Software Engineering

Lecture 02:
Requirements Engineering – Part 1

Fall 2021

Dietmar Pfahl
email: dietmar.pfahl@ut.ee
Schedule of Lectures

Week 01: Introduction to SE
Week 02: Requirements Engineering I
Week 03: Requirements Engineering II
Week 04: Analysis
Week 05: Development Infrastructure
Week 06: Continuous Development and Integration
Week 07: Project Estimation / Architecture and Design I
Week 08: Architecture and Design II
Week 09: Verification and Validation I
Week 10: Verification and Validation II
Week 11: Refactoring (and TDD)
Week 12: Agile/Lean Methods
Week 13: Industry Guest Lecture
Week 14: Course wrap-up, review and exam preparation
Week 15: Reserve time slot (no lecture scheduled as of today)
Goal of this Lecture:
To give answers to the following questions …

<table>
<thead>
<tr>
<th>Questions</th>
<th>Relevant Activities</th>
</tr>
</thead>
<tbody>
<tr>
<td>What is ‘Requirements Engineering’?</td>
<td>What activities are involved in RE?</td>
</tr>
<tr>
<td>Why is RE important?</td>
<td>How to elicit requirements?</td>
</tr>
<tr>
<td>Why is RE difficult?</td>
<td>How to represent/document requirements?</td>
</tr>
<tr>
<td>Who is involved in RE?</td>
<td>How to use requirements for project planning?</td>
</tr>
<tr>
<td>What are ‘Requirements’?</td>
<td></td>
</tr>
<tr>
<td>What types of requirements exist?</td>
<td></td>
</tr>
<tr>
<td>What levels of requirements exist?</td>
<td></td>
</tr>
</tbody>
</table>
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 levels of requirements exist?
What activities are involved in RE?
How to elicit requirements?
How to represent/document requirements?
How to use requirements for project planning?

to be continued next week
Structure of Lecture 02

• Recap from last week: User Stories
• The Nature of Requirements Engineering (RE)
• The Nature of Requirements
• The Process of RE
Comparison of Basic Process Types

- Waterfall
  - Requirements
  - Design
  - Implementation
  - Testing

- Incremental, e.g. RUP

- Agile - XP

RUP = Rational Unified Process
XP = Extreme Programming
Lab 1 Assignment

Joostes Marss AS

Customer: Home Improvement International (HII)

Team

POS System

To do:
- Interviews to identify roles and requirements (user stories)
- List of 20 functional user stories
- List of 10 non-functional user stories (performance & usability)
Lab 1 Assignment

Joostes Marss AS

Customer: Home Improvement International (HII)

POS System

Team

To do:
- Interviews to identify roles and requirements (user stories)
- List of 20 functional user stories
- List of 10 non-functional user stories (performance & usability)

Have a look at the videos on the course wiki:
→ YouTube videos by Lars Bilde (Kaj Car Rental Service)
User Stories

As a tenant | I can lock the doors | to protect my apartment.

role (benefactor)  capability (action/means)  business-value (motivation/rationale/end)

**who** - **what** - **why**

- Similar to NL requirements, but focus on the user benefits, instead on system characteristics (alone).
- Unfortunately, third element (business-value) is often ommitted
- Preferred technique in **agile methods**.
User Story:
As a tenant | I can lock the doors | to protect my apartment.

- role (benefactor)
- capability (action/means)
- business-value (motivation/rationale/end)

who - what - why

- Similar to NL requirements, but focus on the user benefits, instead on system characteristics (alone).
- Unfortunately, third element (business-value) is often omitted
- Preferred technique in agile methods.

… to make sure my child (or my pet) cannot get out unexpectedly
… to make sure I don’t forget my keys when I leave again
‘Normal’ User Story

<Actor/Role> As a tenant

... I can lock the doors

... to protect my apartment (from burglars)

A good User Story is:
- Independent
- Negotiable
- Valuable
- Estimable
- Small
- Testable

INVEST
‘Normal’ User Story – Independent

<Actor/Role> As a tenant

I can lock the doors

... to protect my apartment (from burglars)

INDEPENDENT:
- Does not overlap with or subsume concepts occurring in other user stories
- Can be scheduled and implemented in any order
‘Normal’ User Story – Negotiable

<Actor/Role>  As a tenant

... 

<Action>  I can lock the doors 

... 

<Value>  to protect my 

apartment (from 

burglars)

NEGOTIABLE: 
- It is not an explicit contract for features; rather, details will be co-created by the customer and programmer during development.

Note: A good user story captures the essence, not the details.
‘Normal’ User Story – Valuable

<Actor/Role> As a tenant
...

<Action> I can lock the doors
...

<Value> to protect my apartment (from burglars)

VALUABLE:
- The user story is valuable to the customer

NOTE: We are mainly interested in value for the customer; if the role in a user story is not the customer, then we should still be able to explain it’s value for the customer
‘Normal’ User Story – Estimable

<Actor/Role> As a tenant

... I can lock the doors

... to protect my

<Value> apartment (from burglars)

Note: Being estimable is partly a function of
- being negotiable
- size (small is easier to estimate)
- team’s experience

ESTIMABLE
- No exact estimate needed, just enough to rank and schedule the user story’s implementation
‘Normal’ User Story – Small

<Actor/Role> As a tenant
...

<Action> I can lock the doors
...

<Value> to protect my apartment (from burglars)

Note:
- User stories should typically represent at most a few person-days worth of work
- Smaller user stories are easier to estimate
‘Normal’ User Story – Testable

<Actor/Role> As a tenant

... I can lock the doors

... to protect my apartment (from burglars)

TESTABLE:
- Writing a good user story carries an implicit promise: “I understand what I want well enough that I could write a test for it.”
‘Normal’ User Story – Testable

<Actor/Role> As a tenant

... I can lock the doors

... to protect my apartment (from burglars)

Acceptance test (1):

Given that the door has not been locked (state='unlocked' or 'open'):
Check whether the door can be locked with correct key.

Pass: door changes state to ‘locked’
Fail: door doesn’t change state to ‘locked’
‘Normal’ User Story – Testable

<Actor/Role> As a tenant

... 

<Action> I can lock the doors

... 

<Value> to protect my apartment (from burglars)

Acceptance test (2):

Given that the door has not been locked (state='unlocked' or 'open'):
Check whether the door can be locked {without key / with incorrect key}

Pass: door doesn’t change state to 'locked'
Fail: door changes state to 'locked'
‘Normal’ User Story – Testable

<Actor/Role> As a tenant

... I can lock the doors

... to protect my apartment (from burglars)

Acceptance test (3):
Given that the door has been locked (state='locked'):
Check whether the attempt to lock triggers the expected behaviour.

Pass: message 'door is already locked' is issued' and state unchanged.
Fail: door changes state | no message
Bad User Story Example /1

As a system user
I want to send and receive emails

--------------------------------------------------

As a system user
I want to send and reply to emails

to …

What is bad?
Bad User Story Example /1

<Actor/Role> As a system user
<Action> I want to send and receive emails
<Value> to …

---

<Actor/Role> As a system user
<Action> I want to send and reply to emails
<Value> to …

NOT independent:
US1 overlaps with US2

Better:
US1: … send [new] email …
US2: … receive emails …
US3: … reply to emails …
Bad User Story Example /2

<Actor/Role> As a fulfilment clerk
<Action> I want to get a note telling which items to send, from which shelf to pick the items, how to write the receipt, and where to deliver …
<Value> to …

What is bad?
Bad User Story Example /2

<Actor/Role> As a fulfilment clerk
<Action> I want to get a note telling which items to send, from which shelf to pick the items, how to write the receipt, and where to deliver …
<Value> to …

NOT negotiable:
US gives too much implementation detail.

Better to start with:
US: As a fulfilment clerk, I can send a book and receipt, to …
The details of how to do this (i.e., what and how to automate the process can be negotiated)
Bad User Story Example /3

<Actor/Role> As an end user
<Action> I want to see error messages in any language I want
<Value> --

What is bad?
Bad User Story Example /3

<Actor/Role> As an end user
<Action> I want to see error messages in any language I want
<Value> --

NOT valuable:
Value depends on what we try to achieve, e.g.:
- Increase revenue
- Avoid cost
- Improve service
- Meet requirements
- Build reputation
- Create options
- Generate data

… for our stakeholders
Bad User Story Example /4

<Actor/Role> As a fulfillment clerk
<Action> I want to send a book and receipt
<Value> to …

What is bad?
Bad User Story Example /4

<Actor/Role> As a fulfillment clerk
<Action> I want to send a book and receipt
<Value> to …

NOT estimable:

The user story is too vague and still needs to be negotiated and broken up into several more specific user stories.

Note: We often want to estimate a story before it is fully understood; ‘negotiable’ – ‘size’ – and ‘estimable’ are related and typically change over time.
Bad User Story Example /5

<Actor/Role>  As a system user
<Action>  I want to send emails to all my relatives
<Value>  to …

What is bad?
Bad User Story Example /5

<Actor/Role> As a system user
<Action> I want to send emails to all my relatives
<Value> to …

NOT testable:

The user story is contains elements of ambiguity and subjectivity which make it hard to test

Typical words that signal problems with testability are: all, always, just, only, any, right, suitable, appropriate, best, worst, fast, shortest, longest, …
Structure of Lecture 02

• Recap from last week: User Stories
• The Nature of Requirements Engineering (RE)
• The Nature of Requirements
• The Process of RE
Definition: Requirements Engineering

RE is the process of establishing

• the services that the customer requires from a system

and

• the constraints under which it 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
The Goal of RE

What the Customer wants

What the Customer needs

What the Software (to-be) does

Application Domain
(User Requirements)

System Domain
(System Requirements)
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.
Structure of Lecture 02

• Recap from last week: User Stories
• The Nature of Requirements Engineering (RE)
• The Nature of Requirements
• The Process of RE
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 Student Information System (SIS)?
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)

Cover all product functions?
Example – Usability Requirements

<table>
<thead>
<tr>
<th>Problem counts</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>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Task time</th>
</tr>
</thead>
<tbody>
<tr>
<td>R2: Novice users shall perform tasks Q and R in 15 minutes. Experienced users tasks Q, R, S in 2 minutes.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Keystroke counts</th>
</tr>
</thead>
<tbody>
<tr>
<td>R3: Recording breakfast shall be possible, No mouse.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Opinion poll</th>
</tr>
</thead>
<tbody>
<tr>
<td>R4: 80% of users shall find system easy to understand, recommend system to others.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Score for understanding</th>
</tr>
</thead>
<tbody>
<tr>
<td>R5: Show 5 users 10 common error messages, say cause. 80% of them say “large”. Ask for the cause. 80% of the time.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Design-level reqs</th>
</tr>
</thead>
<tbody>
<tr>
<td>R6: System shall use screen pictures in app. xx, buttons work as app. yy.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Product-level reqs</th>
</tr>
</thead>
<tbody>
<tr>
<td>R7: For all code fields, user shall be able to select value from drop-down list.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Guideline adherence</th>
</tr>
</thead>
<tbody>
<tr>
<td>R8: System shall follow style guide zz. Menus shall have at most three levels.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Development process reqs</th>
</tr>
</thead>
<tbody>
<tr>
<td>R9: Three prototype versions shall be made and usability tested during design.</td>
</tr>
</tbody>
</table>
Structure of Lecture 02

• Recap from last week: User Stories
• The Nature of Requirements Engineering (RE)
• The Nature of Requirements
• The Process of RE
RE Activities

Requirements gathering
(= 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

Requirements validation

• Checking the requirements
RE Activities: Iteration & Concurrency

Initial information
Scope
Constraints

Elicit

Analyze
classify, organize, prioritize, negotiate

Validate

Specify

Model

Requirements traced back to their source

+ Requirements Management
To be continued next week
Next Lecture

• Date/Time:
  • Friday, 17-Sep, 10:15-12:00

• Topic:
  • Requirements Engineering II
    • Representation styles
    • Planning based on requirements

• For you to do:
  • Get started with homework assignment 1!
  • Go to labs next week: assessment of intermediate results for homework assignment 1