MTAT.03.094
Software Engineering

Lecture 02:
Requirements Engineering

Marlon Dumas
(in lieu of Dietmar Pfahl)

email: marlon.dumas@ut.ee

Fall 2015
Lab 1 Task (Homework)

Joostes Marss AS

Customer: Home Improvement International (HII)

POS System

Requirements:
- Interviews with at least 4 user roles (Protocols)
- List of user roles
- List of requirements (50 functional / 5 non-functional (SMART))
Schedule of Lectures

Week 01: Introduction to SE

Week 02: Requirements Engineering

Week 03: Req. Spec. & System Analysis

Week 04: No lecture

Week 05: Development Infrastructure I

Week 06: Development Infrastructure II

Week 07: Architecture and Design

Week 08: No lecture

Week 09: Refactoring

Week 10: Verification and Validation I

Week 11: Verification and Validation II

Week 12: Agile/Lean Methods

Week 13: Software Quality Mgt.

Week 14: No lecture

Week 15: Measurement / Course wrap-up, review and exam preparation

Week 16: No lecture
Goal of this Lecture: To give answers to the following questions …

What is ‘Requirements Engineering’?

Why is RE important?

Why is RE difficult?

Who is involved in RE?

What are ‘Requirements’?

What types of requirements exist?

What process steps are involved in RE?

How to get started with RE?

How to elicit requirements?

How to represent/document requirements?

How to use requirements for project planning?

next week
Definition: Requirements Engineering

RE is the process of establishing

• what the customer requires from a system

and

• the constraints under the software operates and is developed.

RE means to ...

... dig up, understand, write down, check, prioritize, select, follow up on ...

... the **functions and properties of (software) products**
Why RE?
Economic Consequences of RE Defects

![Bar chart showing the cost of correcting a problem at different stages: Req, Design, Coding, System Testing, Acceptance Testing, Operation. The cost is highest during Operation.]

[Davis, 1992]
RE is difficult, because …

- It typically involves many stakeholders.
- Stakeholders (often) don’t know what they really want.
- Stakeholders express requirements in their own terms (might be imprecise, ambiguous).
- Different stakeholders may have conflicting requirements.
- Organisational and political factors may influence the system requirements.
- New stakeholders may emerge and the business environment change.
- The requirements change during the analysis process.
Example: Ambiguous Requirement

The tiny word ’only’

Version 1:
• The spam filter **only delivers** the e-mail that the user wants.

Version 2:
• The spam filter **delivers only** the e-mail that the user wants.

Same meaning? If not, what’s the difference?
Example: Ambiguous Requirement

The tiny word ‘only’

Version 1:
• The spam filter **only delivers** the e-mail that the user wants.

Version 2:
• The spam filter **delivers only** the e-mail that the user wants.

*(Unfortunately, Version 2 is not considered correct English in the UK. The word ‘only’ must always go before the main verb.)*
Example: Ambiguous Requirement

- Requirement: ‘A user of the Library Information System (LIS) shall be able to search the recent publications lists for all libraries.’

- Consider the term ‘search for all’:
  - User intention: search for a publication across all recent publications lists in all libraries;
  - Developer interpretation: search for a publication in an individual recent publications list. User first chooses library then searches list.

Imprecise (ambiguous) requirements may be interpreted in different ways by developers and users.
Example: Ambiguous Requirement

- Requirement: ‘A user of the Library Information System (LIS) shall be able to search the recent publications lists for all libraries.’

- Consider the term ‘search … for all’:
Example: Ambiguous Requirement

- Requirement: ‘A user of the Library Information System (LIS) shall be able to search the recent publications lists for all libraries.’
- Consider the term ‘search … for all’:
  - User intention: search for a publication across all recent publications lists in all libraries;
  - Developer interpretation: search for a publication in an individual recent publications list. User first chooses library then searches list.
- Imprecise (ambiguous) requirements may be interpreted in different ways by developers and users.
Example: Ambiguous Requirement

- Requirement: ‘A user of the Library Information System (LIS) shall be able to search the recent publications lists for all libraries.’

- Consider the term ‘search … for all’:
Example: Ambiguous Requirement

• Requirement: ‘A user of the Library Information System (LIS) shall be able to search the recent publications lists for all libraries.’

• Consider the term ‘search … for all’:
  • User intention: search for a publication across all recent publications lists in all libraries;
  • Developer interpretation: search for a publication in an individual recent publications list. User first chooses library then searches list.

• Imprecise (ambiguous) requirements may be interpreted in different ways by developers and users.
Example: Conflicting Requirements

• A performance requirement may indicate that
  • a core system must be updated in real time
  but
  • the size and scope of the system (as defined by other requirements) may preclude this.

Updating such a large system may not be possible in real time.

• Need to apply conflict resolution procedures (→ negotiation with stakeholders)
SMART Requirements

- Specific
- Measurable
- Attainable (Achievable, Actionable, Appropriate)
- Realistic
- Time-bound (Timely, Traceable)

Source: http://jessica80304.wordpress.com/2008/08/04/smart-requirements/
SMART Requirements

Counter-example (i.e., not SMART):

’The user interface of system xyz should look nice to all users and the response time to inquiries should be as fast as the speed of light’

<table>
<thead>
<tr>
<th>S</th>
<th>Specific</th>
</tr>
</thead>
<tbody>
<tr>
<td>M</td>
<td>Measurable</td>
</tr>
<tr>
<td>A</td>
<td>Attainable (Achievable, Actionable, Appropriate)</td>
</tr>
<tr>
<td>R</td>
<td>Realistic</td>
</tr>
<tr>
<td>T</td>
<td>Time-bound (Timely, Traceable)</td>
</tr>
</tbody>
</table>
SMART Requirements

- **Specific**
  - A good requirement is specific and not generic. It should not be open to misinterpretation when read by others.
  - Avoid using conjunctions (and, or, but)
  - Avoid indeterminate amounts of time (soon, fast, later, immediately)
  - Etc.

- **Measurable**
- **Attainable** (Achievable, Actionable, Appropriate)
- **Realistic**
- **Time-bound** (Timely, Traceable)

Source: http://jessica80304.wordpress.com/2008/08/04/smart-requirements/
SMART Requirements

- **Specific**
  This answers whether you will be able to verify the completion of the project. You should avoid signing up for any requirement that cannot be verified as complete.
  - These are especially risky when you use non-quantitative terms (best, optimal, fastest) for acceptance criteria.

- **Measurable**
- **Attainable** (Achievable, Actionable, Appropriate)
- **Realistic**
- **Time-bound** (Timely, Traceable)

Source: http://jessica80304.wordpress.com/2008/08/04/smart-requirements/
SMART Requirements

- **Specific**
  - This intends to ensure that the requirement is physically and logically possible to be achieved given existing circumstances. There is arguably overlap between attainable and realistic.
  - Reserve attainable to check the likelihood that it will be possible to achieve the requirement

- **Measurable**
- **Attainable** (Achievable, Actionable, Appropriate)
- **Realistic**
- **Time-bound** (Timely, Traceable)

Source: http://jessica80304.wordpress.com/2008/08/04/smart-requirements/
SMART Requirements

- Specific
- Measurable
- Attainable (Achievable, Actionable, Appropriate)
- Realistic
- Time-bound (Timely, Traceable)

This intends to ensure that the requirement is realistic to deliver when considering other constraints of the project and requirements.

Source: http://jessica80304.wordpress.com/2008/08/04/smart-requirements/
SMART Requirements

• Specific
• Measurable
• Attainable (Achievable, Actionable, Appropriate)

Where appropriate each requirement should be time-bound or specify by when or how fast a required function needs to be completed or executed.
In software engineering, you may see the “T” in SMART being used to mark whether a requirement is “traceable”, which is a separate but important topic in developing software.

• Realistic

• Time-bound (Timely, Traceable)

Source: http://jessica80304.wordpress.com/2008/08/04/smart-requirements/
Types of Requirements

- User versus System Requirements
The Goal of RE

- What the Customer wants
- What the Customer needs
- What the Software does

Application Domain (User Requirements)

System Domain (System Requirements)
User vs. System Requirements

User requirements

- Statements in natural language plus diagrams of the services the system provides and its operational constraints.
- Written for customers.

System requirements

- A structured document setting out detailed descriptions of the system’s functions, services and operational constraints.
- Defines what should be implemented and thus may be part of a contract between client and contractor.
Types of Requirements

- User versus System Requirements
- Functional versus Non-Functional (Quality) Requirements
Functional vs. Non-Functional Requirements

Functional reqs:

- What the system shall do
- Often intended to be implemented as a whole or else not implemented at all
- Often regards input/output data and functions that process the input data to produce the output

Non-functional reqs (NFR), Quality Requirements, (extra-functional reqs):

- How good the system shall do it
- Often measured on a scale
- Often put constraints on the system (or the development process)
- Often cross-cutting, may impact many functions

Performance  
Reliability  
Usability  
Safety, Security  
Interoperability  
Maintainability  
...

But the division is not black and white...
What non-functional (=quality) requirements do you expect from a taxi ride booking application?
SW Product Quality -> ISO 9126 (now ISO 25000)

Software Quality Model
(ISO 9126 – Part 1)

Does the software support all the required functions?

How easy is it to transfer the software to another environment?

How efficiently does the software perform?

How reliable is the software?

How easy is the software to use?

How easy is the software to modify?

Efficiency = Performance
SW Product Quality -> ISO 9126 (cont’d)

External and Internal Measures (ISO 9126 – Parts 2 & 3)

- functionality
  - suitability
  - accuracy
  - interoperability
  - security
  - functionality compliance
- reliability
  - maturity
  - fault tolerance
  - recoverability
  - reliability compliance
- usability
  - understandability
  - learnability
  - operability
  - attractiveness
  - usability compliance
- efficiency
  - time behaviour
  - resource utilization
  - efficiency compliance
- maintainability
  - analysability
  - changeability
  - stability
  - testability
  - maintainability compliance
- portability
  - adaptability
  - installability
  - co-existence
  - replaceability
  - portability compliance

Example:
Maturity – internal: actual defect detection/correction during development, test adequacy, ...
Maturity – external: freedom of the SW system from failures caused by defects in the SW
**Example – Efficiency Requirements**

**Performance requirements:**

R1: Product shall be able to process 100 payment transactions per second in peak load.

R2: Product shall be able to process one alarm in 1 second, 1000 alarms in 5 seconds.

R3: In standard work load, CPU usage shall be less than 50% leaving 50% for background jobs.

R4: Scrolling one page up or down in a 200 page document shall take at most 1 s. Searching for a specific keyword shall take at most 5 s.

R5: When moving to the next field, typing must be possible within 0.2 s. When switching to the next screen, typing must be possible within 1.3 s. Showing simple report screens, less than 20 s. (Valid for 95% of the cases in standard load)

R6: A simple report shall take less than 20 s for 95% of the cases. None shall take above 80s. (UNREALISTIC)

From: Soren Løsesen: Software Requirements
© Pearson / Addison-Wesley 2002
## Example – Usability Requirements

<table>
<thead>
<tr>
<th>Problem counts</th>
<th>Design-level reqs</th>
</tr>
</thead>
<tbody>
<tr>
<td>R1: At most 1 of 5 novices shall encounter critical problems during tasks Q and R. At most 5 medium problems on list.</td>
<td>R6: System shall use screen pictures in app. xx, buttons work as app. yy.</td>
</tr>
<tr>
<td><strong>Task time</strong></td>
<td><strong>Product-level reqs</strong></td>
</tr>
<tr>
<td>R2: Novice users shall perform tasks Q and R in 15 minutes. Experienced users tasks Q, R, S in 2 minutes.</td>
<td>R7: For all code fields, user shall be able to select value from drop-down list.</td>
</tr>
<tr>
<td><strong>Keystroke counts</strong></td>
<td><strong>Guideline adherence</strong></td>
</tr>
<tr>
<td>R3: Recording breakfast shall be possible. No mouse.</td>
<td>R8: System shall follow style guide zz. Menus shall have at most three levels.</td>
</tr>
<tr>
<td><strong>Opinion poll</strong></td>
<td><strong>Development process reqs</strong></td>
</tr>
<tr>
<td>R4: 80% of users shall find system easy to recommend system to others.</td>
<td>R9: Three prototype versions shall be made and usability tested during design.</td>
</tr>
<tr>
<td><strong>Score for understanding</strong></td>
<td></td>
</tr>
</tbody>
</table>
RE Activities

Requirements elicitation
- Interacting with stakeholders to discover their requirements:
  - What is to be accomplished?
  - How the system will fit into the needs of the business?
  - How the system will be used on a day-to-day basis?

Requirements analysis
- Refining, classifying/clustering, structuring, prioritizing, and modifying the gathered requirements

Requirements specification
- Documenting the (system) requirements in a semiformal or formal manner to ensure clarity, consistency, and completeness next week

Requirements validation
- Checking the requirements
Requirements Elicitation
Where do we start?

**Identify the problem**
- what is the objective of the project?
- the “vision” of those who are pushing for it?
  - e.g., “Meeting scheduling is too costly right now”

**Scope the problem**
- given the vision, how much do we tackle?
  - e.g. “Build a system that schedules meetings”, …or…
  - e.g. “Build a system that maintains people’s calendars” …or…

**Identify solution scenarios**
- given the problem, what is the appropriate business process for solving it?
  - e.g. “Anyone who wants to schedule a meeting goes to the secretary, gives details and the secretary handles the rest”, …or…

**Scope the solution**
- Given a business process, what parts should be automated, and how?
  - e.g. “Computer takes in scheduling request details, outputs a solution” …or…
  - e.g. “Solution arrived at interactively by secretary and computer” …or…
Difficulties of Elicitation

- Implicit (tacit) knowledge
- Thin spread of domain knowledge
- Conflicting information
- Say-do problem
- Probe (Hawthorne) effect
- Bias
Example: Elicit the rules and procedures for approving a loan

Why this might be difficult?

- **Implicit knowledge:**
  - There is no document in which the rules for approving loans are written down

- **Conflicting information:**
  - Different bank staff have different ideas about what the rules are

- **Say-do problem:**
  - The loan approval process described to you by the loan approval officers is quite different from your observations of what they actually do

- **Probe effect:**
  - The loan approval process used by the officers while you are observing is different from the one they normally use

- **Bias:**
  - The loan approval officers fear that your job is to computerize their jobs out of existence, so they are deliberately emphasizing the need for case-by-case discretion (to convince you it has to be done by a human!)
Elicitation Techniques

- Analyzing existing documents & data
- “Brainstorming” possible requirements
  - within the dev. Team
- Interviews (one-on-one)
  - This is what you need to do for Lab 1…
- Focus groups or workshops (one-on-many)
  - useful for larger projects
- Prototyping/mockups
- Meetings with the customer/users
  - E.g. for checkpoints, or showing prototypes
Analyzing documents & data

Sources of information:

- company reports, organization charts, policy manuals, job descriptions, reports, documentation of existing systems, etc.

Advantages:

- Helps the analyst to get an understanding of the organization before meeting the people who work there
- Helps to prepare for other types of fact finding
  - e.g. by being aware of the business objectives of the organization.
- may provide detailed requirements for the current system

Disadvantages:

- written documents often do not match up to reality
- Can be long-winded with much irrelevant detail
Interviews

Types:
- Structured - agenda of fairly open questions
- Open-ended - no pre-set agenda

Advantages
- Rich collection of information
- Good for uncovering opinions, feelings, goals, as well as hard facts
- Can probe in depth, & adapt follow-up questions to what the person tells you

Disadvantages
- Large amount of qualitative data can be hard to analyze
- Interviewing is a difficult skill to master

Source: Adapted from Goguen and Linde, 1993, p154.
Interview structure

Investigate the “problem”/”opportunity”

- What (Which) problem needs to be solved?
  - identify problem Boundaries
- What might prevent us solving it?
  - identify Feasibility and Risk
- Where is the problem?
  - understand the Context/Problem Domain
- Whose problem is it? Who is affected?
  - identify Stakeholders
- Why does it need solving?
  - identify the stakeholders’ Goals
- When does it need solving?
  - identify Development Constraints
- How does the problem manifest itself?
  - collect some Scenarios
Meetings

- Used for summarization and feedback
  - E.g. meet with stakeholders towards the end of each stage:

- Every meeting should have a clear objective:
  - E.g. presentation, problem solving, conflict resolution, progress analysis, gathering and merging of facts, training, planning,...

- Plan the meeting carefully:
  - Schedule the meeting and arrange for facilities
  - Prepare an agenda and distribute it well in advance
  - Keep track of time and agenda during the meeting
  - Follow up with a summary to be distributed to meeting participants
Prototyping (a.k.a. mockups)

- Paper prototyping
- Wireframes
- Interactive wireframes
- Rich interactive prototypes
  - e.g. Concept.ly

*People often don’t know what they want until they see what they can get.*

A. Wildavsky
Combine Different Techniques

- Background reading (e.g., Internet?)
- (Initial) Meeting
- Hard Data analysis
- Brainstorming
- Interviews
- Prototyping
- Meeting

...
Requirements Specification – Styles

- Natural language (plus supporting tables and graphs)
- Structured natural language / Scenarios
  - e.g., user, stories, use case descriptions, CRC cards, ...
- Semi-formal notations
  - e.g., UML diagrams (use case diagrams, class diagrams, state charts, sequence charts, etc.)
- Formal notations (with formal semantics)
  - e.g., abstract model-based (VDM, Larch, B, ...) or algebraic (OBJ, UML, B, ...)
Example: Home Access Control

Objective:
Design an electronic system for:

- Home access control
  - Locks and lighting operation
- Intrusion detection and warning

[Diagram of system components: Lock, Photosensor, Switch, Alarm bell, Light bulb]
Example – More Details

Please read:
Ch 1.2 / 1.3.1 / 2.2
Next Lecture

- Topic:
  - Requirements Specification & Domain Models
Requirements Specification

Good & less good examples from “Software Project”

- https://github.com/vladislavivanistsev/SuperCap/
- https://github.com/L6mps/LasaLaraAP/
- https://github.com/alajal/license-management/
- https://github.com/TaaviGilden/Prawesome/
Next Lecture

- **Topic:**
  - Requirements Specification & Domain Models

- **For you to do:**
  - Finalize team forming
  - Familiarise with your team members and homework task 1 & readings
  - **Get started with lab task 1!**
  - **Go to labs next week:** consulting about task 1
Acknowledgements

Textbooks:


• Sören Lauesen: Software Requirements - Styles and Techniques, Pearson Education, 2002

Lectures on RE:

• Prof. Björn Regnell, Lund University, Sweden

• Prof. Steve Easterbrook, University of Toronto, Canada