Lecture 03: Requirements Engineering – Part 2

Dietmar Pfahl
email: dietmar.pfahl@ut.ee

Fall 2019
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: 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: Software Craftsmanship
Week 14: Course wrap-up, review and exam preparation
Week 15: Reserve time slot (no lecture scheduled as of today)
Homework 1 Submission

Make sure you submit your Homework 1 solution on time via your team project wiki on Bitbucket!

(double-check YOUR submission deadline: it’s at midnight before your lab takes place)

Penalties apply for late submission!
No exceptions will be allowed!
Labs Next Week

Lab 2 Assignment:

1. Project Planning
   • Refinement of User Stories into Tasks
   • Prioritization of Tasks (P1, P2, P3, P4)
   • Responsibility assignment
   • Effort estimation

2. Five Use Case Descriptions

3. (Initial) Domain Model
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?
# RE Activities

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

<table>
<thead>
<tr>
<th>Requirements analysis</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Refining, classifying/clustering, structuring, prioritizing, and modifying the gathered requirements</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Requirements specification</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Documenting the (system) requirements in a semiformal or formal manner to ensure clarity, consistency, and completeness</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Requirements validation</th>
</tr>
</thead>
<tbody>
<tr>
<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
Difficulties of Elicitation

- Implicit (tacit) knowledge / Limited observability
- Conflicting information / Thin spread (distributed) domain knowledge
- Say-do problem
- Probe effect (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

1. Analyzing existing documents & data
2. “Brainstorming” possible requirements
   • within the dev. Team
3. Interviews (one-on-one)
   • this is what you did for HW 1…
4. Focus groups or workshops (one-on-many)
   • useful for larger projects
5. Prototyping/mockups
6. 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
Interview Structure

W6H
The journalist’s technique:
What? (Which?)
Where?
Who?
Why?
When?
How?

Investigate the “problem”/”opportunity”

1. What (Which) problem needs to be solved?
   • identify problem Boundaries
2. What might prevent us solving it?
   • identify Feasibility and Risk
3. Where is the problem?
   • understand the Context/Problem Domain
4. Whose problem is it? Who is affected?
   • identify Stakeholders
5. Why does it need solving?
   • identify the stakeholders’ Goals
6. When does it need solving?
   • identify Development Constraints
7. 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 – Speaking Truth to Power
Combine Different Techniques

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

- UNIIVERSITY of TARTU
  INSTITUTE OF COMPUTER SCIENCE
Representation Styles

• Natural language (plus supporting tables and graphs)
• Structured natural language / Scenarios
  • e.g., use case descriptions, user stories, CRC cards, ...
• Semi-formal notations
  • e.g., UML diagrams (use case diagrams, class diagrams, state diagram, sequence charts, etc.)
• Formal notations (with formal semantics)
  • e.g., abstract model-based (Z, VDM, Larch, B, ...) or algebraic (Clear, OBJ, ACT-ONE, ...)
Representation 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 diagram, sequence charts, etc.)
- Formal notations (with formal semantics)
  - e.g., abstract model-based (VDM, Larch, B, ...) or algebraic (OBJ, ACT-ONE, ...)

Not covered in this course
Acknowledgements

Textbooks/Slides:

Example: Home Access Control

Objective: Design an electronic system for:

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

Please read: Ch 1.2 / 1.3.1 / 2.2 Ivan Marsic: Software Engineering, 2012 (http://www.ece.rutgers.edu/~marsic/books/SE/book-SE_marsic.pdf)
## Example NL Requirements

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Priority</th>
<th>Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>REQ1</td>
<td>5</td>
<td>The system shall keep the door locked at all times, unless commanded otherwise by authorized user. When the lock is disarmed, a countdown shall be initiated at the end of which the lock shall be automatically armed (if still disarmed).</td>
</tr>
<tr>
<td>REQ2</td>
<td>2</td>
<td>The system shall lock the door when commanded by pressing a dedicated button.</td>
</tr>
<tr>
<td>REQ3</td>
<td>5</td>
<td>The system shall, given a valid key code, unlock the door and activate other devices.</td>
</tr>
<tr>
<td>REQ4</td>
<td>4</td>
<td>The system should allow mistakes while entering the key code. However, to resist “dictionary attacks,” the number of allowed failed attempts shall be small, say three, after which the system will block and the alarm bell shall be sounded.</td>
</tr>
<tr>
<td>REQ5</td>
<td>2</td>
<td>The system shall maintain a history log of all attempted accesses for later review.</td>
</tr>
<tr>
<td>REQ6</td>
<td>2</td>
<td>The system should allow adding new authorized persons at runtime or removing existing ones.</td>
</tr>
<tr>
<td>REQ7</td>
<td>2</td>
<td>The system shall allow configuring the preferences for device activation when the user provides a valid key code, as well as when a burglary attempt is detected.</td>
</tr>
<tr>
<td>REQ8</td>
<td>1</td>
<td>The system should allow searching the history log by specifying one or more of these parameters: the time frame, the actor role, the door location, or the event type (unlock, lock, power failure, etc.). This function shall be available over the Web by pointing a browser to a specified URL.</td>
</tr>
<tr>
<td>REQ9</td>
<td>1</td>
<td>The system should allow filing inquiries about “suspicious” accesses. This function shall be available over the Web.</td>
</tr>
</tbody>
</table>
Example NL Requirements

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Priority</th>
<th>Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>REQ1</td>
<td>5</td>
<td>The system <strong>shall</strong> keep the door locked at all times, unless commanded otherwise by authorized user. When the lock is disarmed, a countdown shall be initiated at the end of which the lock shall be automatically armed (if still disarmed).</td>
</tr>
<tr>
<td>REQ2</td>
<td>2</td>
<td>The system <strong>shall</strong> lock the door when commanded by pressing a dedicated button.</td>
</tr>
<tr>
<td>REQ3</td>
<td>5</td>
<td>The system <strong>shall</strong> given a valid key code, unlock the door and activate other devices.</td>
</tr>
<tr>
<td>REQ4</td>
<td>4</td>
<td>The system <strong>should</strong> allow mistakes while entering the key code. However, to resist &quot;dictionary attacks,&quot; the number of allowed failed attempts shall be small, say three, after which the system will block and the alarm bell shall be sounded.</td>
</tr>
<tr>
<td>REQ5</td>
<td>2</td>
<td>The system <strong>shall</strong> maintain a history log of all attempted accesses for later review.</td>
</tr>
<tr>
<td>REQ6</td>
<td>2</td>
<td>The system <strong>should</strong> allow adding new authorized persons at runtime or removing existing ones.</td>
</tr>
<tr>
<td>REQ7</td>
<td>2</td>
<td>The system <strong>shall</strong> allow configuring the preferences for device activation when the user provides a valid key code, as well as when a burglary attempt is detected.</td>
</tr>
<tr>
<td>REQ8</td>
<td>1</td>
<td>The system <strong>should</strong> allow searching the history log by specifying one or more of these parameters: the time frame, the actor role, the door location, or the event type (unlock, lock, power failure, etc.). This function shall be available over the Web by pointing a browser to a specified URL.</td>
</tr>
<tr>
<td>REQ9</td>
<td>1</td>
<td>The system <strong>should</strong> allow filing inquiries about &quot;suspicious&quot; accesses. This function shall be available over the Web.</td>
</tr>
</tbody>
</table>

*’shall’: mandatory (?)*

*’should’: optional (?)*
## Example NL Requirements

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Priority</th>
<th>Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>REQ1</td>
<td>5</td>
<td>The system shall keep the door locked at all times, unless commanded otherwise by authorized user. When the lock is disarmed, a countdown shall be initiated at the end of which the lock shall be automatically armed (if still disarmed).</td>
</tr>
<tr>
<td>REQ2</td>
<td>2</td>
<td>The system shall lock the door when commanded by pressing a dedicated button.</td>
</tr>
<tr>
<td>REQ3</td>
<td>5</td>
<td>The system shall, given a valid key code, unlock the door and activate other devices.</td>
</tr>
<tr>
<td>REQ4</td>
<td>4</td>
<td>The system should allow mistakes while entering the key code. However, to resist “dictionary attacks,” the number of allowed failed attempts shall be small, say three, after which the system will block and the alarm bell shall be sounded.</td>
</tr>
<tr>
<td>REQ5</td>
<td>2</td>
<td>The system shall maintain a history log of all attempted accesses for later review.</td>
</tr>
<tr>
<td>REQ6</td>
<td>2</td>
<td>The system should allow adding new authorized persons at runtime or removing existing ones.</td>
</tr>
<tr>
<td>REQ7</td>
<td>2</td>
<td>The system shall allow configuring the preferences for device activation when the user provides a valid key code, as well as when a burglary attempt is detected.</td>
</tr>
<tr>
<td>REQ8</td>
<td>1</td>
<td>The system should allow searching the history log by specifying one or more of these parameters: the time frame, the actor role, the door location, or the event type (unlock, lock, power failure, etc.). This function shall be available over the Web by pointing a browser to a specified URL.</td>
</tr>
<tr>
<td>REQ9</td>
<td>1</td>
<td>The system should allow filing inquiries about “suspicious” accesses. This function shall be available over the Web.</td>
</tr>
</tbody>
</table>

**'Compound' REQ: How test it?**
### Example NL Requirements

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Priority</th>
<th>Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>REQ1</td>
<td>5</td>
<td>The system shall keep the door locked at all times, unless commanded otherwise by authorized user. When the lock is disarmed, a countdown shall be initiated at the end of which the lock shall be automatically armed (if still disarmed).</td>
</tr>
<tr>
<td>REQ2</td>
<td>2</td>
<td>The system shall lock the door when commanded by pressing a dedicated button.</td>
</tr>
<tr>
<td><strong>REQ3</strong></td>
<td>5</td>
<td>The system shall, given a valid key code, unlock the door and activate other devices.</td>
</tr>
<tr>
<td>REQ4</td>
<td>4</td>
<td>The system should allow mistakes while entering the key code. However, to resist “dictionary attacks,” the number of allowed failed attempts shall be small, say three, after which the system will block and the alarm bell shall be sounded.</td>
</tr>
<tr>
<td>REQ5</td>
<td>2</td>
<td>The system shall maintain a history log of all attempted accesses for later review.</td>
</tr>
<tr>
<td>REQ6</td>
<td>2</td>
<td>The system should allow adding new authorized persons at runtime or removing existing ones.</td>
</tr>
<tr>
<td>REQ7</td>
<td>2</td>
<td>The system shall allow configuring the preferences for device activation when the user provides a valid key code, as well as when a burglary attempt is detected.</td>
</tr>
<tr>
<td>REQ8</td>
<td>1</td>
<td>The system should allow searching the history log by specifying one or more of these parameters: the time frame, the actor role, the door location, or the event type (unlock, lock, power failure, etc.). This function shall be available over the Web by pointing a browser to a specified URL.</td>
</tr>
<tr>
<td>REQ9</td>
<td>1</td>
<td>The system should allow filing inquiries about “suspicious” accesses. This function shall be available over the Web.</td>
</tr>
</tbody>
</table>
## Example NL Requirements

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Priority</th>
<th>Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>REQ1</td>
<td>5</td>
<td>The system shall keep the door locked at all times, unless commanded otherwise by authorized user. When the lock is disarmed, a countdown shall be initiated at the end of which the lock shall be automatically armed (if still disarmed).</td>
</tr>
<tr>
<td>REQ2</td>
<td>2</td>
<td>The system shall lock the door when commanded by pressing a dedicated button.</td>
</tr>
<tr>
<td>REQ3</td>
<td>5</td>
<td>The system shall, given a valid key code, unlock the door and activate other devices.</td>
</tr>
<tr>
<td>REQ4</td>
<td>4</td>
<td>The system should allow mistakes while entering the key code. However, to resist &quot;dictionary attacks,&quot; the number of allowed failed attempts shall be small, say three, after which the system will block and the alarm bell shall be sounded.</td>
</tr>
<tr>
<td>REQ5</td>
<td>2</td>
<td>The system shall maintain a history log of all attempted accesses for later review.</td>
</tr>
<tr>
<td>REQ6</td>
<td>2</td>
<td>The system should allow adding new authorized persons at runtime or removing existing ones.</td>
</tr>
<tr>
<td>REQ7</td>
<td>2</td>
<td>The system shall allow configuring the preferences for device activation when the user provides a valid key code, as well as when a burglary attempt is detected.</td>
</tr>
<tr>
<td>REQ8</td>
<td>1</td>
<td>The system should allow searching the history log by specifying one or more of these parameters: the time frame, the actor role, the door location, or the event type (unlock, lock, power failure, etc.). This function shall be available over the Web by pointing a browser to a specified URL.</td>
</tr>
<tr>
<td>REQ9</td>
<td>1</td>
<td>The system should allow filing inquiries about “suspicious” accesses. This function shall be available over the Web.</td>
</tr>
</tbody>
</table>

For **REQ3**, the customer may suggest these test cases:
- Test with the valid key of a current tenant on his or her apartment (pass)
- Test with the valid key of a current tenant on someone else's apartment (fail)
- Test with an invalid key on any apartment (fail)
- Test with the key of a removed tenant on his or her previous apartment (fail)
- Test with the valid key of a just-added tenant on his or her apartment (pass)
User Stories

As a tenant, I can unlock the doors to enter my apartment.

user-role (benefactor) capability (functionality) business-value (motivation/rationale)

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 tool in agile methods.
NL Requirements vs. User Stories

Traditional requirement – “shall” statements:

• “The system shall provide a user configurable interface for all user and system manager functions”

• “The user interface shall be configurable in the areas of:
  • Screen layout
  • Font
  • Background and text color

Corresponding “User Story”:

• “As a system user or system manager, …
  … I want to be able to configure the user interface for screen layout, font, background color, and text color, …
  … so that I can use the system in the most efficient manner”

who - what - why
Example User Stories

<table>
<thead>
<tr>
<th>Identifier</th>
<th>User Story</th>
</tr>
</thead>
<tbody>
<tr>
<td>ST-1</td>
<td>As an authorized person (tenant or landlord), I can keep the doors locked at all times. (… so I am safe)</td>
</tr>
<tr>
<td>ST-2</td>
<td>As an authorized person (tenant or landlord), I want the lock be automatically locked after a defined period of time.</td>
</tr>
<tr>
<td>ST-3</td>
<td>As an authorized person (tenant or landlord), I can lock the doors on demand.</td>
</tr>
<tr>
<td>ST-4</td>
<td>As an authorized person (tenant or landlord), I can unlock the doors. (Test: Allow a small number of mistakes, say three.)</td>
</tr>
<tr>
<td>ST-5</td>
<td>As a landlord, I can at runtime manage authorized persons.</td>
</tr>
<tr>
<td>ST-6</td>
<td>As an authorized person (tenant or landlord), I can view past accesses.</td>
</tr>
<tr>
<td>ST-7</td>
<td>As a tenant, I can configure the preferences for activation of various devices.</td>
</tr>
<tr>
<td>ST-8</td>
<td>As a tenant, I can file complaint about “suspicious” accesses.</td>
</tr>
</tbody>
</table>

Note: 'Why' part is missing in the examples above.
Time Estimation with User Story Points

• Points assigned to individual user stories
• Total work size estimate:

\[ \text{Total size} = \sum \text{points-for-story}_i \quad (i = 1..N) \]

• Velocity (= Productivity) estimated from experience
• Estimate the work duration:

\[ \text{Project duration} = \frac{\text{Total size}}{\text{Velocity}} \quad \text{[time or effort unit]} \]
Time Estimation with User Story Points

- Points assigned to individual user stories
- How?

30 User Stories

Team estimates based on experience

1 day
2 days
0.5 days

Beware: Time vs. Effort!
Time Estimation with User Story Points

- Points assigned to individual user stories
- How?

30 User Stories

Planning Poker Cards
Time Estimation with User Story Points

- Points assigned to individual user stories
- Total work size estimate:

\[
\text{Total size} = \sum \text{points-for-story}_i \quad (i = 1..N)
\]
- Velocity (= Productivity) estimated from experience
- Estimate the work duration:

\[
\text{Project duration} = \frac{\text{Total size}}{\text{Velocity}} \quad \text{[time unit]}
\]
# Example User Stories

<table>
<thead>
<tr>
<th>Identifier</th>
<th>User Story</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>ST-1</td>
<td>As an authorized person (tenant or landlord), I can keep the doors locked at all times.</td>
<td>4 points</td>
</tr>
<tr>
<td>ST-2</td>
<td>The lock should be automatically locked after a defined period of time.</td>
<td>3 pts</td>
</tr>
<tr>
<td>ST-3</td>
<td>As an authorized person (tenant or landlord), I can lock the doors on demand.</td>
<td>6 pts</td>
</tr>
<tr>
<td>ST-4</td>
<td>As an authorized person (tenant or landlord), I can unlock the doors. (Test: Allow a small number of mistakes, say three.)</td>
<td>9 pts</td>
</tr>
<tr>
<td>ST-5</td>
<td>As a landlord, I can at runtime manage authorized persons.</td>
<td>10 pts</td>
</tr>
<tr>
<td>ST-6</td>
<td>As an authorized person (tenant or landlord), I can view past accesses.</td>
<td>6 pts</td>
</tr>
<tr>
<td>ST-7</td>
<td>As a tenant, I can configure the preferences for activation of various devices.</td>
<td>6 pts</td>
</tr>
<tr>
<td>ST-8</td>
<td>As a tenant, I can file complaint about “suspicious” accesses.</td>
<td>6 pts</td>
</tr>
</tbody>
</table>
Example User Stories

<table>
<thead>
<tr>
<th>Identifier</th>
<th>User Story</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>ST-1</td>
<td>...</td>
<td>4 points</td>
</tr>
<tr>
<td>ST-2</td>
<td>...</td>
<td>3 pts</td>
</tr>
<tr>
<td>ST-3</td>
<td>...</td>
<td>6 pts</td>
</tr>
<tr>
<td>ST-4</td>
<td>...</td>
<td>9 pts</td>
</tr>
<tr>
<td>ST-5</td>
<td>...</td>
<td>10 pts</td>
</tr>
<tr>
<td>ST-6</td>
<td>...</td>
<td>6 pts</td>
</tr>
<tr>
<td>ST-7</td>
<td>...</td>
<td>6 pts</td>
</tr>
<tr>
<td>ST-8</td>
<td>...</td>
<td>6 pts</td>
</tr>
</tbody>
</table>

Sum: 50 User Story Points (USP)

Team: 3 persons

How long will it take?
Example Estimation

- Points assigned to individual user stories
- Total work size estimate:
  \[ \text{Total size} = \sum \text{points-for-story}_i \quad (i = 1..N) \]
- Velocity (= Productivity) estimated from experience
- Estimate the work duration:
  \[ \text{Project duration} = \frac{\text{Total size}}{\text{Velocity}} \quad \text{[time unit]} \]

8 User Stories
50 US Points

Average Team Size: 6 persons
Average Project Size: 600 USPs
Average Project Duration: 30 days
Average Velocity (6 pers): 20 USP/day
  = 3.33 USP/person-day

50 US Points & 3 Pers. Team:
Duration:
  ? days
Example Estimation

- Points assigned to individual user stories
- Total work size estimate:
  \[ \text{Total size} = \sum \text{points-for-story}_i \quad (i = 1..N) \]
- Velocity (= Productivity) estimated from experience
- Estimate the work duration:
  \[ \text{Project duration} = \frac{\text{Total size}}{\text{Velocity}} \quad \text{[time unit]} \]

- 50 US Points & 3 Pers. Team:
  Duration: \( \frac{50}{3 \times 3.33} = 5 \) days
## Agile Project Estimation/Planning

### Work backlog

- **ST-4: Unlock** 2.7 day (9pts)
- **ST-2: Auto-Lock** 0.9 day (3pts)
- **ST-1: Keep locked** 1.2 day (4pts)
- **ST-5: Manage Users** 3 day (10pts)
- **ST-6: View History** 1.8 day (6pts)

### Estimated work duration

Assumes that 1 person works fulltime on this task without interruption.

### Items pulled by the team into an iteration

- 1st iteration
  - ST-5: Manage Users 3 day (10pts)
  - ST-6: View History 1.8 day (6pts)

- 2nd iteration
  - ...

- n-th iteration

### List prioritized by the customer

- Items pulled by the team into an iteration

### Estimated completion date

- 1st iteration: 20 days
- 2nd iteration: 5 days (40 hours)
- n-th iteration: 5 days (40 hours)
Wait a second!

Do we really estimate User Stories?
User Stories versus Tasks

How to split User Stories into Tasks:
https://www.youtube.com/watch?v=gZ4uLafsxAk

User Story = Point of view of system user (What?)
Task = Point of view of system developer (How?)
User Stories versus Tasks

How to split User Stories into Tasks:
https://www.youtube.com/watch?v=gZ4uLafsxAk

/polly

Did you watch the videos on the course wiki?
Yes No - anonymous
SMART Tasks

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

Note: The SMART analysis may also be applied to requirements!

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

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

’Implement the UI of system xyz so it looks nice to all users and make sure the response time to inquiries is as fast as the speed of light’

S  Specific
M  Measurable
A  Attainable (Achievable, Actionable, Appropriate)
R  Realistic
T  Time-bound (Timely, Traceable)
Counter-example

'Make the user interface of system xyz look nice to all users and make sure the response time to inquiries is as fast as the speed of light'

Not specific:
- Compound (3 requs – 1 F & 2 NF)

Not measurable:
- What is 'nice'?

Not attainable:
- It is not probable (unrealistic) that 'all' users will find that the UI looks nice / 'UI' is too unspecific

Not realistic:
- Speed of light cannot be achieved

Time-bound/Timely:
- To check this we would need to know the project plan and the planned date when the required function shall be available

Traceable:
- Would need other documents to be able to check whether the req is traceable (BUT: since it is a compound, there is already some potential complication of traceability)
Counter-example:

'Make the user interface of system xyz look nice to all users and make sure the response time to inquiries is as fast as the speed of light'

The task is a mix of functional and non-functional aspects.

Implementation tasks are typically related to functionality.

What about the non-functional requirements, then?

S Specific
M Measurable
A Attainable (Achievable, Actionable, Appropriate)
R Realistic
T Time-bound (Timely, Traceable)
Counter-example

'Make the user interface of system xyz look nice to all users and make sure the response time to inquiries is as fast as the speed of light'

S Specific
M Measurable
A Attainable (Achievable, Actionable, Appropriate)
R Realistic
T Time-bound (Timely, Traceable)

AND:
The task is a mix of functional and non-functional aspects.

Implementation tasks are typically related to functionality.

What about the non-functional requirements, then?

Answer:
- Don’t forget the tasks related to creating tests!
- You might implement tests that check the non-functional requirements!
SMART Tasks

• **Specific**
  A good task 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 Tasks

• **Specific**

  This answers whether you will be able to verify the completion of the project.
  You should avoid signing up for any task 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 Tasks

- **Specific**
  
  This intends to ensure that the Task 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 implement the task

- **Measurable**

- **Attainable** (Achievable, Actionable, Appropriate)

- **Realistic**

- **Time-bound** (Timely, Traceable)

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

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

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

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

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

Where appropriate each task 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 task is “traceable”, which is a separate but important topic in developing software.
Use Cases

• For Functional Requirements Analysis and Specification

• A use case is a description of how a user will use the system-to-be to achieve business goals

  • Detailed use cases are usually written as usage scenarios or scripts, listing a specific sequence of actions and interactions between the actors and the system
Scenarios

- Scenario = real-life example of how a system can be used
- They should include
  - A description of the starting situation (state)
  - A description of the normal flow of events
  - A description of what can go wrong
  - Information about other concurrent activities
  - A description of the state when the scenario finishes
Use Case Diagrams and Descriptions

Use Case Description:
Name of Use Case
Actors associated with Use Case
Pre-conditions
Post-conditions
Normal Flow of Events (Basic Scenario)
Alternative Flow of Events (Alternative Scenarios)

...
Types of Actors

- **Initiating actor** (also called primary actor or “user”): initiates the use case to achieve a goal

- **Participating actor** (also called secondary actor): participates in the use case but does not initiate it:
  - **Supporting actor**: helps the system-to-be to complete the use case
  - **Offstage actor**: passively participates in the use case, i.e., neither initiates nor helps complete the use case, but may be notified about some aspect of it (e.g., for keeping records)
Identifying Actors

- **Ask the following questions:**
  - Who will be a primary user of the system? (primary actor)
  - Who will need support from the system to do her daily tasks?
  - Who will maintain, administrate, keep the system working? (secondary actor)
  - Which hardware devices does the system need?
  - With which other systems does the system need to interact with?
  - Who or what has an interest in the results that the system produces?

- **Look for:**
  - the users who directly use the system
  - also others who need services from the system
Finding Use Cases

For each actor, ask the following questions:

- Which functions does the actor require from the system?
- What does the actor need to do?
- Does the actor need to read, create, destroy, modify, or store some kinds of information in the system?
- Does the actor have to be notified about events in the system?
- Does the actor need to notify the system about something?
- What do those events require in terms of system functionality?
- Could the actor’s daily work be simplified or made more efficient through new functions provided by the system?
### Deriving Use Cases from System Requirements

REQ1: Keep door locked and auto-lock  
REQ2: Lock when "LOCK" pressed  
REQ3: Unlock when valid key provided  
REQ4: Allow mistakes but prevent dictionary attacks  
REQ5: Maintain a history log  
REQ6: Adding/removing users at runtime  
REQ7: Configuring the device activation preferences  
REQ8: Inspecting the access history  
REQ9: Filing inquiries

<table>
<thead>
<tr>
<th>Actor</th>
<th>Actor's Goal (what the actor intends to accomplish)</th>
<th>Use Case Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Landlord</td>
<td>To disarm the lock and enter, and get space lighted up.</td>
<td>Unlock (UC-1)</td>
</tr>
<tr>
<td>Landlord</td>
<td>To lock the door &amp; shut the lights (sometimes?).</td>
<td>Lock (UC-2)</td>
</tr>
<tr>
<td>Landlord</td>
<td>To create a new user account and allow access to home.</td>
<td>AddUser (UC-3)</td>
</tr>
<tr>
<td>Landlord</td>
<td>To retire an existing user account and disable access.</td>
<td>RemoveUser (UC-4)</td>
</tr>
<tr>
<td>Tenant</td>
<td>To find out who accessed the home in a given interval of time and potentially file complaints.</td>
<td>InspectAccessHistory (UC-5)</td>
</tr>
<tr>
<td>Tenant</td>
<td>To disarm the lock and enter, and get space lighted up.</td>
<td>Unlock (UC-1)</td>
</tr>
<tr>
<td>Tenant</td>
<td>To lock the door &amp; shut the lights (sometimes?).</td>
<td>Lock (UC-2)</td>
</tr>
<tr>
<td>Tenant</td>
<td>To configure the device activation preferences.</td>
<td>SetDevicePrefs (UC-6)</td>
</tr>
<tr>
<td>LockDevice</td>
<td>To control the physical lock mechanism.</td>
<td>UC-1, UC-2</td>
</tr>
<tr>
<td>LightSwitch</td>
<td>To control the lightbulb.</td>
<td>UC-1, UC-2</td>
</tr>
<tr>
<td>[to be identified]</td>
<td>To auto-lock the door if it is left unlocked for a given interval of time.</td>
<td>AutoLock (UC-2)</td>
</tr>
</tbody>
</table>

(Actors are often given, if working from user stories instead of ‘shall/should’-statements.)
# Traceability Matrix

Mapping: System requirements to Use cases

<table>
<thead>
<tr>
<th>Req't</th>
<th>PW</th>
<th>UC1</th>
<th>UC2</th>
<th>UC3</th>
<th>UC4</th>
<th>UC5</th>
<th>UC6</th>
<th>UC7</th>
<th>UC8</th>
</tr>
</thead>
<tbody>
<tr>
<td>REQ1</td>
<td>5</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>REQ2</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>REQ3</td>
<td>5</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>REQ4</td>
<td>4</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>REQ5</td>
<td>2</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>REQ6</td>
<td>1</td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>REQ7</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>REQ8</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>REQ9</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
</tr>
</tbody>
</table>

**Purpose:**
- Check all REQs are covered by UCs
- Check no UC is added that doesn’t have a REQ
- Prioritize UCs

PW = Priority Weight

<table>
<thead>
<tr>
<th>Max PW</th>
<th>5</th>
<th>5</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>2</th>
<th>5</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total PW</td>
<td>16</td>
<td>9</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>9</td>
</tr>
</tbody>
</table>

- UC1: Unlock
- UC2: Lock
- UC3: AddUser
- UC4: RemoveUser
- UC5: InspectAccessHistory
- UC6: SetDevicePrefs
- UC7: AuthenticateUser
- UC8: Login

REQ1: Keep door locked and auto-lock
REQ2: Lock when “LOCK” pressed
REQ3: Unlock when valid key provided
REQ4: Allow mistakes but prevent dictionary attacks
REQ5: Maintain a history log
REQ6: Adding/removing users at runtime
REQ7: Configuring the device activation preferences
REQ8: Inspecting the access history
REQ9: Filing inquiries
Use Case Diagram: Device Control

First tier use cases:
- UC1: Unlock
- UC2: Lock

Second tier use cases:
- UC3: AddUser
- UC4: RemoveUser
- UC5: InspectAccessHistory
- UC6: SetDevicePrefs
- UC7: AuthenticateUser
- UC8: Login

Actors:
- Tenant
- Landlord

System boundary:
- Timer
- LightSwitch
- LockDevice

Communication:
- «initiate»
- «participate»
- «include»
Use Case Diagrams and Descriptions

Use Case Description:
Name of Use Case
Actors associated with Use Case
Pre-conditions
Post-conditions
Normal Flow of Events (Basic Scenario)
Alternative Flow of Events (Alternative Scenarios)
...

Use Case Model

Actors

Use Cases

Use-Case Descriptions
### Schema for Use Case Description

<table>
<thead>
<tr>
<th>Use Case UC-#:</th>
<th>Name / Identifier</th>
<th>[verb phrase]</th>
</tr>
</thead>
</table>

- **Related Requirements:** List of the requirements that are addressed by this use case
- **Initiating Actor:** Actor who initiates interaction with the system to accomplish a goal
- **Actor's Goal:** Informal description of the initiating actor's goal
- **Participating Actors:** Actors that will help achieve the goal or need to know about the outcome
- **Preconditions:** What is assumed about the state of the system before the interaction starts
- **Postconditions:** What are the results after the goal is achieved or abandoned; i.e., what must be true about the system at the time the execution of this use case is completed

#### Flow of Events for Main Success Scenario:

1. The initiating actor delivers an action or stimulus to the system (the arrow indicates the direction of interaction, to- or from the system)
2. The system's reaction or response to the stimulus; the system can also send a message to a participating actor, if any
3. ...

#### Flow of Events for Extensions (Alternate Scenarios):
What could go wrong? List the exceptions to the routine and describe how they are handled

1a. For example, actor enters invalid data
2a. For example, power outage, network failure, or requested data unavailable
...

The arrows on the left indicate the direction of interaction: → Actor's action; ← System's reaction
# Use Case 1: Unlock

<table>
<thead>
<tr>
<th>Use Case UC-1:</th>
<th>Unlock</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Related Requirem’ts:</strong></td>
<td>REQ1, REQ3, REQ4, and REQ5</td>
</tr>
<tr>
<td><strong>Initiating Actor:</strong></td>
<td>Any of: Tenant, Landlord</td>
</tr>
<tr>
<td><strong>Actor’s Goal:</strong></td>
<td>To disarm the lock and enter, and get space lighted up automatically.</td>
</tr>
<tr>
<td><strong>Participating Actors:</strong></td>
<td>LockDevice, LightSwitch, Timer</td>
</tr>
<tr>
<td><strong>Preconditions:</strong></td>
<td>• The set of valid keys stored in the system database is non-empty.</td>
</tr>
<tr>
<td></td>
<td>• The system displays the menu of available functions; at the door keypad the menu choices are “Lock” and “Unlock.”</td>
</tr>
<tr>
<td><strong>Postconditions:</strong></td>
<td>The auto-lock timer has started countdown from autoLockInterval.</td>
</tr>
</tbody>
</table>

**Flow of Events for Main Success Scenario:**

→ 1. **Tenant/Landlord** arrives at the door and selects the menu item “Unlock”

→ 2. *include::AuthenticateUser (UC-7)*

← 3. **System** (a) signals to the **Tenant/Landlord** the lock status, e.g., “disarmed,” (b) signals to **LockDevice** to disarm the lock, and (c) signals to **LightSwitch** to turn the light on

← 4. **System** signals to the **Timer** to start the auto-lock timer countdown

→ 5. **Tenant/Landlord** opens the door, enters the home [and shuts the door and locks]
Subroutine «include» UC-7

<table>
<thead>
<tr>
<th>Use Case UC-7: AuthenticateUser (sub-use case)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Related Requirements: REQ3, REQ4</td>
</tr>
<tr>
<td>Initiating Actor: Any of: Tenant, Landlord</td>
</tr>
<tr>
<td>Actor’s Goal: To be positively identified by the system (at the door interface).</td>
</tr>
<tr>
<td>Participating Actors: AlarmBell, Police</td>
</tr>
<tr>
<td>Preconditions:</td>
</tr>
<tr>
<td>• The set of valid keys stored in the system database is non-empty.</td>
</tr>
<tr>
<td>• The counter of authentication attempts equals zero.</td>
</tr>
<tr>
<td>Postconditions: None worth mentioning.</td>
</tr>
</tbody>
</table>

Flow of Events for Main Success Scenario:

← 1. **System** prompts the actor for identification, e.g., alphanumeric key
→ 2. **Tenant/Landlord** supplies a valid identification key
← 3. **System** (a) verifies that the key is valid, and (b) signals to the actor the key validity

Flow of Events for Extensions (Alternate Scenarios):

2a. **Tenant/Landlord** enters an invalid identification key
← 1. **System** (a) detects error, (b) marks a failed attempt, and (c) signals to the actor

```
System (a) detects that the count of failed attempts exceeds the maximum allowed number, (b) signals to sound **AlarmBell**, and (c) notifies the **Police** actor of a possible break-in
```
← 1a. **System** (a) detects that the count of failed attempts exceeds the maximum allowed number, (b) signals to sound **AlarmBell**, and (c) notifies the **Police** actor of a possible break-in

→ 2. **Tenant/Landlord** supplies a valid identification key
3. Same as in Step 3 above
Use Case Diagram: Account Management

UC1: Unlock
UC2: Lock
UC3: AddUser
UC4: RemoveUser
UC5: InspectAccessHistory
UC6: SetDevicePrefs
UC7: AuthenticateUser
UC8: Login

Account Management Subsystem

Tenant

UC3: AddUser

UC4: RemoveUser

UC5: InspectAccessHistory

UC6: SetDevicePrefs

UC8: Login

Landlord

«initiate»

«initiate»

«initiate»

«initiate»

«initiate»

«include»

«include»

«include»

«include»

«initiate»

«initiate»

«initiate»

«initiate»

«initiate»

«include»

«include»

«include»

«include»
Use Case Diagram: Account Management

UC1: Unlock
UC2: Lock
UC3: AddUser
UC4: RemoveUser
UC5: InspectAccessHistory
UC6: SetDevicePrefs
UC7: AuthenticateUser
UC8: Login

Why secondary?
Novice developers frequently identify user login as a use case. Expert developers argue that login is not a use case in its own right.

Recall that use case is motivated by user’s goal; The user initiates interaction with the system to achieve a certain goal. You are not logging in for the sake of logging in—you are logging in to do some work, and this work is your use case.
Optional Use Cases: «extend»

Example optional use cases:

Key differences between «include» and «extend» relationships

<table>
<thead>
<tr>
<th></th>
<th>Included use case</th>
<th>Extending use case</th>
</tr>
</thead>
<tbody>
<tr>
<td>Is this use case optional?</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Is the base use case complete</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>without this use case?</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Is the execution of this use</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>case conditional?</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Does this use case change the</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>behavior of the base use case?</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Where do we want to take you?  
(by the end of this course)

Sample student software projects (Fall)

- [https://github.com/vladislavivanistsev/SuperCap/](https://github.com/vladislavivanistsev/SuperCap/)
- [https://github.com/alajal/license-management/](https://github.com/alajal/license-management/)
Where do we want to take you? (by the end of this course)

Sample student software projects (Fall)
• [https://github.com/vladislavivanistsev/SuperCap](https://github.com/vladislavivanistsev/SuperCap)
• [https://github.com/alajal/license-management/](https://github.com/alajal/license-management/)

Project deadlines


Other deadlines

- Intermediate demo day **21 October**
- [Peer reviews due] (Peer review) **23 November**
- Final Demo Day **16 December**
Project summary

#Project summary

##Introduction

In this course our group consisting of 4 people worked on a project named "SuperCap" for our client Vladislav Ivanistsev. The project created a single page web application, which lets users compare different chemical properties of trios of anions, cations and electrodes and change their properties to see the changes. The properties were visualized by a graph which can be saved for printing or as a PDF for later use.

##Technical overview

Initially we planned on using codeigniter with php, but we ended up using Node.js for the server, which is a javascript based server framework with which none of us had previous experience. For front-end we used Three.JS for WebGL rendering (2 members out of 4 had some previous experience), AngularJS (everyone had some experience) and bootstrap for UI design.

We used MEAN stack which means that we were a team of developers who were capable of developing more agile software together by using a single language (JavaScript) across all layers.

1. [MongoDB](http://www.mongodb.org/) as the database.
2. [Express](http://expressjs.com/) as the web framework
3. [AngularJS](https://angularjs.org/) as the frontend framework
Detailed Use Cases

AndreasGP edited this page on Oct 11, 2015 · 12 revisions

## Use case: Drawing a single graph from input

## Use Case: Printing the graphs selected by the user

## Use Case: Exporting the graphs selected by the user

## Use Case: Adding a new input set with a graph

## Use Case: Removing an input set with a graph

## Use Case: Switching between input sets

## Use case: Changing a graph's parameters

## Use Case: Viewing the about page

## Use Case: Interacting with the 3D models

## Use case: Mock-generating data for an input set
Use Case: Removing an input set with a graph

Preconditions

User has at least one input set added so there is at least one input panel on the sidebar.

Primary actor

User

Main success scenario

1. User clicks on 'X' button on the sidebar input panel.
2. The sidebar input panel is removed, the graph corresponding to that input set is removed.
3. The user can proceed adding or removing more input sets and their graphs.

Functional requirements covered:

- iv. After the user has given input, the system should draw a line graph with a legend with the corresponding chemical substance name.

[Functional requirements](https://github.com/vladislavivanistsev/SuperCap/wiki/Functional%20requirements)

Related issues:
Project Estimation

with

Use Cases

Will be covered later!
Next Lecture

• Date/Time:
  • Friday, 27 Sep, 10:15-12:00

• Topic:
  • Analysis

• For you to do:
  • Make sure you submit your HW1 solution on time (double-check submission deadlines!)
  • Go to next lab → Second HW Assignment