Requirements Engineering
Fundamentals, Approaches, and Tools
(SE502)

Dr. Mohamed Tounsi
mtounsi@cis.psu.edu.sa

Course Detail

- **Date/time:**
  - Sunday (M) 06:00 pm - 09:00 pm
  - Wednesday (F) 06:00 pm - 09:00 pm

- **Email:** mtounsi@cis.psu.edu.sa

- **Course page:** soon!! (LMS)?
  - Lecture material -- PowerPoint files
  - Assignments & Project
  - Some additional items such as short articles or book excerpts

- **Evaluation:**
  - Assignments: 3-4 ➔ 30%
  - 1 Project ➔ 30%
  - Final ➔ 40%

- **Tools:** Axiom, Doors
Books and References

- **1st Reference:** More About Software Requirements: Thorny Issues and Practical Advice
  
  **Author:** Karl Wiegers
  
  **Publisher:** Microsoft  (2010)
  
  **ISBN:** 978-0-7356-2267-8

- **2nd Reference:** The Requirements Engineering Handbook
  
  **Author:** Ralph R Young
  
  **Publisher:** ArtechPublishings (2004)
  
  **ISBN:** 978-1580532662

- **Articles**

Course syllabus
(I) Introduction

- What are Requirements?
  - Scope (for this course): “Software-intensive Systems”
  - Separating the Problem from the Solution
  - What Requirements Engineers do

- What is Engineering?
  - Engineering as a profession
  - Engineering projects
  - Engineering lifecycles
  - Engineering design

- What is a System?
  - General systems theory
  - Formal foundations of software systems
  - Conceptual foundations of information systems
  - Empirical foundations of human activity systems
  - Observability of systems

(II) Eliciting and Planning

- Elicitation Targets
  - Stakeholders & User Classes
  - System boundaries
  - Goals
  - Scenarios

- Elicitation Techniques
  - Interviews, questionnaires, surveys, meetings
  - Prototyping
  - Ethnographic techniques
  - Knowledge elicitation techniques
  - Conversation Analysis
  - Text Analysis

- The Feasibility Study
  - Types of Feasibility
  - Cost/benefit analysis

- Risk Analysis
  - Identifying and managing risk
(III) Modeling & Analyzing

- Basics of Modeling
  - Notations and their uses
  - Formality and Expressiveness
  - Abstraction and Decomposition
  - Model management and viewpoints
  - Types of Analysis

- Behavior
  - Activities and Interactions
  - States and Transitions
  - Concurrency

- Enterprises
  - Business rules and organizational structures
  - Goals, tasks and responsibilities
  - Soft Systems analysis

- Information Structures
  - Entities and Relationships
  - Classes and Objects
  - Domain Ontologies

- Quality Requirements
  - Taxonomies of NFRs
  - Performance
  - Usability
  - Safety
  - Security
  - Reliability
  - Maintainability

(IV) Communicating & Agreeing

- Validation
  - Refutable descriptions
  - Role of contracts and procurement
  - Role of organizational politics

- Documenting Requirements
  - Properties of a good specification
  - Documentation standards
  - Specification languages
  - Making requirements testable

- Prototyping and Walkthroughs
  - Throwaway prototyping
  - Operational prototyping
  - Walkthroughs of operational models

- Reviews and Inspections
  - Effectiveness of Inspection
  - Conducting an Inspection
  - Collaborative Requirements Workshops

- Negotiation and Prioritization
  - Representing argumentation and rationale
  - Computer-supported negotiation
  - Trade-off analysis
  - Release planning
(V) Realizing and Evolving

- Software Evolution
  - Laws of evolution
  - Release planning
  - Product families
  - Requirement Reuse
- Requirements and Architectures
  - Architectural Patterns and Description Languages
  - Mapping requirements to architectures
  - Architectural Robustness
- Managing Change
  - Baselines and change requests
  - Configuration management and version control
  - Impact Analysis

- Traceability and Rationale
  - Pre- and Post- traceability
  - Capturing Design Rationale
  - Traceability techniques

- Managing Inconsistency
  - On the inevitable intertwining of inconsistency and change
  - Learning from inconsistency
  - Feature interaction
  - Living with inconsistency

- IR and NLP in RE
- Security Requirements

Requirement Engineering Overview
Cobb’s Paradox

"We know why projects fail, we know how to prevent their failure -- so why do they still fail?"

Martin Cobb
Treasury Board of Canada Secretariat
Ottawa, Canada

---

Truth about Software Requirements

- **Truth #1**: If you don't get the requirements right, it doesn't matter how well you execute the rest of the project.
- **Truth #2**: Requirements development is a discovery and invention process, not just a collection process.
- **Truth #3**: Change happens.
- **Truth #4**: The interests of all the project stakeholders intersect in the requirements process.
- **Truth #5**: Customer involvement is the most critical contributor to software quality.
- **Truth #6**: The customer is not always right, but the customer always has a point.
- **Truth #7**: The first question an analyst should ask about a proposed new requirement is, "Is this requirement in scope?"
- **Truth #8**: Even the best requirements document cannot—and should not—replace human dialogue.
- **Truth #9**: The requirements might be vague, but the product will be specific.
- **Truth #10**: You're never going to have perfect requirements.
Project Resolution

Resolution Type 1
Project Success

Resolution Type 2
Challenged

Resolution Type 3
Impaired

Cost Overruns

Percent Over Budget

- <20%
- 21% - 50%
- 51% - 100%
- 101% - 200%
- 201% - 400%
- >400%
Time Overruns

Percent of Time Under Estimated

![Pie chart showing time overruns with percentages in the legend: 
- <20%: 11% 
- 21%-50%: 5% 
- 51%-100%: 14% 
- 101%-200%: 18% 
- 201%-400%: 36% 
- >400%: 1%]

Content Deficiencies

Percent of Originally Planned Functionality

![Pie chart showing content deficiencies with percentages in the legend: 
- <25%: 3% 
- 25%-49%: 7% 
- 50%-74%: 22% 
- 75%-99%: 39% 
- 100%: 27%]

SE502: Software Requirements Engineering
Top Ten Project Success Factors

1. user involvement
2. executive management support
3. clear statement of requirements
4. proper planning
5. realistic expectations
6. smaller project milestones
7. competent staff
8. ownership
9. clear vision and objectives
10. hard-working, focused staff
Properties of Challenged Projects

Top Ten Challenged Project Factors

1. Lack of user involvement
2. Incomplete requirements and specifications
3. Changing requirements and specifications
4. Lack of executive support
5. Technology Incompetence
6. Lack of Resources
7. Unrealistic expectations
8. Unclear objectives
9. Unrealistic timeframe
10. New technology
Properties of Impaired Projects

Top Ten Impaired Project Factors

1. Incomplete requirements
2. Lack of user involvement
3. Lack of resources
4. Unrealistic Expectations
5. Lack of executive support
6. Changing requirements & specs
7. Lack of planning
8. Didn’t need anymore
9. Lack of IT management
10. Technology illiteracy
Case Studies

<table>
<thead>
<tr>
<th>SUCCESS CRITERIA</th>
<th>POINTS</th>
<th>FAILED</th>
<th>SUCCESS</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. User Involvement</td>
<td>19</td>
<td>NO (0)</td>
<td>YES (19)</td>
</tr>
<tr>
<td>2. Executive Management Support</td>
<td>16</td>
<td>NO (0)</td>
<td>YES (16)</td>
</tr>
<tr>
<td>3. Clear Statement of Requirements</td>
<td>15</td>
<td>NO (0)</td>
<td>YES (15)</td>
</tr>
<tr>
<td>4. Proper Planning</td>
<td>11</td>
<td>NO (0)</td>
<td>YES (11)</td>
</tr>
<tr>
<td>5. Realistic Expectations</td>
<td>10</td>
<td>YES (10)</td>
<td>YES (10)</td>
</tr>
<tr>
<td>6. Smaller Project Milestones</td>
<td>9</td>
<td>NO (0)</td>
<td>YES (9)</td>
</tr>
<tr>
<td>7. Competent Staff</td>
<td>8</td>
<td>NO (0)</td>
<td>YES (8)</td>
</tr>
<tr>
<td>8. Ownership</td>
<td>6</td>
<td>NO (0)</td>
<td>YES (6)</td>
</tr>
<tr>
<td>9. Clear Vision &amp; Objectives</td>
<td>3</td>
<td>NO (0)</td>
<td>YES (3)</td>
</tr>
<tr>
<td>10. Hard-Working, Focused Staff</td>
<td>3</td>
<td>NO (0)</td>
<td>YES (3)</td>
</tr>
</tbody>
</table>

Research Literature

- Conferences
  - IEEE International Symposium on Requirements Engineering
  - Many Workshops

- Journals
  - Requirements Engineering Journal, by Springer
  - IEEE Transactions on Software Engineering
  - ACM Transactions on Software Engineering
  - Annals of Software Engineering
  - Software Practice and Experience
  - Automated Software Engineering
  - Journal of Systems and Software
Requirement Engineering Scope

- “The disciplined application of scientific principles and techniques for developing, communicating, and managing requirements.”

Terms and Definitions

- **Requirements Engineering**: The process taken to discovering the purpose of the system

- **Traceability**: Traditional definition revolves around the bi-directional tracing of a written requirement.
  - *Definition extends to the traceability of key pieces of information of a system’s design.*

- **Systems Engineering**: The process and activities associated with developing a system.
  - *This includes all the engineering disciplines needed to build a system: hardware, mechanical, firmware, electrical, and yes – software*

- **Stakeholder**: all involved person in the system (i.e. customer, developer, users)

See reference slides for sources.
Watch this…

How the customer explained it
How the project leader understood it
How the analyst designed it
How the programmer wrote it
What the beta testers received
How the business consultant described it
How the project was documented
What operations installed

How the customer was asked
How it was supported
What marketing advertised
When it was delivered
What the customer really needed
What the dog effect can do to your site
The disaster recovery plan

Definition and Importance of Requirements
What are “Requirements”? (1)

- A **requirement** is:
  - *Capturing the purpose of a system*

- An expression of the ideas to be embodied in the system or application under development

  - *A statement about the proposed system that all stakeholders agree must be made true in order for the customer's problem to be adequately solved*
    - Short and concise piece of information
    - Says something about the system
    - All the stakeholders have agreed that it is valid
    - It helps solve the customer’s problem

What are Requirements? (2)

- **Two basic principles:**
  - 1. *It is useful to separate the problem from the solution*
    - "And to document a problem statement separately from any design solutions"
  - 2. *This separation can never be achieved fully in practice*
    - Because design changes the world, and therefore changes the original problem

- **Why RE is important?**
  - Because failure is expensive!

- **Applications Domains**
  - *RE is more about studying human activity than it is about computers*
Importance of RE (1)

- **Problems**
  - Increased dependence on software
    - E.g. cars, dishwashers, mobile phones, web services, ...
  - Software now the biggest cost element for mission critical systems
    - E.g. Boeing 777
  - Wastage on failed projects
    - E.g. 1997 GAO report: $145 billion over 6 years on software that was never delivered
  - High consequences of failure
    - E.g. Ariane 5: $500 million payload
    - E.g. Intel Pentium bug: $475 million
  - Key factors:
    - Certification costs
      - E.g. Boeing 777: >40% of software budget spent on testing
    - Re-work from defect removal
      - E.g. Motorola: 60-80% of software budget (was) spent on re-work
    - Changing Requirements
      - E.g. California DMV system

Importance of RE (2)

**Mars Climate Orbiter**

- In 1999, the Mars Climate Orbiter disappears around Mars
  
- Cost: about $125M US
  
- Problem caused by a misunderstanding between a team in Colorado and one in California
  
- One team used the metric system while the other used the English system for a key function...
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 is a set of activities that 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.
    - detailed analysis can reduce the risk that these will cause serious problems...
    - ...but that risk can never be reduced to zero

- 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
  - Should be a way of incorporating changes periodically
According to IEEE 830-1993

- A **requirement** is defined as:
  - A condition or capability needed by a user to solve a problem or achieve an objective
  - A condition or a capability that must be met or possessed by a system ... to satisfy a contract, standard, specification, or other formally imposed document ...

What is “Requirements Engineering”??

- **Requirements Engineering (RE)** is:
  - The activity of development, elicitation, specification, analysis, and management of the stakeholder requirements, which are to be met by a new or evolving system
  - **RE** is concerned with identifying the purpose of a software system... and the contexts in which it will be used
    - How/where the system will be used
    - Big picture is important
  - Captures real world needs of stakeholders affected by a software system and expresses them as artifacts that can be implemented by a computing system
    - Bridge to design and construction
    - How to communicate and negotiate?
    - Is anything lost in the translation between different worlds?
Requirements Engineering Activities

---

**Requirements Inception**

- Elicitation
- Analysis
- Specification

**Requirements Development**

**Requirements Management**

**Verification**

About these RE Activities...

- **Inception**
  - Start the process (business need, market opportunity, great idea, ...), business case, feasibility study, system scope, risks, etc.

- **Requirements elicitation**
  - Requirements discovered through consultation with stakeholders

- **Requirements analysis and negotiation**
  - Requirements are analyzed and conflicts resolved through negotiation

- **Requirements specification**
  - A precise requirements document is produced

- **Requirements validation**
  - The requirements document is checked for consistency and completeness

- **Requirements management**
  - Needs and contexts evolve, and so do requirements!
General Problems with the Requirements Process

- Lack of the right expertise (software engineers, domain experts, etc.)

- Initial ideas are often incomplete, too optimistic in the minds of the people leading the acquisition process

- Difficulty of using complex tools and diverse methods associated with requirements discovery may negate the anticipated benefits of a complete and detailed approach

Statistics from NIST Report

- NIST (National Institute of Standards and Technology) has published a comprehensive (309 pages) and very interesting report on project statistics and experiences based on data from a large number of software projects¹
  - 70% of the defects are introduced in the specification phase
  - 30% are introduced later in the technical solution process
  - Only 5% of the specification inadequacies are corrected in the specification phase
  - 95% are detected later in the project or after delivery where the cost for correction on average is 22 times higher compared to a correction directly during the specification effort
  - The NIST report concludes that extensive testing is essential, however testing detects the dominating specification errors late in the process

¹ http://www.nist.gov/public_affairs/releases/n02-10.htm (May 2002)
Why Focus on Requirements?

- Distribution of Defects
  - Requirements: 56%
  - Code: 7%
  - Other: 10%
  - Design: 27%

- Distribution of Effort to Fix Defects
  - Requirements: 82%
  - Code: 1%
  - Other: 4%
  - Design: 13%

Source: Martin & Leffinwell

View of the Software Engineering Institute (SEI)

- Improve software development with the CMM/CMMI model for software development
  - Capability Maturity Model (CMM)
  - For software development, superseded by Capability Maturity Model Integration (CMMI)

- SEI’s vision is:
  - The right software, delivered defect free, on time & on cost, every time
  - “Right software” implies software that satisfies requirements for functionality and qualities (e.g., performance, cost...) throughout its lifetime
  - “Defect free” software is achieved either through exhaustive testing after coding or by developing the code right the first time
Managing Evolving Requirements

“Changing requirements is as certain as death and taxes”

Requirement tools: These seem to have the biggest impact on the success of a project. This may seem strange since “Firm Basic Requirements” is number six on the top ten list. However, these tools, if used as a platform for communications between all the stakeholders, such as executive sponsors and users, can provide enormous benefits. This tool needs to be at the top of the shopping list for any firm involved in developing software applications.


System Requirements versus Software Requirements

- **System requirements**: what the system-to-be should meet;
  “The handbrake shall be released when the driver wants to start.”

- **Software requirements**: what the software-to-be should meet on its own;
  “The software output variable handBrakeCtrl shall have the value off when the software input variable motorRegime gets the value up.”
The scope of RE: the *WHY*, *WHAT*, *WHO* dimensions

System-as-is ...................................... System-to-be
problems, opportunities, system knowledge

Objectives

WHY a new system?

requirements, constraints, assumptions

WHAT services?

WHO will be responsible for what?

assignment

The *WHY* dimension

- Identify, analyze, refine the system-to-be’s objectives
  - to address analyzed deficiencies of the system-as-is
  - in alignment with business objectives
  - taking advantage of technology opportunities

- **Example**: airport train control
  “Serve more passengers”
  “Reduce transfer time among terminals”
The **what** dimension

- Identify & define the system-to-be’s **functional services** (software services, associated manual procedures)
  - to satisfy the identified objectives
  - according to quality constraints: security, performance, ...
  - based on realistic assumptions about the environment

- Example: airport train control
  
  “Computation of safe train accelerations”
  “Display of useful information for passengers inside trains”

Taken from A. van Lamsweerde’s slide s, Chapter 2 of his book. Minor edits. See reference slide.

---

The **who** dimension

- Assign responsibilities for the objectives, services, constraints among system-to-be components
  - based on their capabilities and on the system’s objectives
  - yielding the software-environment boundary

- Example: airport train control
  - “Safe train acceleration” ... **under direct responsibility of software-to-be** (driverless option) or **of driver following software indications** ?
  - “Accurate estimation of train speed/position” ... **under responsibility of tracking system** or **of preceding train** ?

Taken from A. van Lamsweerde’s slide s, Chapter 2 of his book. Minor edits, see reference slide.
What do Requirements Engineers do?

- *identify the "problem"/"opportunity"*
  - Which problem needs to be solved? (identify problem Boundaries)
  - Where is the problem? (understand the Context/Problem Domain)
  - Whose problem is it? (identify Stakeholders)
  - Why does it need solving? (identify the stakeholders’ Goals)
  - How might a software system help? (collect some Scenarios)
  - When does it need solving? (identify Development Constraints)
  - What might prevent us solving it? (identify Feasibility and Risk)

- *and become an expert in the problem domain*

Types of Requirements
So Many “Requirements”... (1)

- A goal is an objective or concern that guides the RE process. It can be used to discover and evaluate functional and non-functional requirements
  - A goal is not yet a requirement...
- Note: All requirements must be verifiable (by some test, inspection, audit etc.)
- A functional requirement is a requirement defining functions of the system under development
  - Describes what the system should do
- A non-functional requirement is a requirement that is not functional. This includes many different kinds of requirements. – Therefore one often considers the following sub-categories:

Different types of non-functional requirements

- Performance requirements, characterizing system properties such as expected performance, capacity, reliability, robustness, usability, etc.
- Design constraints (also called process requirements), providing constraints on how the system should be designed and built – related to development process, documentation, programming language, maintainability, etc.
- Commercial constraints, such as development time frame and costs.
So Many “Requirements”... (2)

- A user requirement is a desired goal or function that a user and other stakeholders expect the system to achieve
  - Does not necessarily become a system requirement
- Application domain requirement (sometimes called business rules) are requirements derived from business practices within a given industrial sector, or in a given company, or defined by government regulations or standards.
  - May lead to system requirements. Can be functional or non-functional
- Problem domain requirements should be satisfied within the problem domain in order to satisfy some of the goals
- System requirements are the requirements for the system to be built, as a whole
  - A system is a collection of interrelated components working together towards some common objective (may be software, mechanical, electrical and electronic hardware and be operated by people)
  - Systems Engineering is a multidisciplinary approach to systems development – software is only a part (but often the problematic part)

So Many “Requirements”... (3)

- Important note: Software Requirements Engineering is a special case of Requirements Engineering. Many topics discussed in this course are quite general and apply to requirements engineering, in general.
- In a system containing software, software requirements are derived from the system requirements. The system then consists of hardware and software, and the hardware (and often the operating system and other existing software modules) are part of the environment in which the software is used.
**Functional Requirements**

- What inputs the system should accept
- What outputs the system should produce
- What data the system should store other systems might use
- What computations the system should perform
- The timing and synchronization of the above

- Depend on the type of software, expected users, and the type of system where the software is used
- Functional user requirements may be high-level statements of what the system should do, but functional system requirements should describe the system services in detail

**Examples of Functional Requirements**

- The user shall be able to search either all of the initial set of databases or select a subset from it.

- The system shall provide appropriate viewers for the user to read documents in the document store.

- Every order shall be allocated a unique identifier (ORDER_ID) which the user shall be able to copy to the account’s permanent storage area.

*Note: not all requirements on this and following slides are high quality requirements but are typical requirements found too often in documents.*
Non-Functional Requirements (NFR) (1)

- Non-functional requirements are important
  - *If they are not met, the system is useless*
  - *Non-functional requirements may be very difficult to state precisely (especially at the beginning) and imprecise requirements may be difficult to verify*
- They are sometimes called quality requirements, quality of service, or extra-functional requirements.
- *Three main categories* \(^1\):
  - *Performance requirements* reflecting: *usability, efficiency, reliability, maintainability and reusability* (note: also security requirements)
    - Response time, throughput
    - Resource usage
    - Reliability, availability
    - Recovery from failure
    - Allowances for maintainability and enhancement
    - Allowances for reusability

\(^1\) Lethbridge and Laganière, Object Oriented Software Engineering: Practical Software Development using UML and Java, 2005

Non-Functional Requirements (NFR) (2)

- *Design constraints*: Categories constraining the *environment and technology* of the system.
  - Platform (minimal requirements, OS, devices...)
  - Technology to be used (language, DB, ...)

- *Commercial constraints*: Categories constraining the *project plan and development methods*
  - Development process (methodology) to be used
  - Cost and delivery date
    - Often put in contract or project plan instead
Various NFR Types

- Other ontologies also exist

Examples of Non-Functional Requirements

- **Product requirement**
  - *It shall be possible for all necessary communication between the APSE and the user to be expressed in the standard Ada character set.*

- **Process requirement**
  - *The system development process and deliverable documents shall conform to the process and deliverables defined in XYZ***

- **Security requirement**
  - *The system shall not disclose any personal information about customers apart from their name and reference number to the operators of the system.*
Measurable Non-Functional Requirements

<table>
<thead>
<tr>
<th>Property</th>
<th>Measure</th>
</tr>
</thead>
<tbody>
<tr>
<td>Speed</td>
<td>Processed transactions/second&lt;br&gt;User/Event response time&lt;br&gt;Screen refresh time</td>
</tr>
<tr>
<td>Size</td>
<td>K Bytes&lt;br&gt;Number of RAM chips</td>
</tr>
<tr>
<td>Ease of use</td>
<td>Training time&lt;br&gt;Number of help frames</td>
</tr>
<tr>
<td>Reliability</td>
<td>Mean time to failure&lt;br&gt;Probability of unavailability&lt;br&gt;Rate of failure occurrence&lt;br&gt;Availability</td>
</tr>
<tr>
<td>Robustness</td>
<td>Time to restart after failure&lt;br&gt;Percentage of events causing failure&lt;br&gt;Probability of data corruption on failure</td>
</tr>
<tr>
<td>Portability</td>
<td>Percentage of target dependent statements&lt;br&gt;Number of target systems</td>
</tr>
</tbody>
</table>


Goals

- **A Goal**
  - Conveys the intention or the objective of one or many stakeholders
  - Can guide the discovery of verifiable non-functional requirements that can be tested objectively
Example of Goal and NFR

- A system goal
  - The system should be easy to use by experienced controllers and should be organized in such a way that user errors are minimized.

- A verifiable usability requirement derived from this goal
  - Experienced controllers shall be able to use all the system functions after a total of three hours of training.
  - The average number of errors made by experienced controllers shall not exceed two per day.

  - Assumption: An experienced controller has at least 2 years experience with the old system (as stated by the stakeholder)

Application-Domain Requirements

- Derived from the application domain

- Describe system characteristics and features that reflect the domain

- May be new functional requirements, constraints on existing requirements, or define specific computations

- If domain requirements are not satisfied, the system may be unworkable
Examples of Application-Domain Requirements

**Library system**
- The system interface to the database must comply with standard Z39.50.
- Because of copyright restrictions, some documents must be deleted immediately on arrival. Depending on the user’s requirements, these documents will first be printed either locally or printed to a network printer and retrieved by the user.

**Train protection system**
- The deceleration of the train shall be computed as:
  \[ D_{\text{train}} = D_{\text{control}} + D_{\text{gradient}} \]
  where \( D_{\text{gradient}} \) is \( 9.81 \text{ms}^2 \times \text{compensated gradient / alpha} \) and where the values of \( 9.81 \text{ms}^2 / \alpha \) are known for different types of train.

Problems Concerning Application-Domain Requirements

**Understandability**
- Requirements are expressed in the language of the application domain
- This is often not understood by software engineers developing the system

**Implicitness / Tacit knowledge**
- Domain specialists understand the area so well that they do not think of making the domain requirements explicit
- People are often unaware of the tacit knowledge they possess and therefore cannot express it to others
Emergent Properties (when the system consists of several sub-systems)

- **Properties of the system as a whole**
  - Requirements which cannot be addressed by a single component, but which depend for their satisfaction on how all the software components interoperate
  - Only emerge once all individual subsystems have been integrated
  - Dependent on the system architecture

- **Examples of emergent properties**
  - Reliability
  - Maintainability
  - Performance
  - Usability
  - Security
  - Safety

The Requirements Engineering Process
Requirements within the software development process

What is the right system to build?
RE activities and documents (Wiegers)

Notes on previous slide

- There needs to be an arrow from User requirements to System requirements. (The system has to be able to perform certain use cases. The same use cases must be supported by the software, therefore become Software requirements.)

- Business rules (including standards and regulations) are not only non-functional, they also include functional aspects (as shown by the arrows in the diagram).
RE process model
(suggested by Bray)

Again, this diagram shows
• RE activities (elicitation, analysis,
specification, HMI design)
• subsequent design activity (internal
design)
• RE documents (elicitation notes,
requirements, specification, HMI
specification)

Important point:
Distinction between
• Problem domain (described by requ.
doc.)
• System (to be built) (described by spec.
doc.)

Note: One has to distinguish between
current (problematic) version of the
problem domain, and the projected future
version which includes the system to be

Typical Layered Approach (V-shaped)

Source: Hull, Jackson, Dick: Requirements Engineering, 2004
Notes on previous slide

Note: This looks like the waterfall process model, but this diagram describes a quite different situation.

- The layers correspond to step-wise refinement in terms of component decomposition.
- For instance, the transition from the first to the second layer is the typical RE process: one starts with the information from elicitation (shown in the first layer), that is, the problematic domain model, and one ends up with a proposal for a new system to be built (which is a component within the projected new domain model).
- **Important note:** The process of identification of the system to be built and defining its relationship with the new domain model (note that the environment of the system to be built may also be re-organized within the new domain model) is a kind of “design process” that requires creativity.
- The transitions to the lower layers in the diagram are similar processes (you may call them RE at a more detailed level or design processes)

Difference between RE and design?

- Much research towards automated SE
  - **Compilers automatically generate machine code** (correct in respect to program source code)
  - **CASE tools automatically generate implementations of UML State Machine models** (correct in respect to the given model)
  - **CASE tools automatically generate state machine models from a set of use case scenarios**
    - E.g. PhD work of Dr. Somé
    - Tool for Live Sequence Charts by Dave - described in the book “Come, Let's Play: Scenario-Based Programming Using LSCs and the Play-Engine”
**Harel’s “scenario-based programming” (1)**

- Scenarios (use cases) are played into the tool, and may be played out for testing the recorded behavior model.

![Fig. 2.4. Conventional system development](image1)

**Harel’s “scenario-based programming” (2)**

Main idea:
eliminate the design and implementation activities by providing efficient execution of behavior directly defined by the requirements.

![Fig. 2.7. A futuristic system development cycle](image2)
**Requirements and Modeling go together**

- The systems engineering sandwich!


---

**Comments on previous slide**

**Why combine RE with modeling?**

- **For analysis** – models help to understand the problem domain

- **For documentation** – models can be used for describing requirements (instead of solely using natural language)
Back to the Sandwich – consider different levels of details

<table>
<thead>
<tr>
<th>Requirements management</th>
<th>Modeling and analysis</th>
</tr>
</thead>
<tbody>
<tr>
<td>Statement of need</td>
<td>e.g. Goal / Usage modeling</td>
</tr>
<tr>
<td>Stakeholder requirements</td>
<td>e.g. Functional modeling</td>
</tr>
<tr>
<td>System requirements</td>
<td>e.g. Performance modeling</td>
</tr>
<tr>
<td>Architectural design</td>
<td></td>
</tr>
</tbody>
</table>

Source: Hull, Jackson, Dick: Requirements Engineering, 2004

Benefits of Requirement Levels (Sandwich)

**Principle of step-wise refinement:**
- Focus the attention on the big picture before addressing the details
- Reduce the number of changes by specifying at a lower level the requirements that will not affect the requirements at a higher level
- Promote the division of work

*This diagram [Lamsweerde] is another way to present this kind of (spiral) process*
Requirements Engineering

- Requirements engineering is a set of activities but not necessarily a separate phase.

Source: Donald C. Gause, Risk Focused Requirements Management, Tutorial at RE’09, September 2009
The Problem Domain and the System/Software-to-be

Problem Domain

- **The problem domain** is the context for requirements
  - *Part of the world within which the problem exists*
  - *Needs to be understood for effective requirements engineering*

- **Domain model**
  - *Set of properties assumed / believed to be true about the environment*
  - *Properties relevant to the problem*
  - *Problem domain requirements should hold in the proposed new version of the domain.*

- **Define the system requirements** such that:
  - *If the system that is built satisfies the system requirements and the environment satisfies the properties assumed for the environment, then the problem domain requirements will be satisfied.*
  - *In simple words: The system will behave as required if the assumptions hold.*
Problem Domain and System-to-be

A domain model should be reusable
(Michael Jackson, 1995)

Diagram also showing activities [Bray]

Problem domain with system-to-be [Bray]

System interface and software interface

Generic architecture of a control system including embedded software [Lamsweerde]

- System and software interface for a control system with embedded software:
  - Software interface: through input and output variables, for instance measuredSpeed (is read by program) and doorState (is set by program)
  - The system includes the software and I/O devices. Therefore the interface of the system with the environment are the monitored and controlled variables of the real world, for instance trainSpeed and doorsClosed.
Software objects representing real objects

- The software (model) normally contains objects that represent objects in the system environment (e.g. the doorState variable represents the state of the doors in the train).

- Whether they represent the situation in the environment correctly, is another question (for the doorState variable, this may depend on the correct functioning of the door state sensing device).

![Diagram showing software objects and their relationships](image)

**Figure 1.3 Phenomena and statements about the environment and the software-to-be**

Main Requirements Activities
Requirements Inception

- **Start the process**
  - Identification of business need
  - New market opportunity
  - Great idea

- **Involves**
  - Building a business case
  - Preliminary feasibility assessment
  - Preliminary definition of project scope

- **Stakeholders**
  - Business managers, marketing people, product managers...

- **Examples of techniques**
  - Brainstorming, Joint Application Development (JAD) meeting...

Requirements Elicitation (1)

- **Gathering of information**
  - About problem domain
  - About problems requiring a solution
  - About constraints related to the problem or solution

- **Questions that need to be answered**
  - What is the system?
  - What are the goals of the system?
  - How is the work done now?
  - What are the problems?
  - How will the system solve these problems?
  - How will the system be used on a day-to-day basis?
  - Will performance issues or other constraints affect the way the solution is approached?
Requirements Elicitation (2)

- **Overview of different sources**
  - Customers and other stakeholders
  - Existing systems
  - Documentation
  - Domain experts
  - More ...

- **Overview of different techniques**
  - Brainstorming
  - Interviews
  - Task observations
  - Use cases / scenarios
  - Prototyping
  - More ...

Requirements Analysis

- The process of studying and analyzing the needs of stakeholders (e.g., customer, user) in view of coming up with a “solution”. Such a solution may involve:
  - A new organization of the workflow in the company.
  - A new system (system-to-be, also called solution domain) which will be used in the existing or modified workflow.
  - A new software to be developed which is to run within the existing computer system or involving modified and/or additional hardware.

- **Objectives**
  - Detect and resolve conflicts between requirements (e.g., through negotiation)
  - Discover the boundaries of the system / software and how it must interact with its environment
  - Elaborate system requirements to derive software requirements
Requirements Specification

- The invention and definition of the behavior of a new system (solution domain) such that it will produce the required effects in the problem domain
- Requirements Analysis has defined the problem domain and the required effects

- Specification Document
  - A document that clearly and precisely describes, each of the essential requirements (functions, performance, design constraints, and quality attributes) of the software and the external interfaces
  - Each requirement being defined in such a way that its achievement is capable of being objectively verified by a prescribed method (e.g., inspection, demonstration, analysis, or test)
  - Different guidelines and templates exist for requirements specification

Requirements Verification and Validation

- Validation and verification
  - Both help ensure delivery of what the client wants
  - Need to be performed at every stage during the process
- Validation: checks that the right product is being built (refers back to stakeholders – main concern during RE)
- Verification: checks that the product is being built right
  - During design phase: refers back to the specification of the system or software requirements
  - During RE: mainly checking consistency between different requirements, detecting conflicts
- Techniques used during RE
  - Simple checks
  - Formal Review
  - Logical analysis
  - Prototypes and enactments
  - Design of functional tests
  - Development of user manual
Requirements Management

- Necessary to cope with changes to requirements
- Requirements change is caused by:
  - Business process changes
  - Technology changes
  - Better understanding of the problem
- Traceability is very important for effective requirements management

Requirements Documents

- Vision and Scope Document
- Elicitation notes: (Raw) requirements obtained through elicitation; often unstructured, incomplete, and inconsistent
- (Problem domain) Requirements document
- System requirements document
- Software requirements document
  - The software is normally part of a system that includes hardware and software. Therefore the software requirements are normally part of the system requirements.
  
  Note: System and Software requirements may exist in several versions with different levels of details, such as
  - User (customer) requirements: Statements in natural language plus diagrams of the services the system provides and its operational constraints; written for customers
  - Detailed requirements: A structured document setting out detailed descriptions of the system services; often used as a contract between client and contractor. This description can serve as a basis for a design or implementation; used by developers.
Types of Requirements Documents

Two extremes:

- An informal outline of the requirements using a few paragraphs or simple diagrams
  - This is called the requirements definition

- A long list of specifications that contain thousands of pages of intricate requirements describing the system in detail
  - This is called the requirements specification

Requirements documents for large systems are normally arranged in a hierarchy

The Requirements Analyst

- Plays an essential communication role
  - Talks to users: application domain
  - Talks to developers: technical domain
  - Translates user requirements into functional requirements and quality goals

- Needs many capabilities
  - Interviewing and listening skills
  - Facilitation and interpersonal skills
  - Writing and modeling skills
  - Organizational ability

- RE is more than just modeling...
  - This is a social activity!

[1] Karl Wiegers, In Search of Excellent Requirements
For More Information

   http://www.wired.com/news/technology/bugs/0,2924,69355,00.html
3. INCOSE Requirements Working Group  
   http://www.incose.org/practice/techactivities/wg/rgmts/
4. Tools Survey: Requirements Management (RM) Tools  
   http://www.incose.org/productspubs/products/rmsurvey.aspx  
   http://www.volere.co.uk/tools.htm
7. Requirements Engineering Conference  
   http://www.requirements-engineering.org/

Main References

2. Soren Lauesen: Software Requirements - Styles and Techniques, Addison Wesley, 2002