Lecture 02: Requirements Engineering – Part 1

Dietmar Pfahl

email: dietmar.pfahl@ut.ee

Fall 2018
Project Teams & Exam Dates

THANK YOU for being very co-operative (and fast) with forming project teams!

Exam Dates:

• Exam 1: Tuesday, 08-Jan-2019 at 14:15-16:45, rooms J. Liivi 2-403/404/405 (limit: 100 students)
• Exam 2: Friday, 11-Jan-2019 at 14:15-16:45, room J. Liivi 2-403/404/405 (limit: 100 students)
• Re-take Exam: Monday, 28-Jan-2019 at 14:15-16:45, room J. Liivi 2-611
Schedule of Lectures (Tentative)

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

<table>
<thead>
<tr>
<th>What is ‘Requirements Engineering’?</th>
<th>What process steps are involved in RE?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Why is RE important?</td>
<td>How to get started with RE?</td>
</tr>
<tr>
<td>Why is RE difficult?</td>
<td>How to elicit requirements?</td>
</tr>
<tr>
<td>Who is involved in RE?</td>
<td>How to represent/document requirements?</td>
</tr>
<tr>
<td>What are ‘Requirements’?</td>
<td>How to use requirements for project planning?</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 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?

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

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

Joostes Marss AS

Customer: Home Improvement International (HII)

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

Team

Customer: Home Improvement International (HII)

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)

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.

- 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.
‘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

... 

<Action> I can lock the doors

... 

<Value> 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 its value for the customer
‘Normal’ User Story – Estimable

<Actor/Role> As a tenant

... 

<Action> I can lock the doors

... 

<Value> to protect my apartment (from burglars)

ESTIMABLE
- No exact estimate needed, just enough to rank and schedule the user story’s implementation

Note:
Being estimable is partly a function of
- being negotiable
- size (small is easier to estimate)
- team’s experience
‘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 typically should represent at most a few person-days worth of work
- Smaller user stories are easier to estimate

SMALL
- Good stories tend to be small
‘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
  ...

<Action>  I can lock the doors
  ...

<Value>  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
...

<Action> I can lock the doors
...

<Value> 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

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

What is bad?

<Actor/Role> As a system user
<Action> I want to send and reply to emails
<Value> to ...
Bad User Story Example /1

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

NOT independent:
US1 overlaps with US2
Better:
US1: … send [new] email ...
US2: … receive emails ...
US3: … reply to emails ...

<Actor/Role> As a system user
<Action> I want to send and reply to emails
<Value> to …
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 …
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 an fulfillment clerk
>Action>        I want to send a book
<Value>         and receipt
<Value>         to …

What is bad?
Bad User Story Example /4

<Actor/Role> As an 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)
Why is RE important?

Requirements (Specification) are central for development and verification / validation.
Economic Consequences of RE Problems

[Diagram showing cost comparison of different stages of software development (Req, Design, Coding, System Testing, Acceptance Testing, Operation) with a peak in 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**.
Stakeholders in SE

External:
- Customers
  - Those who pay for the software
- Users
  - Those who use the software

Internal:
- Software Developers
- Development (Project) Managers
- Product Managers
- ...
Example of a Real World Situation
(Mobile Phones)

Where do all requirements come from?

... find the right person to talk to ...
... get the deep domain knowledge ...

External stakeholders
Customers
- Direct customers
- Operators
  - Global customers
  - Regional customers
  - Other key customers
Retailers
Indirect customers
- Consumers
  - Market segments
Service providers
- Content providers
Product providers
- Direct Competitors
  - Mobile phone developers
- Indirect Competitors
  - Cameras
  - Mobile music players
  - ... consumer wallet competition
Platform providers
- Operating Systems
- Technical Platforms
Network system providers
- Standardization bodies
  - National
  - International
Manufacturing sub-contractors
Component providers

Internal stakeholders
Marketing
- Long term branding
  - Customer relations
Product management
- Product planning
  - Roadmapping and portfolios
Product development
- Hardware design
  - Electronics
    - Analog
    - Digital
  - Mechanics
Software design
- User interface
  - Service logic
Network access
- Codecs
Platform development
- Mother, daughters, cluster
  - Global functions
Sub-contracting management
- Technical platforms
- Operating systems
  - Original Design Manufacturing
Technology forecasting
Market research
Customer Services
- Support
  - Repair
Legal
- Sourcing
  - Accessories
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: ‘As a user of the Library Information System (LIS) I 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:** ‘As a user of the Library Information System (LIS) I shall be able to search the recent publications lists for all libraries.’

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

- Requirement: ‘As a user of the Library Information System (LIS) I shall be able to search the recent publications lists for all libraries.’
- Consider the term ‘search … for all’:

  ![Diagram illustrating ambiguity in requirement interpretation]
Example: Ambiguous Requirement

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

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

- Requirement: ‘As a user of the Library Information System (LIS) I 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)
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)?
SW Product Quality -> ISO 9126 (now ISO 25000)

Efficiency = Performance
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>
<th>Task time</th>
<th>Keystroke counts</th>
<th>Opinion poll</th>
<th>Score for understanding</th>
<th>Design-level reqs</th>
<th>Product-level reqs</th>
<th>Guideline adherence</th>
<th>Development process 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>R2: Novice users shall perform tasks Q and R in 15 minutes. Experienced users tasks Q, R, S in 2 minutes.</td>
<td>R3: Recording breakfast shall be possible for guest. No mouse.</td>
<td>R4: 80% of users shall find system easy to use. Recommend system to others.</td>
<td>R5: Show 5 users 10 common error messages. Ask for the cause. 80% of them should give &quot;large.&quot;</td>
<td>R6: System shall use screen pictures in app. xx, buttons work as app. yy.</td>
<td>R7: For all code fields, user shall be able to select value from drop-down list.</td>
<td>R8: System shall follow style guide zz. Menus shall have at most three levels.</td>
<td>R9: Three prototype versions shall be made and usability tested during design.</td>
</tr>
</tbody>
</table>
From Goal to Design

- Requirements can be formulated at various levels:

  - **Goal-level requirement**
    - R1. Our pre-calculations shall hit within 5%

  - **Domain-level requirement**
    - R2. Product shall support cost recording and quotation with experience data

  - **Product-level requirement**
    - R3. Product shall have recording and retrieval functions for experience data

  - **Design-level requirement**
    - R4. System shall have screen pictures as shown in app. xx

Underlying purpose, business, goals, expected/intended improvements

Context, how user and system-to-be-developed collaborate in order to achieve the goals

Externally visible functions and properties of the system

Precise description of data, functions, user-interfaces, etc.

From: Soren Lauesen: Software Requirements
© Pearson / Addison-Wesley 2002
Structure of Lecture 02

• Recap from last week: User Stories
• The Nature of Requirements Engineering (RE)
• The Nature of Requirements
• The Process of RE
• Project Planning based on Requirements
## RE Activities

<table>
<thead>
<tr>
<th>Requirements gathering (= Requirements elicitation)</th>
<th>Requirements analysis</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Interacting with stakeholders to discover their requirements:</td>
<td>• Refining, classifying/clustering, structuring, prioritizing, and modifying the gathered requirements</td>
</tr>
<tr>
<td>• What is to be accomplished?</td>
<td></td>
</tr>
<tr>
<td>• How the system will fit into the needs of the business?</td>
<td></td>
</tr>
<tr>
<td>• How the system will be used on a day-to-day basis?</td>
<td></td>
</tr>
</tbody>
</table>

## Requirements specification

<table>
<thead>
<tr>
<th>Requirements specification</th>
<th>Requirements validation</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Documenting the (system) requirements in a semiformal or formal manner to ensure clarity, consistency, and completeness</td>
<td>• Checking the requirements</td>
</tr>
</tbody>
</table>
RE Activities: Iteration & Concurrency

Initial information
Scope
Constraints

Elicit

Analyze
classify, organize, prioritize, negotiate

Specify

Validate

Model

Requirements traced back to their source

+ Requirements Management
Where/How do we start?
Where/How 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”

**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 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…

**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…
Where/How 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”

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 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…

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 / Limited observability
- Conflicting information / Thin spread (distributed) domain knowledge
- 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.
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
- Brainstorming
- Hard Data analysis
- Interviews
- Prototyping
- Meeting
- ...

- LTAT.05.003 / Lecture 02 / © Dietmar Pfahl 2018
Next Lecture

• Date/Time:
  • Friday, 21-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
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