
## Failure intensity guideline

<table>
<thead>
<tr>
<th>Impact</th>
<th>Failure intensity</th>
<th>Time btwn failures</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hundreds of deaths, $10^9$ cost</td>
<td>$10^{-9}$</td>
<td>114 000 years</td>
</tr>
<tr>
<td>1-2 deaths, $10^6$ cost</td>
<td>$10^{-6}$</td>
<td>114 years</td>
</tr>
<tr>
<td>$1000$ cost</td>
<td>$10^{-3}$</td>
<td>6 weeks</td>
</tr>
<tr>
<td>$100$ cost</td>
<td>$10^{-2}$</td>
<td>100 h</td>
</tr>
<tr>
<td>$10$ cost</td>
<td>$10^{-1}$</td>
<td>10 h</td>
</tr>
<tr>
<td>$1$ cost</td>
<td>1</td>
<td>1 h</td>
</tr>
</tbody>
</table>
Why do we need requirements?

Useful for:
- Design
- Test
- Support
- Planning
- Maintenance

Communication to stakeholders

Make trade-offs

Resolve conflicts

Finding true needs

Contract document
Things to keep an eye on

- Human bias:
- We tend to be more detailed in areas where we already have good knowledge.
- Deleted requirements
- The more we invest in working with requirements, the more we lose when they are deleted

Benefit of detail

$P(\text{change})$