A Software Life-cycle Model
Which part will we talk about today?

Validate Requirements, Verify Specification

Requirements

System Design
(Architecture, High-level Design)

Module Design
(Program Design, Detailed Design)

Implementation of Units (classes, procedures, functions)

Unit testing

Acceptance Test
(Release testing)

System Testing
(Integration testing of modules)

Module Testing
(Integration testing of units)

Verify System Design

Verify Module Design

Verify Implementation

Project Management, Software Quality Assurance (SQA), Supporting Tools, Education

Part I
Requirements Elicitation

Part II
Requirements Analysis

Part III
Requirements Specification

Aseel.Berglund@liu.se
Kristian.Sandahl@liu.se
What is a software requirement?

 “Software requirements express the needs and constraints placed on a software product that contribute to the solution of some real-world problems.” (Kotonya and Sommerville, 2000)

 Example:

When the user enters the degrees in Farenheit, the system shall calculate and write the degrees in Celsius.
How do you write a natural language requirement?

- To avoid misunderstandings, always use a complete **sentence**.
- A sentence expresses a complete thought and consists of a subject and a predicate.

- Use modal verbs: ”**shall**”, ”**will**”, and ”**must**”
- Don’t use: ”should”, ”would”, or ”might”

- “Timely review should be done as soon as possible, and shall not exceed two working weeks.”
  (TS/ISO 16949)
365 IT companies with 8 380 different IT-projects were analysed 1995

**31.1% canceled** at some point during the development cycle

**52.7% challenged** - completed and operational but over-budget, over the time estimate, and offers fewer features and functions than originally specified.

**16.2% completed** on-time and on-budget, with all features as initially specified

*Source: The Standish Group CHAOS Report 1995*
IT Project Success Rates

User involvement is the number one reason for project success.

Source: The Standish Group CHAOS Reports
Feature and function usage

- Never: 45%
- Always: 7%
- Often: 13%
- Sometimes: 16%
- Rarely: 19%
Part I
Requirements Elicitation

Part II
Requirements Analysis

Part III
Requirements Specification

Iterative Process

Maturation / Knowledge

Risk

Time
Iterative process

Part I
Requirements Elicitation

Collect user requirement

Elicitation

Analyze

Specify

Validate

Understand

Check that it matches user/customer requirements

Document/build

Part II
Requirements Analysis

Part III
Requirements Specification

Aseel.Berglund@liu.se
Kristian.Sandahl@liu.se
Important questions

- Why – purpose, goals
- Who – the customer, the user, stakeholder
- What – the needs
- How – realization
What will you learn today?

- Part I – Requirements elicitation
- Part II - Requirements analysis
- Part III - Requirements specification
Part I

Requirements Elicitation
“If I’d asked my customers what they wanted, they’d have said a faster horse.”

Henry Ford
"You can't just ask customers what they want"

Steve Jobs
Elicitation

Purpose:

- Identify the customer
- Understand the **true** needs of the customer
- Trace future implementation to needs
Elicitation

Sources:
- Goals
- Domain knowledge
- Stakeholders
- Environment
- Current situation

Techniques:
- Interviews
- Scenarios
- Prototypes
- Facilitated meeting
- Observation
- Wizard-of-oz studies
- Workshops
- Focus groups
- Split testing
- .....
Interviews

Process:
- Start
- Q & A
- Summary teach-back
- Thank you!
- What’s next

Kinds:
- Structured
- Unstructured

Tips:
- Be 2 interviewers – shift roles
- Plan the interview
- Don’t stick to the plan – use feelings
- Let the customer talk
- Prepare ice-breakers
- Probe thinking
- Look for body language
- Think of human bias
- Why do you get the answers you get?
Part II

Requirements Analysis
Goal

- Detect and resolve conflicts between requirements
- Discover bounds of software
- Define interaction with the environment
- Elaborate high-level requirements to derive detailed requirements
- Classify requirements for more effective management

Means:
- Often accomplished with conceptual modelling
Functional and non-functional requirements

- **Functional requirements:**
  - Describe the functions that the software is to execute.
  - Can be tested by giving input and checking the output.
  - Example: “The user shall be able to add a contact.”

- **Non-functional requirements:**
  - Design constraints
    Example: “The system shall be implemented in php.”
  - Quality requirements, possible to measure
    Example: “The minimum response time is 2.0 seconds”
Quality factors

- Correctness
- Reliability
- Efficiency
- Usability
- Integrity
- Maintainability
- Flexibility
- Testability
- Security
- Portability
- Reusability
- Interoperability
- Survivability
- Safety
- Manageability
- Supportability
- Replaceability
- Functionality
Usability engineering

**Part I**
Requirements Elicitation

**Part II**
Requirements Analysis

**Part III**
Requirements Specification

---

**DILBERT by Scott Adams**

**YOUR USER REQUIREMENTS INCLUDE FOUR HUNDRED FEATURES.**

**DO YOU REALIZE THAT NO HUMAN WOULD BE ABLE TO USE A PRODUCT WITH THAT LEVEL OF COMPLEXITY?**

**GOOD POINT. I'D BETTER ADD “EASY TO USE” TO THE LIST.**
The business benefits of adding usability to a product development process include:

- Increased productivity
- Increased sales and revenues
- Decreased training and support costs
- Reduced development time and costs
- Reduced maintenance costs
- Increased customer satisfaction
Usability

The extent to which a product can be used by specified **USERS** to achieve specified **GOALS** with effectiveness, efficiency, and satisfaction in a specified context of use.

ISO 9241-11

- Effectiveness - Task completion by users
- Efficiency - Task in time
- Satisfaction - Responded by user in term of experience

context of use - users, tasks, equipments & environments
Usability factors

- Relevance
- Efficiency
- Attitude
- Learnability

- [Usability metrics](#)
The probability that the software executes with no failures during a specified time interval

- Approximation: \( \text{MTTF}/(1+\text{MTTF}) \)
- Example
- Easier to manage: Failure intensity, [failures / hours of execution time]
- Another approximation: \( \lambda = (1-R)/t \)
- Example
## Failure intensity guideline

<table>
<thead>
<tr>
<th>Impact</th>
<th>Failure intensity</th>
<th>Time btwn failures</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hundreds of deaths, $10^9$ cost</td>
<td>$10^{-9}$</td>
<td>114 000 years</td>
</tr>
<tr>
<td>1-2 deaths, $10^6$ cost</td>
<td>$10^{-6}$</td>
<td>114 years</td>
</tr>
<tr>
<td>$1000$ cost</td>
<td>$10^{-3}$</td>
<td>6 weeks</td>
</tr>
<tr>
<td>$100$ cost</td>
<td>$10^{-2}$</td>
<td>100 h</td>
</tr>
<tr>
<td>$10$ cost</td>
<td>$10^{-1}$</td>
<td>10 h</td>
</tr>
<tr>
<td>$1$ cost</td>
<td>1</td>
<td>1 h</td>
</tr>
</tbody>
</table>
A distinguishing characteristic of a system item (includes both functional and nonfunctional attributes such as performance and reusability).

(IEEE Std 829)

Higher level stuff, used in advertising:
“The system shall have an SMS-service”
Requirements Abstraction Model

Organizational Strategies

Product Strategies

RAM - Abstraction Levels
- Product Level (goal)
- Feature Level (features)
- Function Level (functions/actions)
- Component Level (details- consists of)

Utilizing several levels of abstraction

A structured way of working

Source: Tony Gorschek, BTH

Part I
Requirements Elicitation

Part II
Requirements Analysis

Part III
Requirements Specification
Part I
Requirements Elicitation

Part II
Requirements Analysis

Part III
Requirements Specification

REQUIREMENTS

RAM - Action Steps

- Specify (elicit)
- Place (evaluate)
- Abstraction (work-up)

RAM - Abstraction Levels

- Product Level
  - Usability internationally
  - Created requirement

- Feature Level
  - C: Support for multiple languages
  - Original requirement

- Function Level
  - Choose language during system operation
  - Help functions displayed in the right language
  - Addition of languages to the system
  - Created requirement

- Component Level
  - Interface adoption to language text length, e.g. pictograms
  - Created requirement
Requirements classification

- Functional vs non-functional requirements
- Source
- Product or process requirements
- Priority
- Scope in terms of affected components
- Volatility vs stability
Conceptual Modelling

- Representation in semi-formal notation
- Often diagrammatic representation
- Examples:
  - Object-orientation, use-cases, state-machines
  - Activity diagrams
  - Data flow diagrams
  - Entity-relationship models

Requires a paradigm shift to give full advantage
Use-case modelling

A use-case is:
“… a particular form or pattern or exemplar of usage, a scenario that begins with some user of the system initiating some transaction of sequence of interrelated events.”

Jacobson, m fl 1992: Object-oriented software engineering. Addison-Wesley
A CoffeeDrinker approaches the machine with his cup and a coin of SEK 5. He places the cup on the shelf just under the pipe. He then inserts the coin, and presses the button for coffee to get coffee according to default settings. Optionally he might use other buttons to adjust the strength and decide to add sugar and/or whitener. The machine processes the coffee and bell when it is ready. The CoffeeDrinker takes his cup from the shelf.
Use-case diagram for the coffee-machine

Subject boundary

CoffeDrinker

TeaDrinker

Subject

CoffeeMachine

Buy a cup of coffee

Get coin in return

Pour hot water

Service

Porter

Clean the Machine

Add substances

Collect coins

Brew a can of coffee

Subject name
Relations between use-cases

- Extend loan
- Borrow copy of book
- Check for reservation
- Refuse loan

Stereotype: extended classification of meaning

"Separating scenarios"

"Reuse"

Please, keep as simple as possible.
Identifying classes: noun analysis

A CoffeeDrinker approaches the **machine** with his **cup** and a **coin** of SEK 5. He places the cup on the **shelf** just under the **pipe**. He then inserts the coin, and press the **button** for coffee to get coffee according to default settings. Optionally he might use other buttons to adjust the strength and decide to add **sugar** and/or **whitener**. The machine processes the **coffee** and bell when it is ready. The CoffeeDrinker takes his cup from the shelf.

- **machine** – real noun handled by the system
  - **cup** – unit for beverage
  - **coin** – detail of user and machine
  - **shelf** – detail of machine
  - **pipe** – detail of machine

- **button** – handled by the system
  - **sugar** – detail of coffee
  - **whitener** – detail of coffee

- **cup of coffee** – handled by the system
  - **indicator** – not discovered
The coffee machine class model

Even small models take space. You need good drawing tools and large sheet.
Data model: ER-diagram

- **Student**
  - Name
  - Personal number
  - Curriculum

- **Enrolled-in**

- **Course**
  - Subject
  - Course code
  - Max-enrolment
Part III

Requirements Specification
Advice towards a good specification

- There is no perfect specification, but you can write a good one
- The RS, or SRS avoids many misunderstandings
- The RS is of special importance in outsourcing programming
1 Introduction
   1.1 Purpose
   1.2 Scope
   1.3 Definitions, acronyms and abbreviations
   1.4 References
   1.5 Overview

2 Overall description
   2.1 Product perspective
   2.2 Product functions
   2.3 User characteristics
   2.4 General constraints
   2.5 Assumptions and dependencies
   2.6 Lower ambition levels

3 Specific requirements
   3.1 Interface requirements
      3.1.1 User interfaces
      3.1.2 Hardware interfaces
      3.1.3 Software interfaces
      3.1.4 Communication interfaces
   3.2 Functional requirements
   3.3 Performance requirements
   3.4 Design constraints
   3.5 Software system attributes
   3.6 Other requirements

4 Supporting information
   4.1 Index
   4.2 Appendices
Requirements specification

Requirements are:
- Numbered
- Inspected
- Prioritised
- Unambiguous
- Testable
- Complete
- Consistent

- Traceable
- Feasible
- Modifiable

Useful for:
- operation
- maintenance
- customer
- developer
- ....
The role of requirements in the life-cycle

Part I
Requirements Elicitation

Part II
Requirements Analysis

Part III
Requirements Specification

The role of requirements in the life-cycle

- Carol: the customer
- Tim: the tester
- Diana: the developer

fuzziness
elicitation
specification
modelling
formalisation
time
Why do we need requirements?

Useful for:
• Design
• Test
• Support
• Planning
• Maintenance

Communication to stakeholders
Make trade-offs
Resolve conflicts
Finding true needs
Contract document
Summary - What have we learned today?

- Elicitation is a very human-centered phase
- Important classes: Functional- and non-functional requirements
- Use-cases describe the mainstream flow of event
- A written specification is read far more often than it is written