MTAT.03.306
Requirements Engineering

Dr. Raimundas Matulevičius
University of Tartu
Email: rma@ut.ee

On successful completion of this course, students will able to

- Explain the concepts, theories, and best practices associated with requirements engineering
- Elicit, negotiate and document software requirements
- Develop major requirements artefacts and use them during the software development projects
- Apply requirements validation techniques
- Manage software requirements, priorities, and trace them
About the Course

• Course Website

• Lectures
  • Presented during lectures - uploaded to before the lecture

• Practicals
  • Exercises and Workshops done during the practical sessions
  • Exercises for home assignments

• Readings
  • Self-study material
  • Articles and other readings

• Upload
  • Place where you will be able to upload solutions to all home assignments

• Exam
  • This year exam tasks will be uploaded after the exam

About the Course

• Course Website

• Lectures
  – Presented during lectures - uploaded to before the lecture

• Message Board
  – Message Board: http://www.quicktopic.com/52/H/5D9Qn3q6siLs
  – Fell free to post and discuss the course related questions, or provide feedback.

• Upload
  • Place where you will be able to upload solutions to all home assignments

• Exam
  • This year exam tasks will be uploaded after the exam
Course outline

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Introduction</td>
<td>8 September</td>
</tr>
<tr>
<td>2</td>
<td>Stakeholders, goals, scenarios</td>
<td>22 September</td>
</tr>
<tr>
<td>3</td>
<td>Requirements elicitation</td>
<td>6 October</td>
</tr>
<tr>
<td>4</td>
<td>Requirements modelling</td>
<td>20 October</td>
</tr>
<tr>
<td>5</td>
<td>Requirements specification</td>
<td>3 November</td>
</tr>
<tr>
<td>6</td>
<td>Non-functional requirements</td>
<td>17 November</td>
</tr>
<tr>
<td>7</td>
<td>Requirements prioritisation and risk management</td>
<td>24 November</td>
</tr>
<tr>
<td>8</td>
<td>Requirements validation</td>
<td>15 December</td>
</tr>
</tbody>
</table>

Changes are possible!

Workload

6 ECTS = 156 hours of study
(1 ECTS = 26 hours of study)

• Lectures – 16 hours
• Practicals – 16 hours
• Independent work – 124 hours
  – Self-study – 72 hours
    • E.g., reading literature for preparation of colleague presentation, homework assignments, exam, etc.
  – Homework assignments – 40 hours
  – Presentation of the RE article – 12 hours
Modalities and Assessment

• Exercises – 50 % of the final grade
• Presentation – 15 % of the final grade
• Exam – 40 % of the final grade

To be admitted to the exam, at least 30% of grade (exercises+presentation) need to be collected
Lecture 1: Introduction

- Software intensive systems
- Project management
- Requirements engineering
- What is “Requirement”? 
- What is “Engineering”? 
- Lifecycles of engineering projects 
- ‘Requirements’ lifecycles
Lecture 1: Introduction

Software intensive systems

- Project management
- Requirements engineering
- What is “Requirement”?  
- What is “Engineering”? 
- Lifecycles of engineering projects
- ‘Requirements’ lifecycles

Software-Intensive Systems

- **Software (on its own) is useless**
  - Software is an abstract description of a set of computations
  - Software only becomes useful when run on some hardware
    - we sometimes take the hardware for granted
  - Software + Hardware = “Computer System”

- **A Computer System (on its own) is useless**
  - Only useful in the context of some human activity that it can support
    - we sometimes take the human context for granted
  - A new computer system will change human activities in significant ways
  - Software + Hardware + Human Activities = “Software-Intensive System”

- **‘Software’ makes many things possible**
  - It is complex and adaptable
  - It can be rapidly changed on-the-fly
  - It turns general-purpose hardware into a huge variety of useful machines
Quality = Fitness for purpose

• **Software technology is everywhere**
  – Affects nearly all aspects of our lives
  – But our experience of software technology is often frustrating/disappointing

• **Software is designed for a purpose**
  – If it doesn’t work well then either:
    • ...the designer didn’t have an adequate understanding of the purpose
    • ...or we are using the software for a purpose different from the intended one
  – Requirements analysis is about identifying this purpose
  – Inadequate understanding of the purpose leads to poor quality software

• **The purpose is found in human activities**
  – E.g. Purpose of a banking system comes from the business activities
    of banks and the needs of their customers
  – The purpose is often complex:
    • Many different kinds of people and activities
    • Conflicting interests among them

Cost of getting it wrong

• **Cost of fixing errors**
  – Typical development process:
    requirements analysis ⇒ software design ⇒ programming ⇒
    development testing ⇒ acceptance testing ⇒ operation
  – Errors cost more to fix the longer they are undetected
    • E.g. A requirements error found in testing costs 100 times more than
      a programming error found in testing

• **Causes of project failure**
  – Survey of US software projects by the Standish group:

<table>
<thead>
<tr>
<th></th>
<th>2004</th>
<th>2008</th>
</tr>
</thead>
<tbody>
<tr>
<td>Successful</td>
<td>16%</td>
<td>26%</td>
</tr>
<tr>
<td>Challenged</td>
<td>53%</td>
<td>46%</td>
</tr>
<tr>
<td>Cancelled</td>
<td>31%</td>
<td>28%</td>
</tr>
</tbody>
</table>

**Top 3 success factors:**
1) User involvement
2) Executive management support
3) Clear statement of requirements

**Top 3 factors leading to failure:**
1) Lack of user input
2) Incomplete requirements & specs
3) Changing requirements & specs
Where are the challenges?

Which systems are soft?

- **Generic software components**
  - E.g. Core operating system functions, network services, middleware, ...
  - Functionality relatively stable, determined by technical interfaces
  - But note that these systems still affect human activity
    - E.g. concepts of a ‘file’, a ‘URL’, etc.

- **Control Systems**
  - E.g. aircraft flight control, industrial process control, ...
  - Most requirements determined by the physical processes to be controlled
  - But note that operator interaction is usually crucial
    - E.g. accidents caused when the system doesn’t behave as the operator expected

- **Information Systems**
  - E.g. office automation, groupware, web services, business support,…
  - These systems cannot be decoupled from the activities they support
  - Design of the software entails design of the human activity
    - The software and the human activities co-evolve
Lecture 1: Introduction

• Software intensive systems

• **Project management**

• Requirements engineering
• What is “Requirement”?  
• What is “Engineering”?  
• Lifecycles of engineering projects  
• ‘Requirements’ lifecycles

Project Management

• **A manager can control 4 things:**
  – Resources (can get more dollars, facilities, personnel)  
  – Time (can increase schedule, delay milestones, etc.)  
  – Product (can reduce functionality - e.g. scrub requirements)  
  – Risk (can decide which risks are acceptable)

• **To do this, a manager needs to keep track of:**
  – Effort - How much effort will be needed? How much has been expended?
  – Time - What is the expected schedule? How far are we deviating from it?
  – Size - How big is the planned system? How much have we built?
  – Defects - How many errors are we making? How many are we detecting?
    • And how do these errors impact quality?

• **Initially, a manager needs good estimates**
  – …and these can only come from a thorough analysis of the problem.

You cannot control that which you cannot measure!
Project Types

• **Reasons for initiating a software development project**
  – Problem-driven: competition, crisis,…
  – Change-driven: new needs, growth, change in business or environment,…
  – Opportunity-driven: exploit a new technology,…
  – Legacy-driven: part of a previous plan, unfinished work,…

• **Relationship with Customer(s):**
  – Customer-specific - one customer with specific problem
    • May be another company, with contractual arrangement
    • May be a division within the same company
  – Market-based - system to be sold to a general market
    • In some cases the product must generate customers
    • Marketing team may act as substitute customer
  – Community-based - intended as a general benefit to some community
    • E.g. open source tools, tools for scientific research
    • funder ≠ customer (if funder has no stake in the outcome)
  – Hybrid (a mix of the above)

Project Context

• **Existing System**
  – There is nearly always an existing system
    • May just be a set of ad hoc workarounds for the problem
  – Studying it is important:
    • If we want to avoid the weaknesses of the old system,…
    • …while preserving what the stakeholders like about it

• **Pre-Existing Components**
  – Benefits:
    • Can dramatically reduce development cost
    • Easier to decompose the problem if some subproblems are already solved
  – Tension:
    • Solving the real problem vs. solving a known problem (with ready solution)
Lecture 1: Introduction

- Software intensive systems
- Project management

**Requirements engineering**

- What is “Requirement”?
- What is “Engineering”?
- Lifecycles of engineering projects
- ‘Requirements’ lifecycles

**Definition of RE**

Requirements Engineering (RE) is a set of activities concerned with identifying and communicating the purpose of a software-intensive system, and the contexts in which it will be used. Hence, RE acts as the bridge between the real world needs of users, customers, and other constituencies affected by a software system, and the capabilities and opportunities afforded by software-intensive technologies.

- Designers need to know how and where the system will be used
- Requirements are partly about what is needed...
- …and partly about what is possible
- Communication is as important as the analysis
- Quality means fitness-for-purpose. Cannot say anything about quality unless you understand the purpose
- Need to identify all the stakeholders - not just the customer and user
- Not a phase or stage!
Some observations about RE

• RE is not necessarily a sequential process:
  – Don’t have to write the problem statement before the solution statement
    • (Re-)writing a problem statement can be useful at any stage of development
  – RE activities continue throughout the development process

• The problem statement will be imperfect
  – RE models are approximations of the world
    • will contain inaccuracies and inconsistencies
    • will omit some information.
    • analysis should reduce the risk that these will cause serious problems...

• Perfecting a specification may not be cost-effective
  – Requirements analysis has a cost
  – For different projects, the cost-benefit balance will be different

• Problem statement should never be treated as fixed
  – Change is inevitable, and therefore must be planned for
  – There should be a way of incorporating changes periodically

Separate the problem from the solution

• A separate problem description is useful:
  – Most obvious problem might not the right one to solve
  – Problem statement can be discussed with stakeholders
  – Problem statement can be used to evaluate design choices
  – Problem statement is a source of good test cases

• Still need to check:
  – Solution correctly solves the stated problem
  – Problem statement corresponds to the needs of the stakeholders
But design changes the world…

Intertwining of problems and solutions
A problem to describe…

- E.g. “prevent unauthorized access to the OIS system”

Lecture 1: Introduction

- Software intensive systems
- Project management
- Requirements engineering

**What is “Requirement”?**

- What is “Engineering”?
- Lifecycles of engineering projects
- ‘Requirements’ lifecycles
What are requirements?

- **Domain Properties:**
  - things in the application domain that are true whether or not we ever build the proposed system

- **Requirements:**
  - things in the application domain that we wish to be made true by delivering the proposed system
    - Many of which will involve phenomena the machine has no access to

- **A Specification:**
  - is a description of the behaviours that the program must have in order to meet the requirements
    - Can only be written in terms of shared phenomena!

Fitness for purpose?

- **Two correctness (verification) criteria:**
  - The Program running on a particular Computer satisfies the Specification
  - The Specification, in the context of the given domain properties, satisfies the requirements

- **Two completeness (validation) criteria:**
  - We discovered all the important requirements
  - We discovered all the relevant domain properties

- **Example:**
  - Requirement R:
    - “Reverse thrust shall only be enabled when the aircraft is moving on the runway”
  - Domain Properties D:
    - Wheel pulses on if and only if wheels turning
    - Wheels turning if and only if moving on runway
  - Specification S:
    - Reverse thrust enabled if and only if wheel pulses on
  - Verification: S, D ⊨ R
Systems vs. Software Engineering

Lecture 1: Introduction

- Software intensive systems
- Project management
- Requirements engineering
- What is “Requirement”?

**What is “Engineering”?**

- Lifecycles of engineering projects
- ‘Requirements’ lifecycles
What is engineering?

“Engineering is the development of cost-effective solutions to practical problems, through the application of scientific knowledge”

“…Cost-effective…”
- Consideration of design trade-offs, esp. resource usage
- Minimize negative impacts (e.g. environmental and social cost)

“… Solutions …”
- Emphasis on building devices

“… Practical problems …”
- Solving problems that matter to people
- Improving human life in general through technological advance

“… Application of scientific knowledge …”
- Systematic application of analytical techniques

Devices vs. Systems

- Normal design:
  - Old problems, whose solutions are well known
    - Engineering codifies standard solutions
    - Engineer selects appropriate methods and technologies
  - Design focuses on well understood devices
    - Devices can be studied independent of context
    - Differences between the mathematical model and the reality are minimal

- Radical design:
  - Never been done, or past solutions have failed
    - Often involves a very complex problem
  - Bring together complex assemblies of devices into new systems

- Examples:
  - Most of Computer Engineering involves normal design
  - All of Systems Engineering involves radical design (by definition!)
  - Much of Software Engineering involves radical design (soft systems!)
Is software different?

• **Software is different!**
  – software is invisible, intangible, abstract
    • its purpose is to configure some hardware to do something useful
  – there are no physical laws underlying software behaviour
  – there are no physical constraints on software complexity
  – software can be replicated perfectly
    • …no manufacturing variability

• **Software Myths:**
  – Myth: Cost of software is lower than cost of physical devices
  – Myth: Software is easy to change
  – Myth: Computers are more reliable than physical devices
  – Myth: Software can be formally proved to be correct
  – Myth: Software reuse increases safety and reliability
  – Myth? Computers reduce risk over mechanical systems

Lecture 1: Introduction

• Software intensive systems
• Project management
• Requirements engineering
• What is “Requirement”? 
• What is “Engineering”? 

• **Lifecycles of engineering projects**
• ‘Requirements’ lifecycles
Lifecycle of Engineering Project

• **Lifecycle models**
  – Useful for comparing projects in general terms
  – Not enough detail for project planning

• **Examples:**
  – Sequential models: Waterfall, V model
  – Rapid Prototyping
  – Phased Models: Incremental, Evolutionary
  – Iterative Models: Spiral
  – Agile Models: eXtreme Programming

---

**Waterfall Model**

• **View of development:**
  – a process of stepwise refinement
  – largely a high level management view

• **Problems:**
  – Static view of requirements - ignores volatility
  – Lack of user involvement once specification is written
  – Unrealistic separation of specification from design
  – Doesn’t accommodate prototyping, reuse, etc.
**V-Model**

- system requirements
- software requirements
- preliminary design
- detailed design
- code and debug
- unit test
- component test
- software integration
- acceptance test
- system integration

“analyse and design”

“test and integrate”

**Prototyping lifecycle**

<table>
<thead>
<tr>
<th>Preliminary requirements</th>
<th>design prototype</th>
<th>build prototype</th>
<th>evaluate prototype</th>
</tr>
</thead>
</table>

Specify full requirements

- design
- code
- test
- integrate

**Prototyping is used for:**
- understanding the requirements for the user interface
- examining feasibility of a proposed design approach
- exploring system performance issues

**Problems:**
- users treat the prototype as the solution
- a prototype is only a partial specification
Phased Lifecycle Models

Incremental development (each release adds more functionality)

Evolutionary development (each version incorporates new requirements)

The Spiral Model

Determine goals, alternatives, constraints
Evaluate alternatives and risks

Sources: Adapted from Dorfman, 1997, p10
Adapted from Pfleeger, 1998, p57
Agile Models

Source: Adapted from Nawrocki et al, RE’02

- **Basic Philosophy**
  - Reduce communication barriers
    - Programmer interacts with customer
  - Reduce document-heavy approach
    - Documentation is expensive and of limited use
  - Have faith in the people
    - Don’t need fancy process models to tell them what to do!
  - Respond to the customer
    - Rather than focusing on the contract

- **Weaknesses**
  - Relies on programmer’s memory
    - Code can be hard to maintain
  - Relies on oral communication
    - Mis-interpretation possible
  - Assumes single customer representative
    - Multiple viewpoints not possible
  - Only short term planning
    - No longer term vision

---

**Extreme Programming**

- **Instead of a requirements spec, use:**
  - User story cards
  - On-site customer representative

- **Pair Programming**

- **Small releases**
  - E.g. every two or three weeks

- **Planning game**
  - Select and estimate user story cards at the beginning of each release

- **Write test cases before code**

- **The program code is the design doc**
  - Can also use CRC cards (Class-Responsibility-Collaboration)

- **Continuous Integration**
  - Integrate and test several times a day

---

**Extreme Programming**

Each cycle: approx 2-3 weeks

- Collect User stories
- Planning game
- Release
- Write test cases
- code
- integrate
- test

Source: Adapted from Nawrocki et al, RE’02
Lecture 1: Introduction

- Software intensive systems
- Project management
- Requirements engineering
- What is “Requirement”? 
- What is “Engineering”? 
- Lifecycles of engineering projects

Is there a “Requirements Lifecycle”

Source: Adapted from Pohl, CAiSE 1993
Inquiry Cycle

Prior Knowledge (e.g. customer feedback)

Observe (what is wrong with the current system?)

Intervene (replace the old system)

Design (invent a better system)

Model (describe/explain the observed problems)

Messages to take home

- Software intensive systems
- Project management
- Requirements engineering
- What is “Requirement”?
- What is “Engineering”?
- Lifecycles of engineering projects
- ‘Requirements’ lifecycles

Note similarity with Process of Scientific Investigation: Requirements models are theories about the world; Designs are tests of those theories.
Complexity of Purpose

- **People and software are closely-coupled**
  - Complex modes of interaction
  - Long duration of interaction
  - Mixed-initiative interaction
  - Socially-situated interaction
  - …software systems and human activity shape each other in complex ways

- **The problems we’d like software to solve are “wicked”**
  - No definitive formulation of the problem
  - No stopping rule (each solution leads to new insights)
  - Solutions are not right or wrong
  - No objective test of how good a solution is (subjective judgment needed)
  - Each problem is unique (no other problem is exactly like it)
  - Each problem can be treated as a symptom of another problem
  - Problems often have strong political, ethical or professional dimensions

Dealing with problem complexity

- **Abstraction**
  - Ignore detail to see the big picture
  - Treat objects as the same by ignoring certain differences
  - (beware: every abstraction involves choice over what is important)

- **Decomposition**
  - Partition a problem into independent pieces, to study separately
  - (beware: the parts are rarely independent really)

- **Projection**
  - Separate different concerns (views) and describe them separately
  - Different from decomposition as it does not partition the problem space
  - (beware: different views will be inconsistent most of the time)

- **Modularization**
  - Choose structures that are stable over time, to localize change
  - (beware: any structure will make some changes easier and others harder)
Designing for people

• What is the real goal of software design?
  – Creating new programs, components, algorithms, user interfaces, …?
  – Making human activities more effective, efficient, safe, enjoyable, …?

• How rational is the design process?
  – **Hard systems view:**
    • Software problems can be decomposed systematically
    • The requirements can be represented formally in a specification
    • This specification can be validated to ensure it is correct
    • A correct program is one that satisfies such a specification
  
  – **Soft systems view:**
    • Software development is is embedded in a complex organisational context
    • There are multiple stakeholders with different values and goals
    • Software design is part of an ongoing learning process by the organisation
    • Requirements can never be adequately captured in a specification
    • Participation of users and others throughout development is essential
  
  – **Reconciliation:**
    • Hard systems view okay if there is local consensus on the nature of the problem