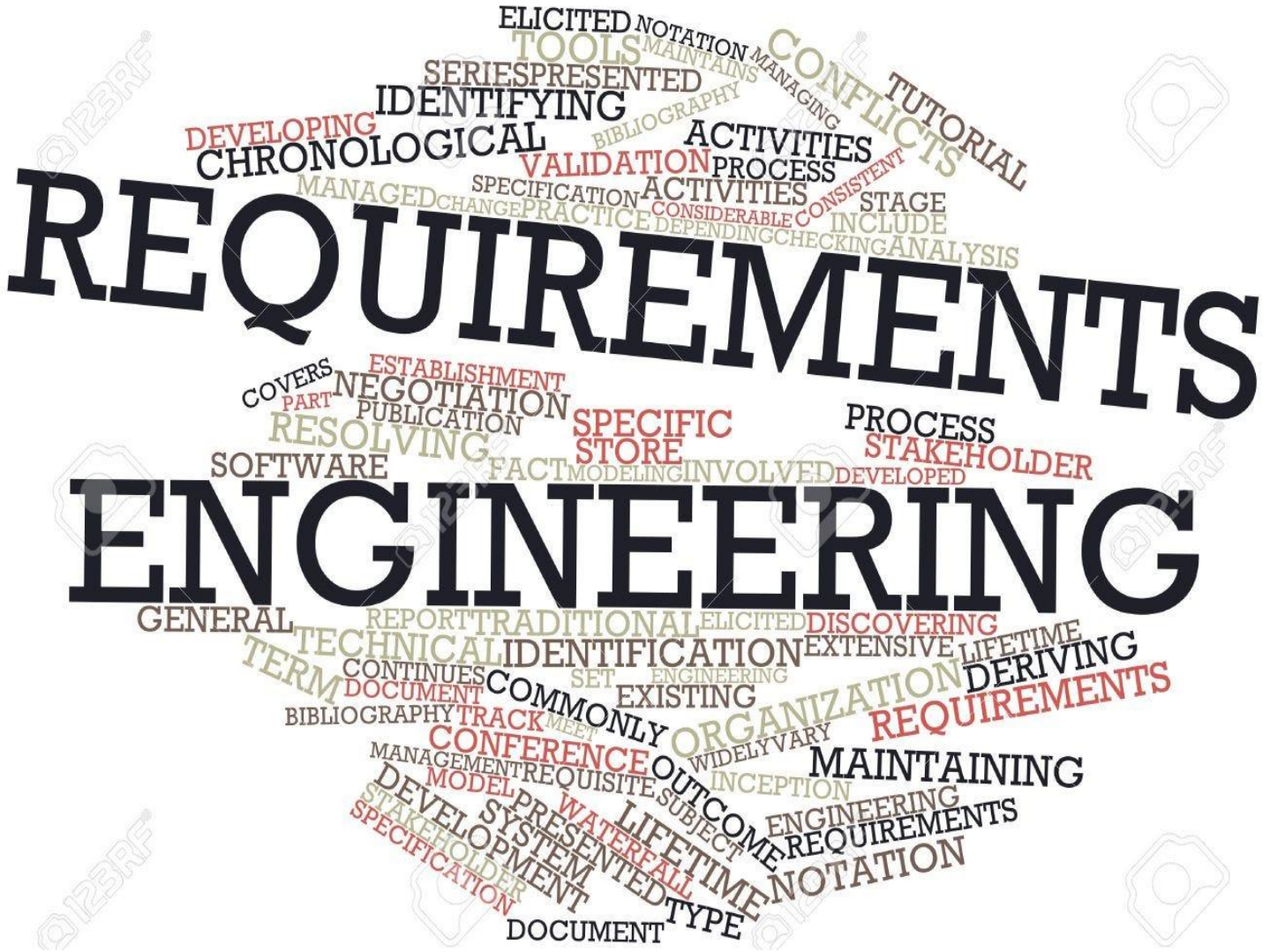
JYVÄSKYLÄN YLIOPISTO
UNIVERSITY OF JYVÄSKYLÄ

Requirements Engineering and User-Centered Design (UCD)

Dr. Rahul Mohanani, 2025

Today's Agenda:

- Basics of SDLC and RE
- Types of Requirements
- Requirements Elicitation
- Activity 1
- What is UCD
- UCD Techniques
- Introduction to Design Thinking
- Activity 2 - 5



Why is RE essential:

- Requirements gathering and management form the foundation of **successful SE projects**.
- Clear and well-managed requirements **might prevent costly misunderstandings and re-work**.
- Helps in identifying **potential issues early, saving time and other resources**.
- 'Good' requirements engineers can **navigate complex requirements, prioritize them effectively and adapt to changes more smoothly.**
- Facilitates **communication/discussion with stakeholders**, **design, implementation, test- and integration phases** of a project, **verification and acceptance of the product** (quality assurance), General project organisation and management.

Basics of SDLC and RE:

- System software VS Application software
- What is Software Development?
- SDLC – Software Development Life Cycle – can be thought about as a **framework, or a process, or a model, or a guideline** that describes the activities performed to solve certain problems.
- Involves six stages (sometimes seven) – r**equirements gathering, design and prototyping**, i**mplementation, testing, deployment and maintenance.**
- Importance of Requirements Engineering in SDLC.
- Waterfall Vs Agile development approach.

Basics of Requirements Engineering

What is a 'Requirement'?

- A **need** or a **constraint** imposed by a stakeholder
- A **capability** or a **property** that a system shall have.
- A **documented and clear representation** of need or capability or property

What is Requirements Engineering?

Requirements Engineering (RE) is the systematic, iterative, and disciplined approach to develop an explicit set of requirements and system specification that all stakeholders agree upon.

Elementary tasks include, but are not limited to:

- Requirements **elicitation** (identifying all relevant requirements)
- Requirements **analysis** (understanding requirements and achieving consensus)
- Requirements **specification** (structuring, modelling, documenting requirements)
- Requirements **validation and verification** (ensuring validity and quality)

Basics of Requirements Engineering

Requirements engineering aims at the **problem spac**e...

- Framing the problem for a development task as comprehensively and as precisely as possible --- **Why is something necessary?**

- Deriving appropriate requirements towards the desired solution and agreement between stakeholders --- **What (capabilities, properties) is necessary?**

HOWEVER …

Design and implementation, in contrast, aims at the solution space, i.e. development and evolution of of possible solutions of: **How will the solution* be realised?**

Differentiating the problem space from the solution space is essential!

Basics of Requirements Engineering

Overall: .....

1. RE is not a means in itself, but a means to an end.
2. RE is not just about software, but about humans, systems, and processes.
3. Collecting all requirements and specifying them in an unambiguous manner upfront is an illusion. Nevertheless, RE should be done with care as it decides upon project success and failure.
4. RE can have different forms and interpretations.

Basics of Requirements Engineering

Properties of “good” requirements

Conditions that individual requirements should fulfil:

- Correctness
- Feasibility
- Unambiguity
- Verifiability

Characteristics collections of requirements should fulfil:

- Completeness
- Consistency

Basics of Requirements Engineering

Related: Form and content of requirements

A (documented) requirement has:

- A form (its “syntax”) --- **The way a requirement is described (text, figures, models etc.)**
- A content (its “semantics”) --- **Which capabilities are demanded?**

Both the syntax and the semantics are key to:

- **Precision of a requirement**: Is the content described unambiguously? Are clear, free of misunderstandings, or free of conflicts?
- **Understandability:** Is the requirement intuitive and easy to understand (for the respective stakeholder groups)?
- **Unambiguity:** Is it possible to clearly decide if a system/product fulfils a requirement (independent of subjective interpretation/available domain expertise)?
- **Verifiability**: Is the fulfilment of a requirement efficient enough to assess?

Types of Requirements

1. **Functional Requirements:**

- Behavior of a system in general
- Behavior from the perspective of the user.

2. **Quality Requirements:**

- Quality characteristics and properties of a system. E.g. safety, security, compliant etc.
- (Ideally) quantified properties regarding the behavior.

3. **Process Requirements (project-specific)**

- Requirements towards the development process (Scheduling, milestones, budget etc.)
- Definitions or rules for implementation (constraints, components, re-use etc.).

2 and 3 are together often called as ‘non-functional requirements’.

Types of Requirements

Not a requirement, but also in scope of RE

In RE, we only consider...

- Requirements that define the **properties and capabilities the desired system must have.**

However,

- **Contextual information** that is relevant to understand requirements but that are no requirements themselves

Examples:

- **Domain knowledge** and terminology typically captured in a glossary
- **Data models, system boundaries, interface descriptions**
- **Immutable rules,** e.g., laws, physics, and any relevant limitations.

Requirements Need to be Engineered

Typically, requirements are vague, ambiguous, not measurable

An elementary task in RE is to continuously engineer the requirements by

- refining,
- classifying,
- specifying, and
- validating them

Requirements Need to be Engineered: Refining

Problem space (In scope of RE):

- Goals and context: general specification and description of the rationale →**WHY**?

- Requirements:

Coarse-grained requirements: general properties/characteristics of the system (often without precise definitions)

Fine-grained (atomic) requirements: detailed requirements which are specific, measurable, verifiable etc. to realise in the system –> **WHAT?** (System view: “black box”).

Solution space (In scope of Design and Implementation):

- **System specification**: description of the functional and non-functional properties/characteristics of a system at a level of detail that supports the further development → **HOW?** (System view: “glass box”)

Requirements Need to be Engineered: Abstraction

In an "ideal world", RE is conducted top-down as follows:

- Starting with agreed and "generically" phrased goals... e.g., "The product must be usable in an international setting."

- ...coarse-grained requirements are defined, discussed, and documented... "The product must be usable in developing countries", "The product must be resistant to issues in power supply".

- ...from which refined and detailed requirements are derived...

"The system needs to have an independent emergency power generator..."

- ...to finally conclude everything in a system specification:

"For the generator, the company XYZ is contracted..."

Note: The other way around is also possible, i.e., to start with a given system and infer the requirements and goals from that system (Reverse Engineering)

Requirements Need to be Engineered: Requirements Elicitation

- Can be simply understood as a process where we extract the **desires**, **preferences** and '**must-haves**' of a system.
- Need to understand the **stakeholders** – the clients, users or anyone that has a vested interest in the software system in terms of what they need, and to meet certain 'expectations'.
- Elicitation is usually a **dialogue** - asking the right questions, actively listening and sometimes deciphering what users might struggle to articulate. i.e. *gathering accurate*, *correct*, *complete and clear information*.

STAKEHOLDERS:

- Individuals or groups that have a vested interest in the output. Having, usually, different and often conflicting opinions.
- Types: End-users, clients, managers, investors, regulatory bodies or government organizations.
- Recognizing and actively involving the stakeholders is critical for any requirements elicitation process.

Requirements Need to be Engineered: Techniques of Requirements Elicitation

1. Conducting Interviews: Can explore and reveal User-Needs, Functionality Requirements, Any integration with existing tools or resources, Security and Safety Requirements, Usability and Training, Usability Features etc.

2. Surveys and Questionnaires: Instead of individually interviewing, SaQ allows us to gather inputs from a large population of sample stakeholders. The key characteristic is the scalability, anonymity, time efficiency. One must look for trends, areas and preferences where opinions divert; which will eventually guide the development team.

3. Joint Application Sessions (JAD): JAD brings together stakeholders, managers and development team to explore and define the goals, requirements and other details of the system. Select participants from each sample set, define the objectives, facilitate collaborative workshops, increase the communication and real-time problem solving. Facilitator holds the key role by keeping the session focused and productive.

4. Use Cases and Scenarios: Blueprint and a script illustrating how users interact with a system. Use-cases describe system's functionalities, while scenarios provide detailed narratives of specific interactions. Identifying the main actors, the uses-cases, and detail the use-case scenarios. Best practices. - workshops, iterative refinement, integrate with other techniques

Requirements Need to be Engineered: Requirements Prioritization

What is Requirements Prioritization:

1. The process of ranking the requirements, based on their importance or criticality, to ensure that the most critical requirements are addressed first in the process.

1. It helps stakeholders take informed decisions about the scope and schedule of the project.

Requirements Prioritization can be achieved by multiple ways:

1. MoSCoW Prioritization: Prioritizes the requirements into four categories - "Must Have", "Should Have", "Could Have, and "Won't Have".
2. Kano Prioritization: It’s a prioritization technique where requirements are prioritized based on customers' satisfaction using a two-dimesional graph.
3. Value-Based Prioritization: Prioritizes the requirements based on their business value, using a cost-benefit analysis.

4. And more ...

Requirements Need to be Engineered: Requirements Prioritization

1. **MoSCoW Prioritization Technique:**

M: Must Have – Non-negotiable requirements that are mandatory to be implemented.

S: Should Have – Important requirements that are not vital but add significant value.

C: Could Have – Nice to have requirements that will leave little impact if not implemented.

W: Won't Have – Requirements that are not needed to be implemented as of now.

What Are the Best Practices for using MoSCoW Prioritization?

1. Choose an objective scoring system for requirements.
2. Seek input from all the stakeholders involved.
3. Share your Moscow progress with the entire organization.

Requirements Need to be Engineered: Requirements Elicitation Output

1. SRS document; usually in SDLC approaches
2. User Stories; in Agile models.

- Team divides the work into functional increments known as user- stories.
- Usually written in physical form or in digital platforms like JIRA.

Role-Action-Benefit Template:

- Usually in the form of ”AS A ROLE… I WANT TO ACTION… SO THAT BENEFIT”.
- As a role – Who wants to accomplish something.
- I want to action – What they want to accomplish.
- So that benefit - Why they want to accomplish that thing.

E.g.: As a bank customer, I want to withdraw money from ATM, so that I am not constrained by opening hours or spend my time inefficiently in the long queues.

Requirements Need to be Engineered: Requirements Elicitation Output

1. SRS document; usually in SDLC approaches
2. User Stories; in Agile models.

Consider using the INVEST approach:

I: **Independent** – each user story is independent and atomic in nature.

N: **Negotiable** – user stories is never absolute. It should allow a discussion/negotiation between the developers and the customers or other stakeholders.

V: **Valuable** – every user story must add some value to the product.

E: **Estimable** – the developer must be able to understand and have an estimable idea before implementing it.

S: **Small** – The stories must be small enough to be completed in one sprint. If it not small enough, we got to break it down further.

T: **Testable** – Must have a valid acceptance criteria to determine its completeness. It should be objective, specific, testable and measurable.

Requirements Engineering: ACTIVITY 1

Phase 1:

Step 1: Select a project / problem statement of your choice.

Step 2: Each team member can assume the role of one stakeholder – could be anyone – managers, customers, developers etc. BUT NOT AS "USERS"

Step 3: Use the requirements elicitation techniques (preferably interviews OR/AND JAD) to elicit the requirements for your project.

Step 4: Document those requirements in two ways – SRS and User Stories.

Phase 2:

Step 5: Each team then proceeds to validate the elicited requirements with the other team that will assume the role as potential "users" of the product or the service.

Step 6: Prioritize the requirements based on 'MoSCoW Prioritization' technique.

User-Centered Design and User Experience

UCD process outlines the phases throughout a design and development life cycle all while gaining a deeper understanding of **who will be using the product** (i.e. the system).

User Experience (UX) is a person's perceptions and responses that results form the use or the anticipated use of a product, system or any service.

Is it '**useful**', is it '**usable**' (i.e. user friendly) and is it '**delightful**' (i.e. makes the user satisfied/happy/effective).

User-Centered Design and User Experience

UX Vs User Interface (UI):

- **UI** focuses on anticipating what users might need to do and making sure that the interface has the elements that are easy to access, understand and use to facilitate those actions.
- UI brings together concepts and ideas from interaction design, visual design and information architecture.
- We can understand as UCD encompassing the everything of the design, nested within is the UX and then within UX is the UI aspects.
- What is UI/UX – In an over simplified manner Low Fidelity Wireframes (UX) – high fidelity designs (UI).

- **Interaction Design (IXD**): focuses on creating engaging interfaces with well and effective can behaviors. i.e., understanding how users and technology communicate with each other.
- **Information Design (IA):** it focuses on organizing, structuring and labelling content in an effective and sustainable way. The goal is to make sure the users find the information and can complete the task.

Design Thinking: An Introduction

DT is an iterative process in which we seek to understand the user, challenge assumptions, and redefine the problems, identify alternative strategies and solutions that might not be instantly apparent with our initial level of understanding.

DT Aspects:

- **Empathy for the person/users**: Who is going to use the product/service, what are their motivations, What benefits will they receive.
- **User-Centric problem solving**: What are their goals, what tasks do they complete, What will your product solve for them.
- **Make Data-Driven decisions**: Conducting research to identify the features that your users/customers want.
- **Interdisciplinary nature**: Different perspectives from different knowledge domains and teams.
- **Have focused goals**: Are the goals aligned to address the challenges.
- **Clear Communication**: Keeping team members and stakeholders informed about the decisions and progress (i.e. in Agile approaches).

This can be summarized as follows:

Empathize → Define → Ideate → Prototype → Test (this is not a linear process though!!).

User-Centered Design Techniques

1. **PERSONAS**: Identify the key target audience and draft their personas.

Personas are focused on: **Tasks** (When, where and how); Goals (What are they trying to achieve after using the product successfully) and **Motivations** (The Why? Why should a person perform these tasks).

ACTIVITY 2: Create Personas

1. Identify one role or target audience you would like to focus on (or two?)
2. Discuss and identify 1-3 workflows that are typical for that role.

User-Centered Design Techniques

2. **UX One-Sheet**: Vision Statement, situation, design criteria, lifecycle phases and metrics.

Simplified view: Strategy → Concept → Design → Implement → Deploy and Measure (towards success)

ACTIVITY 3: Create UX One Sheet

To Think: What's the story for your user / target audience?

Sections:

1. Vision statement: Align the vision statement for the project to the vision of the organization (i.e. WHY)
2. Situation: Who, What, When, Where, Why, and How.
3. Design criteria: How the UX should feel. Base these on the vision statement
4. Lifecycle phases: Phases and the steps the user will go through as they experience this solution.
5. Metrics: What is the criteria to measure the success? How ill you measure if the users are successfully using the system?

User-Centered Design Techniques

3. **Journey Mapping**:

- Tells the story of the experience for your target audience or customers.
- Includes all the touchpoints from the initial contact through the process of engagement and into long term relationships.
- Process of ‘mapping’ with your teams, offering rich shared insights.

ACTIVITY 4: Create Journey Maps

- Helps to bring the target audience at the forefront.
- Helps to identify the gaps.
- Synchronizes you and your customers.

User-Centered Design Techniques

3. Wireframes: A two-dimensional illustration of an interface that specifically focuses on space allocation and prioritization of the content, the available functionalities and the intended behaviors.

Wireframes are focused on: Elements, organization of those elements in an interface, and the interactions between those elements (i.e. how users are going to interact with the interface(s)).

- To capture ideas quickly, using simple diagrams.
- Wireframes are usually the blueprints. Used to get a clear and high-level idea of the product.

- Focus on placement of elements within an interface. No images or high-level graphics.

- Strive to create a simulation of the experience by creating low-fidelity mockups – quick to create, less detailed and abstract designs that uses simple images and line diagrams.

ACTIVITY 5: Design Low-Fidelity Wireframes

UX UI Design using FIGMA

What is FIGMA and what does it do?

- Figma is a web-based design tool used for prototyping, interface design, and collaboration, Basically, UI design.
- Helps to create low-fidelity wireframes.
- Figma is used to create user interfaces for websites and mobile apps.
- Receive feedback from the customers and clients and include changes, if any. i.e. user-testing.

It does not 'code' the website or the app.

Goto: figma.com

Create your own account and LogIn.

Use the FREE version, for now.

UX UI Design using FIGMA

1. Create a 4-5 steps workflow of ANY system; preferably based on the requirements you elicited. E.g., An app to sell certain product:

Workflow example: Create a homepage/marketing page --> Product Features --> Checkout/Cart --> Confirmation.

1. Choose one (or maximum) two persona developed in the previous activities.

2. Play with the tool. Explore as much as you can.