UNIVERSITÀ DEGLI STUDI DI NAPOLI FEDERICO II

SOFTWARE ENGINEERING – LECTURE 03

Requirements Engineering: Elicitation And Analysis

Prof. Luigi Libero Lucio Starace

luigiliberolucio.starace@unina.it

https://luistar.github.io

https://www.docenti.unina.it/luigiliberolucio.starace

Previously, on Software Engineering

- We've discussed what Software Engineering is and why it's important
- We've talked about **Software Processes** and **SDLC**
  - We had an overview of the **Waterfall** Software Process Model
- We've seen the many aspects of **Software Quality**
- Today, we'll get into Requirements Engineering

The Software Life Cycle

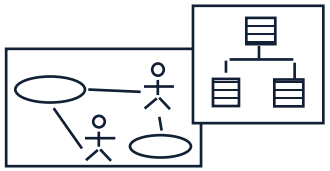
Requirements Engineering → System Design → Software and UI/UX Design → Implemen-tation → Testing → Operation and Maintenance

Requirements collected via:

- Interviews with Stakeholders
- Personas
- Stories and Scenarios

Specified using:

- Use Cases
- Natural Language
- Domain Models
- Mock-ups



Define System Architecture

- Requirements are allocated to software sub-systems
- Sub-systems are allocated to hardware resources
- Architectural Patterns

Define Subsystems

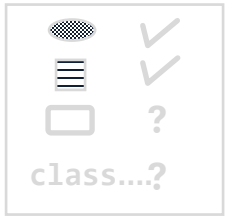
- Objects required to realize each subsystem are defined.
- Software Design Patterns
- Usability Engineering
- High-fidelity Wireframing

Each Subsystem is implemented

- Source code and other artifacts
- Clean Code
- Frameworks and ORMs
- Focus on Software Quality

Ensure the Software satisfies customers

- Code inspections
- Functional Testing (unit, integration, system testing)
- Usability Testing



System is put into practical use

Maintance will be required at some point

- To fix errors that were not discovered in previous phases
- To adapt the software to changes in requirements on in its environment

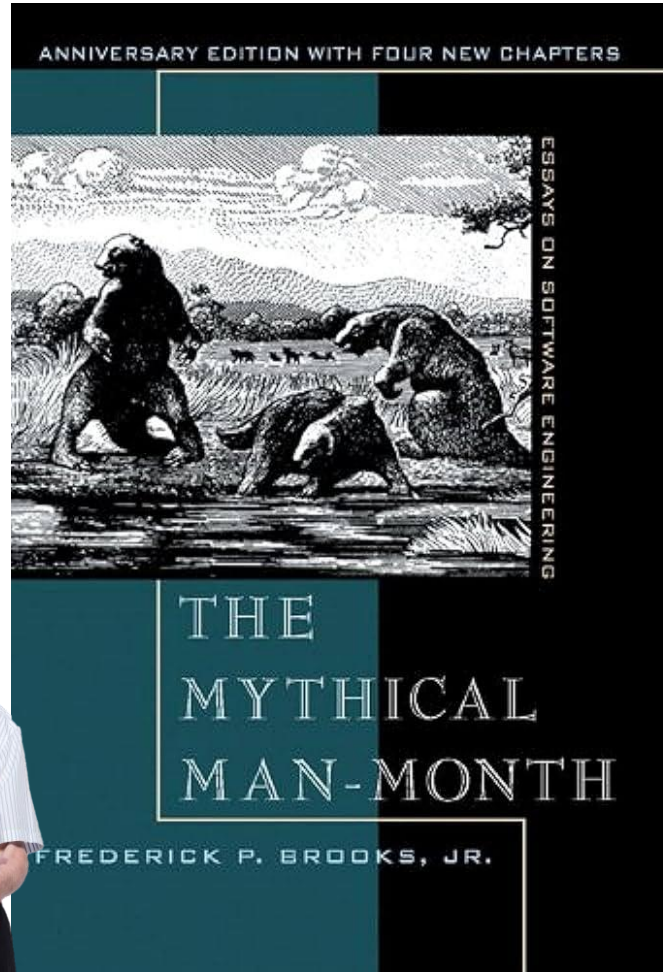
Fred Brooks on Requirements Engineering

The hardest single part of building a software system is deciding precisely ***what to build****.*

No other part of the conceptual work is so difficult as establishing the detailed technical requirements [...].

No other part of the work so cripples the resulting system if done wrong. No other part is more difficult to rectify later.

- Fred P. Brooks



Software Requirements

- Requirements are descriptions of what the system should do:
  - **services** that the system should provide to its users
  - **operational constraints**
- Some examples:
  - The system shall allow students to search for courses based on keywords
  - The system shall allow students to enroll in courses
  - Users interfaces shall be implemented as native Android and iOS apps
  - The system shall handle at least 10k concurrent students
  - The system shall authenticate users also by using built-in fingerprint sensors

Requirements: Different Levels

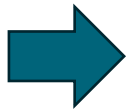
The term «**requirement**» is used inconsistently in the Software Industry

- In some cases, a requirement is an abstract, high-level description of a service the system should provide or a constraint on the system.
  - These are called **User Requirements** (focus is on end users' perspective)
- In others, it is a more detailed and formal definition of a system function
  - These are called **System Requirements** (focus is on the system to be built)

Requirements: Different Levels

User Requirement

1. The system should generate a monthly course enrollment report



System Requirements

1.1. Course enrollment reports should be generated on the last working day of each month

1.2. A report should be generated for each course, displaying the total number of enrollments in the current month and the average age of the enrolled students

1.3. Access to the reports should be restricted to course managers

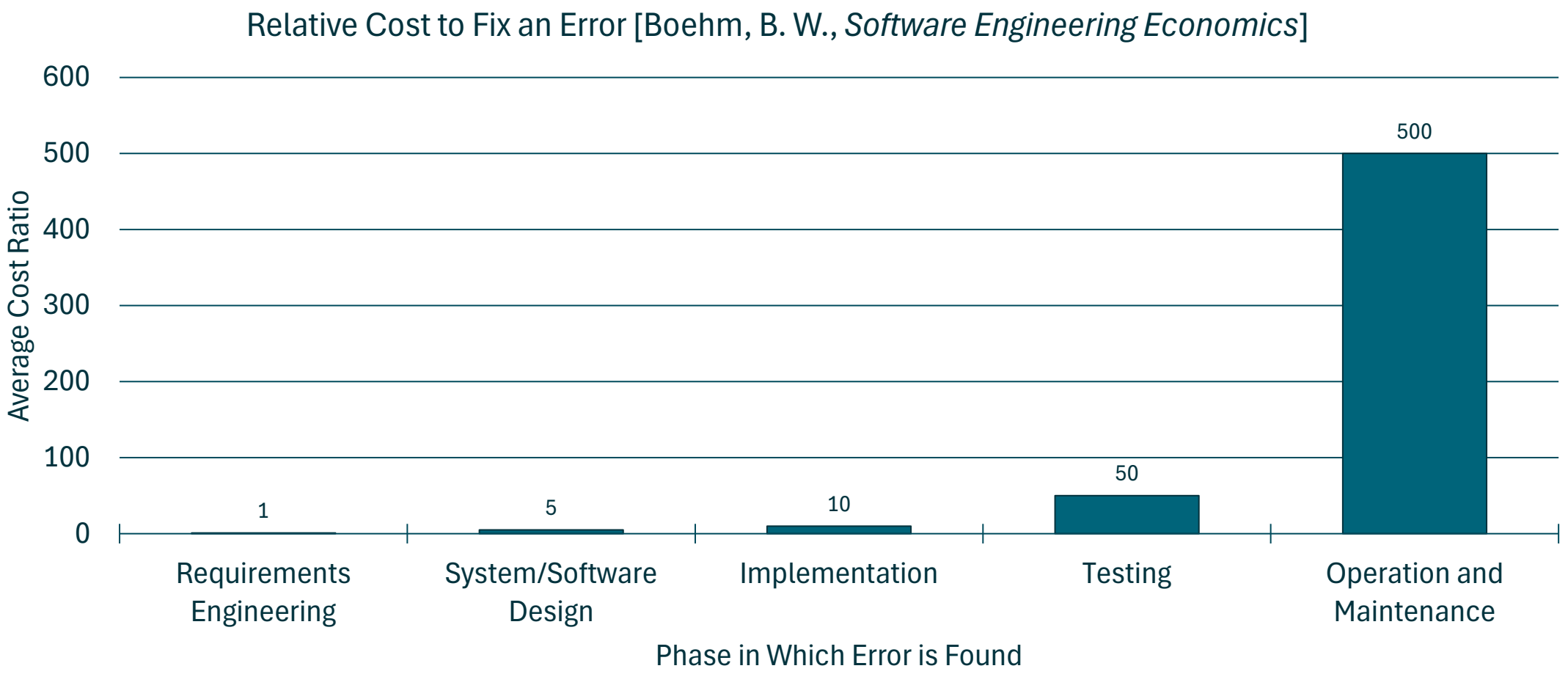
1.4. ...

Requirements: Different Levels

Such ambiguitiy is inevitable, as requirements may serve a dual function:

- **User Requirements** may be the basis for a bid for a contract
  - Customer may define general user requirements
  - Different contractors may propose different ways to meet the user requirements
- **System Requirements** may be the basis for the contract itself
  - Once a contract to develop the software has been awarded, the contractor formulates a more detailed set of System Requirements
  - So that the client understands exactly what the software will do, and can validate the proposal
  - Once accepted and validated, System Requirements can be put in the final contract and are binding!

Cost of Errors Through Project Lifecycle



Requirements Engineering (RE)

- RE is a sub-area of Software Engineering dealing with the process of defining the requirements for a software-to-be
- **Goal**:
  - Provide Software Engineers with methods, techniques and tools to understand and document what a software system must do

Functional and Non-functional Requirements

Requirements are often classified as functional or non-functional

- **Functional Requirements:**
  - Services the system should provide
  - How the system should react to particular inputs or behave in a given situation
- **Non-functional Requirements:**
  - **Constraints** on the services or functions offered by the system
  - Include timing contraints, process constraint, or standard-imposed constraints
  - Often, apply to the whole system rather than individual features or services

Functional and Non-functional Requirements

In practice, the distinction between these two types of requirements is not cut-and-dried. Consider the following:

- *Only authorized users should be able to access the system*

- Seems like a non-functional requirement
- However, when developed in greater detail, it generates additional requirements that are clearly functional
  - E.g. *Users shall be able to perform the login and authenticate themselves*
- **Requirements are not independent and one requirement often generates or constrains other requirements**

Functional Requirements

- When expressed as **functional user requirements**, they may be written in natural language, so that they can be undestood by non-technical people (e.g.: users and managers)
- When expressed as **functional system requirements**, they should be detailed, accurately describe system inputs and outputs and exceptions, so that software engineers know exactly what to implement

Non-functional Requirements

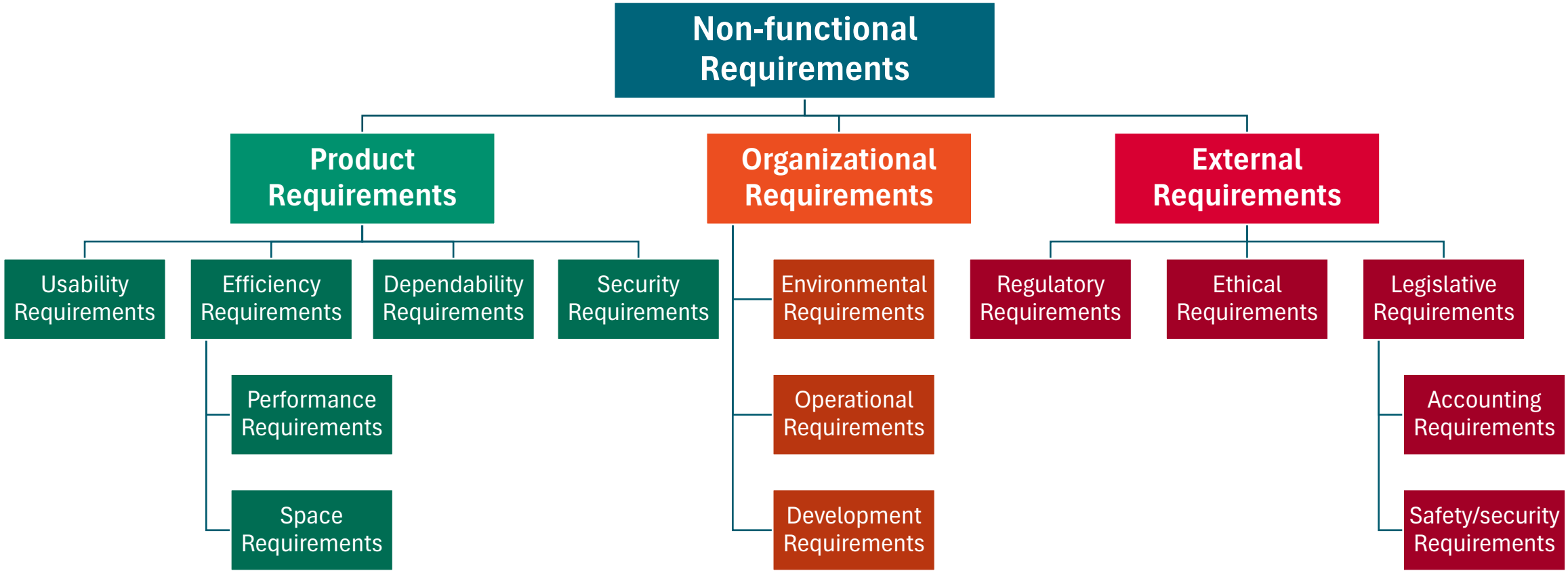
- Non-functional requirements are not directly concerned with specific services offered by the sytem
- Typically specify or constrain characteristics of the system as a whole
  - Constrains on how it should be implemented (e.g.: must use the Java language)
  - Specify other properties (e.g.: response times, memory use, ...)
- Often non-functional reqs are more critical than functional ones
  - Users may find a way around a sub-optimal implementation of a functional req..
  - ..but failing to meet a non-functional req may mean the system is unusable
    - A system that is not compliant with the GDPR cannot be deployed in Europe!

Types of Non-functional Requirements

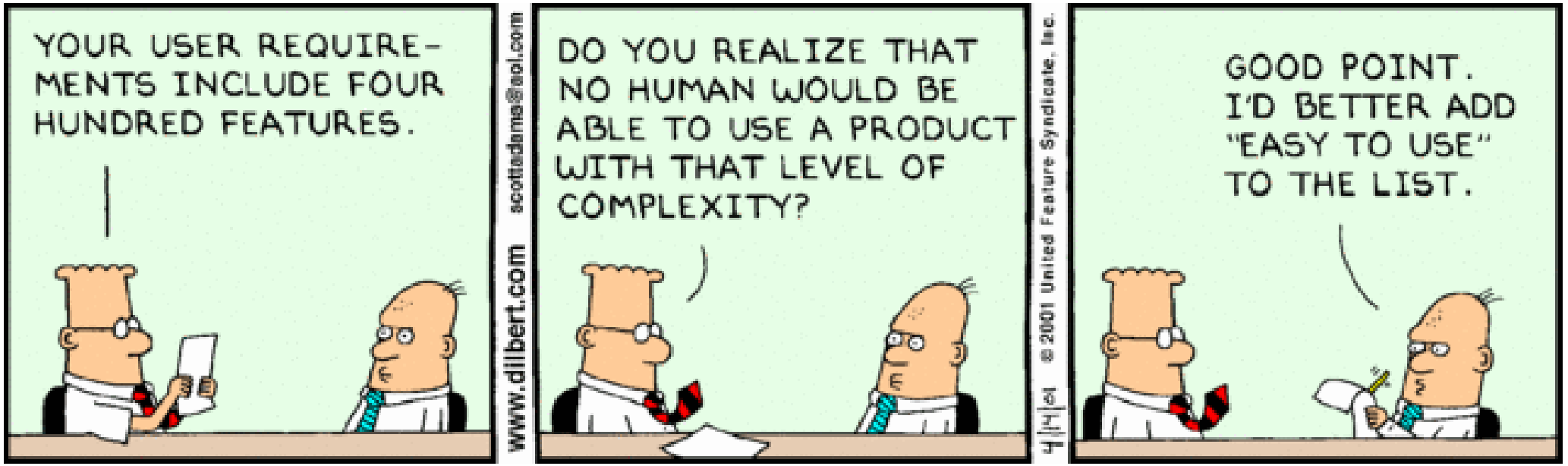
Non-functional requirements may arise from:

- Required characteristic of the product (**Product Reqs**)
  - How fast the system should be, how much memory it should require, usability requirements, acceptable failure rates, ...
- The customer's and the developers' organizations (**Organizational Reqs**)
  - Operational processes describing how the system will be used, required programming languages, requirements specifying the operational environment, ...
- External sources (**External Reqs**)
  - What must be done for the system to be approved by a third-party regulator, legal requirements, ethical requirements to ensure that the system will be acceptable to its users and the general public

Types of Non-functional Requirements



Dilbert on Usability Requirements



Domain Requirements

- **Domain requirements** are derived from the particular purpose or industry context in which a software is used
- Requirements for the control software of a medical device may include:
  - *The system safety shall be assured according to standard IEC 60601-1:Medical Electrical Equipment – Part 1: General Requirements for Basic Safety and Essential Performance.*
- The above requirement constrains both the design of the system and the development process.
- A perfectly functional software that was not developed according to the standard could not be used in practice!
- Domain requirements may be functional or non-functional.

Properties of Good Requirements

Requirements should be:

- **Clear** and **easy to understand** (especially User Requirements)
- **Unambiguous** (ambiguity leads to disputes with customers)
- **Complete**
- **Consistent** (i.e., should not conflict with each other)
- **Testable** (lack of testability leads to disputes with customers)
  - Given a requirement, it should be possible to **unambiguously determine** wheter the system satisfies that requirement

The Demons of Ambiguity

Consider the traffic sign on the right

- To us, its intent is clear: keep kids safe in a school zone
- How would we explain it to a computer?
  - Why does the sign use the plural "*children*" instead of the singular "*child*"? Does the rule only apply if more than one child is present?
  - Who qualifies as a "child"? Is a 17-year-old a child?
  - What does it mean to be "present"?
  - Does this rule apply only when school is in session?

H. Robinson, The demons of ambiguitiy. Full article archived here.

The Demons of Ambiguity: Example

Requirement: The utility shall allow users to start/stop a service on a remote machine

- What if we start a service that's already running? Or try to stop a service that is not running?
- What if the remote machine reboots? Should the service restart?
- What if the connection drops?
  - Shall the utility display an error message?
  - Shall it try again?
    - After how much time? How many retries should be performed at most?

Driving out the Demons of Ambiguity

- Dissect the description. Are the words open to different interpretations?
  - "*Mary had a little lamb*"
    - Mary kept a little lamb as a pet?
    - Mary ate an entire small lamb?
    - Mary ate a small portion of lamb?
- Poke at the borders. Terms often map imperfectly onto each other, so find the fuzzy areas.
  - In the remote service utility example, how will we check that a service is running? Are we checking a flag somewhere? Could that flag be improperly set even when the service is not running?

Illustration by William Wallace Denslow (1902)

Requirements Testability

- It should be possible to **unambiguously determine** whether the system satisfies a requirement
- Otherwise, developers may argue that the requirement is satisfied, while the customer may not agree at all!
- System Functional Requirements should be as detailed as possible and leave no room to interpretation
- System Non-functional Requirements should include **quantitative** indicators whenever possible

Testable Non-functional Requirements

Not so testable requirements:

a. The system shall be reliable

b. The system shall be easy to use

c. The system shall be fast and reactive

More testable requirements:

a. The system shall have a monthly 99.9% uptime

b. Users shall be able to use all system functions after at most two hours of training. After the training, the average number of user errors should not exceed two per hour of system use.

c. The average response time of the system shall be no longer than 100ms

Metrics For Non-func. Requirements

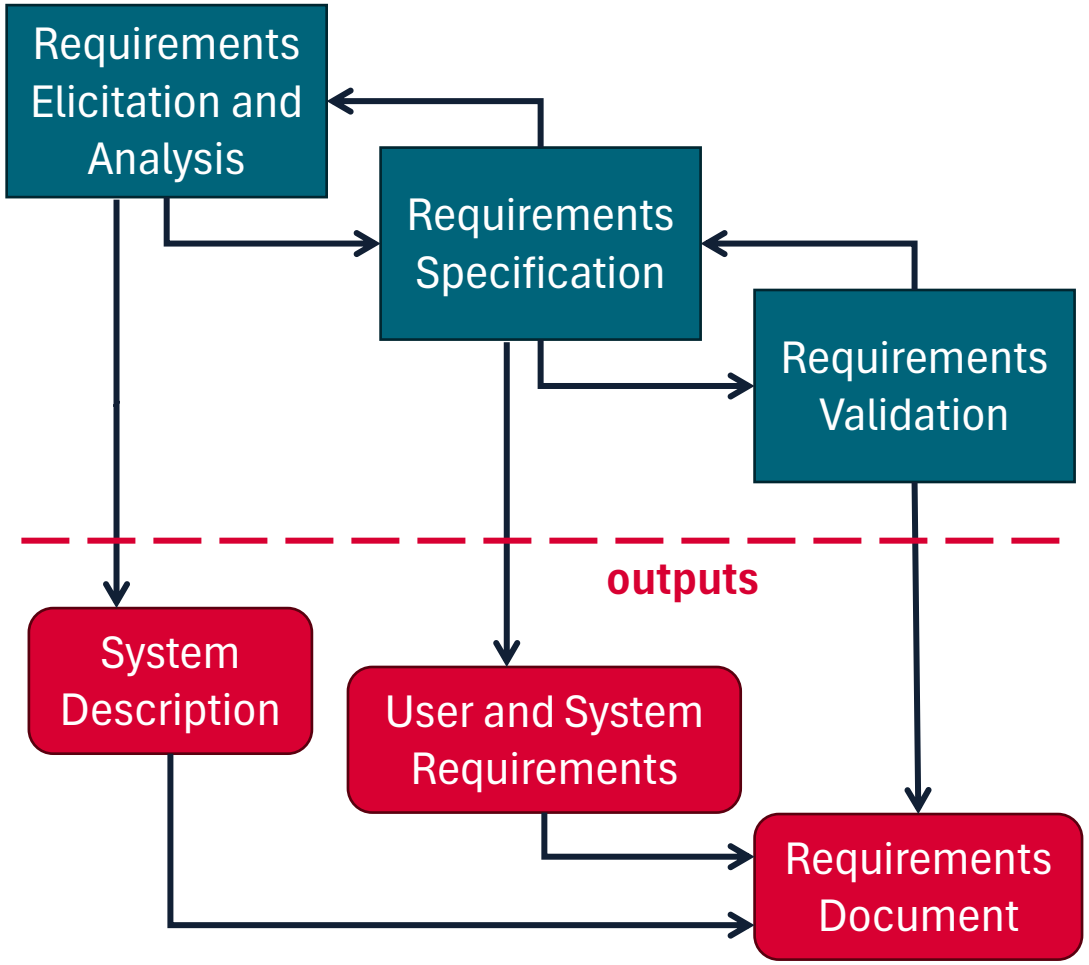
Property	Metrics
Performance	Processed operations / second User / event response time Screen refresh rate
Size	Megabytes
Ease of use	Required training time User error rate Number of requests for support
Reliability	Time to failure Availability rate (e.g.: uptime)
Robustness	Time to recover after failure Probability of data loss on failure

Requirements Engineering Processes

The Requirement Engineering Process

Three key steps:

- **Requirement Elicitation and Analysis**: Discover requirements by interacting with stakeholders
- **Requirement Specification**: Converting the requirements in a standardized form
- **Requirement Validation**: Checking that the requirements actually define the system the customer wants

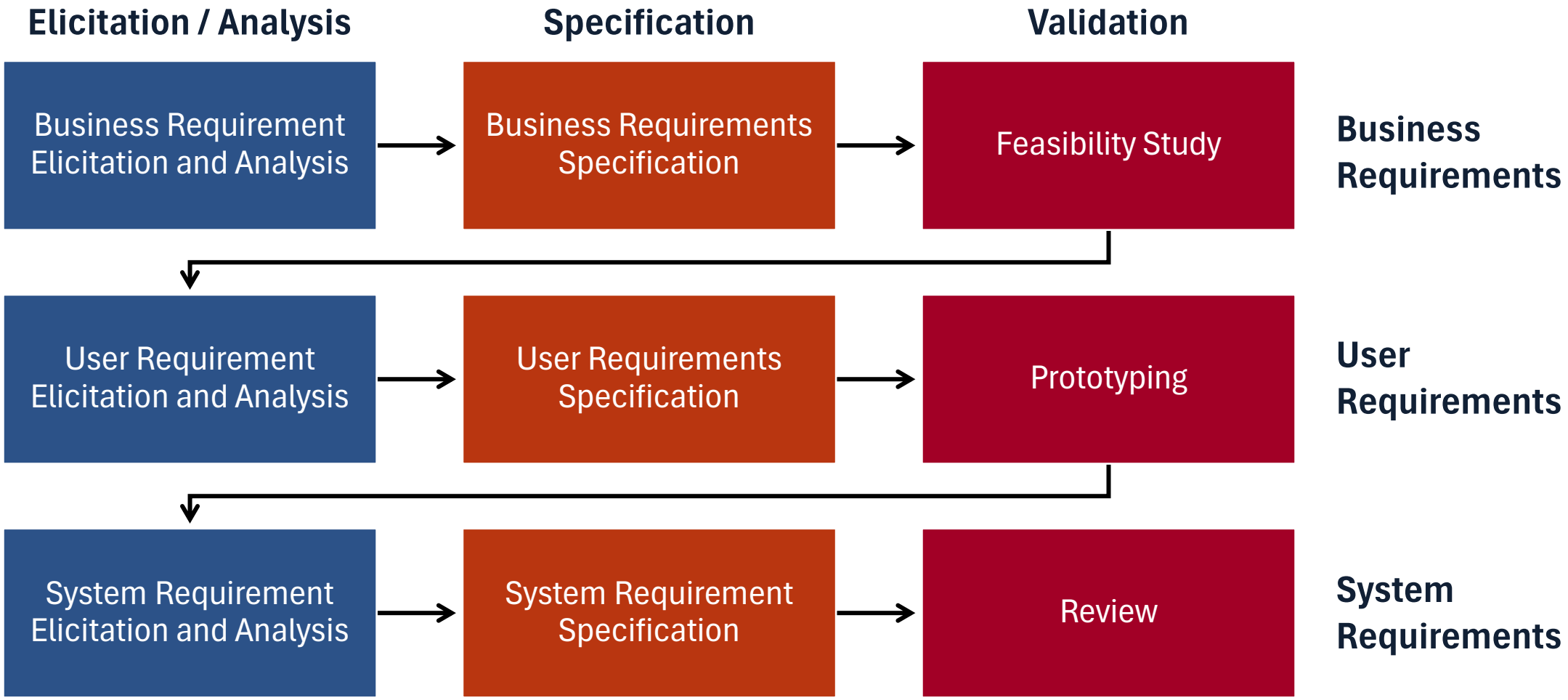


The Requirement Engineering Process

In practice, the RE process is not linear. Its activities are actually interleaved in an **iterative process** at different granularity levels

- In the first iteration, we focus on **Business-level Requirements**
  - Why is the project needed? Who benefits from it? When and where will it take place?
- Then, we define **User Requirements**
- Finally, we define **System Requirements**
- Each iteration includes elicitation and analysis, specification and validation phases

Iterative Requirement Engineering Process



Requirements Elicitation And Analysis

Requirement Elicitation and Analysis

- The goal of this activity is to find out what the customers want and need
- That's easier said than done!

Requirements Elicitation And Analysis

Requirement Elicitation is considered as the most critical part in RE

- Involves collaboration between different **stakeholders**
  - **Stakeholders** are the people or groups affected in some way by a software development project
    - End-users of the software. This may be an heterogeneous group. If the software-to-be is a order management system, users may be members of the sales department that put in the orders, members of the logistics departments that need to process the orders and deliver the items, or members of the customer care department.
    - People that use the outputs of the software
    - Client management and executives
    - Client technical staff, that may need to operate the system on a day-to-day basis

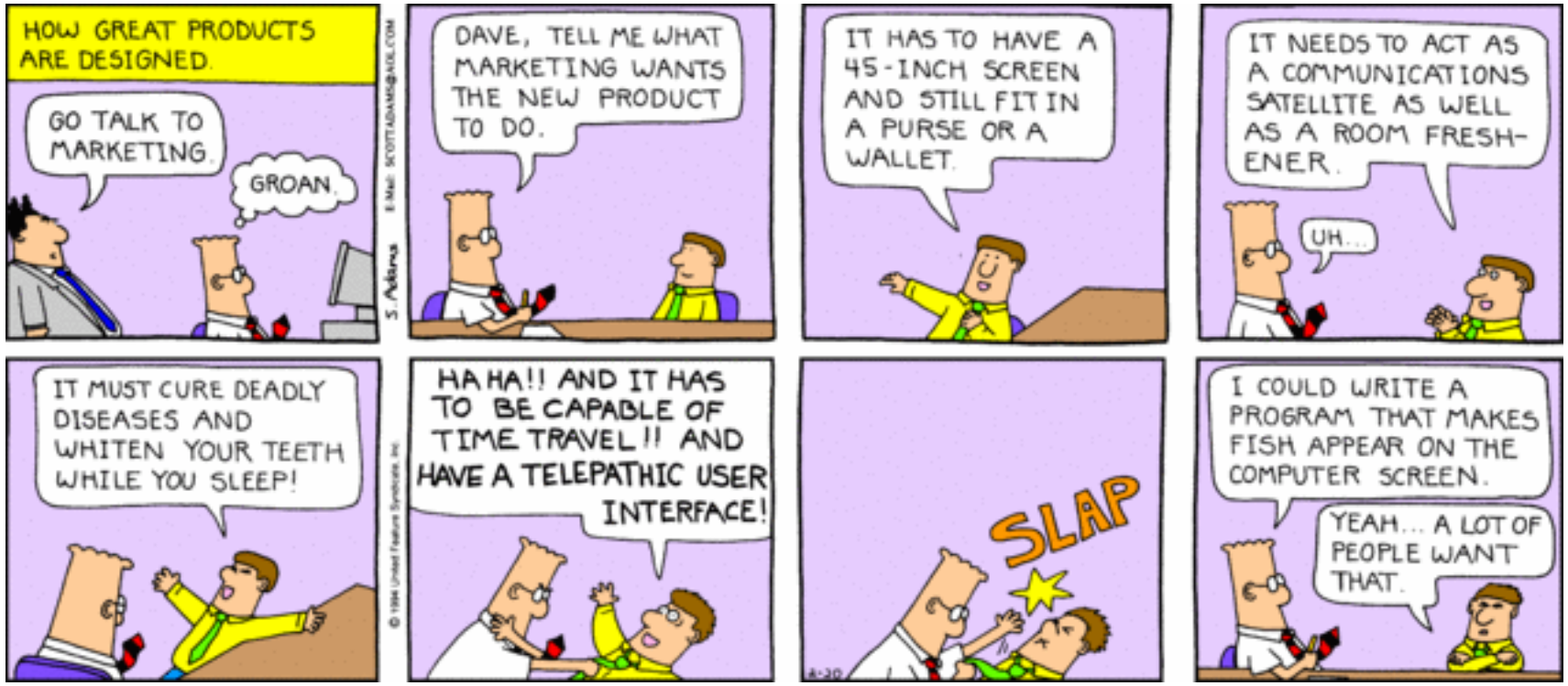
Requirement Elicitation: Challenges (1/2)

- Stakeholders **do not know what they want** from the software, except in the most general terms. They do not know what is feasible and what isn't, and **may make unrealistic demands**.
- Stakeholders are experts in their own domain. They naturally express requirements in their own terms and jargon, with implicit knowledge of their work. They may give some details for granted, when they're not.
- Different stakeholders, with different requirements, may express their requirements in different ways. Requirement Engineers need to work their way around **commonalities** and **conflicts**

Requirement Elicitation: Challenges (2/2)

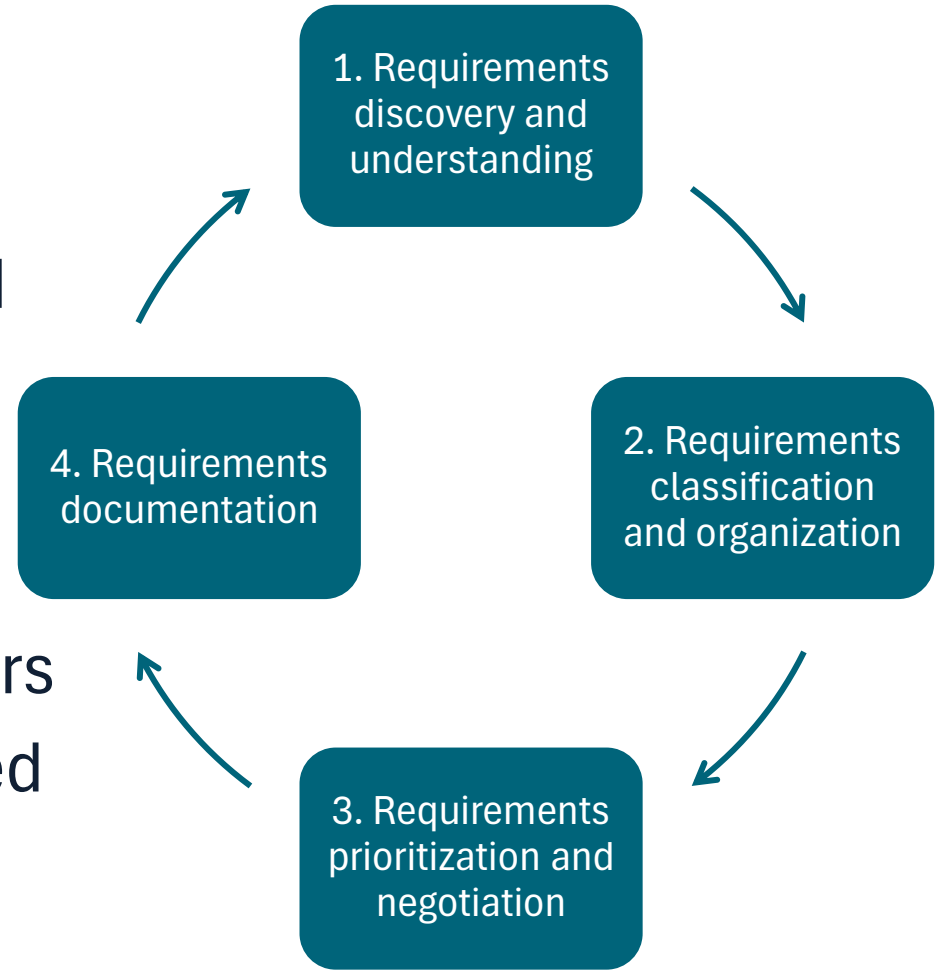
- Political factors may influence the elicitation process. Managers may demand specific requirements because these would allow them to increase their influence in the organization.
- Stakeholders have different (and sometimes **conflicting**) views on the importance and priority of requirements. If some stakeholders feel that their views have not been properly considered, then they may deliberately attempt to undermine the RE process.
- Requirements **will change** during the elicitation process. New requirements may emerge from new stakeholders that were not initially considered.

Dilbert on RE Challenges



Requirement Elicitation And Analysis Process

- **Discovery and understanding**: interact with stakeholders to discover requirements
- **Classification and organization**: related requirements are grouped together and organized
- **Prioritization and negotiation**: solve requirements conflicts arising from conflicting needs of different stakeholders
- **Documentation**: keep track of discovered requirements for the next iteration of the elicitation process



Requirements Elicitation Techniques

Different approaches can be used and combined to discover requirements

- Interviews with stakeholders
- Observation and Ethnographic Studies
- Definition of Personas
- Definition of Scenarios and User Stories
- Definition of mock-ups and prototypes

Interviews with Stakeholders

Formal or informal interviews with stakeholders are fundamental in RE

- During interviews, the RE team asks the stakeholders about their work, the system they currently use, and the system-to-be-developed
- Requirements are derived from the answers to these questions
- Interviews may be:
  - **Closed interviews**: stakeholders answer a pre-defined list of questions
  - **Open interviews**: there is no pre-defined agenda
  - In practice, interviews are usually a combination of both approaches. Completely open-ended interviews rarely work well. It's better to start from a set of questions defined in advance, which usually lead to other issues that are discussed in a less structured way.

Interviews with Stakeholders

- People generally like talking about their job, and take part in interviews with a positive attitude
- However, stakeholders often have their own job to do, you should not waste their time with useless meetings, and should try to minimize the impact of interviews on their job
- Challenges:
  - Stakeholders use jargon and give some requirements for granted
  - Stakeholders know about their job, they may have a partial or distorted vision of the job of other colleagues in different areas or departments
  - Stakeholders may be reluctant to discuss organizational requirements and constraints due to subtle power dynamics in the organization

Real life story time

- Back in my days, I did my curricular internship for the B.Sc. in Computer Science at the **Central Direction for Financial Services** at the **Municipality of Naples**
- Developed a full-stack web app to support the management of a minor tribute (approx. tax revenue: € 1-2 M per year). Before, the tribute was managed mostly via Excel spreadsheets.
- Project started from scratch, so – being a good SE student – I started with Requirement Elicitation and interviewed some stakeholders

Real life story time

Signor Catello, la ringrazio innanzitutto per la sua disponibilità. Mi racconta come funziona, dal suo punto di vista, la gestione di questo tributo?

Certo! È molto semplice! Innanzitutto, un operatore sul campo mi invia la lettura per un dato anno per un certo contribuente.

Quindi, inserisco la lettura in una maschera che calcola l'imponibile. Successivamente, faccio la stampa delle ingiunzioni di pagamento, che vengono poi emesse.

Una volta notificata l'ingiunzione, il contribuente ha 60 giorni per pagare. Se non paga entro il termine, devo procedere all'iscrizione al ruolo coattivo. Per l'iscrizione, inserisco i dati in una maschera che genera il Tracciato 290 da trasmettere all'ente di riscossione.

Ethnography

- Software rarely exist in isolation, and is used in a social and organizational environment
  - Requirements can be generated or constrained by such environment
  - Often, the organization and processes that are **actually** used in an organization are different from the formal processes that are supposed to be used
- **Ethnography** is an observational technique that can be used to understand operational processes
  - A requirement engineer immerses themselves in the working environment of end users, where the system will be used
  - Day-to-day work is observed, taking notes of the actual tasks and participants involved

Ethnography

Ethnography is particularly effective for discovering:

- Requirements derived from the way in which people actually work, rather than the way in which business process definitions say they ought to work.
- Requirements derived from cooperation and awareness of other people's activities.
  - E.g.: Before sending a customer feedback survey, a Customer Care employee checks with his colleagues in the logistics department that the order has actually been delivered to that customer.
  - "Customer care employees shall be able to check whether a customer has already received an order"

Personas

- Knowing the user plays an important role in discovering requirements and (we'll see later on) in designing effective user interfaces
- Sometimes we know the users (e.g.: when we develop a tool for other software engineers)
- In general, software is developed for other people, with different needs
  - As Software Engineers, you should be aware that the actual users of our software are different from you, and that your own experience may not provide a complete picture of what they might want and how they might work.
- **Personas** are a powerful tool to gain a comprehensive understanding of the users, and promote empathy.

Personas

- **Personas** (sing. Persona) are **hypothetical archetypes** of actual users.
- Personas are not real people, but they represent them.
- Although they are imaginary, they are defined with rigor and precision.
- They’re not completely “made up”, but rather **discovered** as a byproduct of the requirement elicitation process.

Personas definition

- There is no standard way to represent a Persona
- Commonly used features include:
  - **Personalization**: name, age, short bio, …
  - **Job-related information**: What's the individual's job?
  - **Education**: Details of their education and experience
  - **Goals**: What is their interest in the software?
  - **Frustations/Pain points**: Is there any pain points the software might help address?

Personas: examples

Dwight Schrute

Age: 35
Location: Scranton, PA
Marital status: single
Job title: Assistant Regional Manager (self-proclaimed), Assistant to the Regional Manager, Salesman, Dunder Mifflin Scranton Branch
Personal traits: loyal, competitive, ambitious

Goals:

- Improve efficiency and accuracy in managing orders.
- Ensure seamless integration with existing sales processes.
- Prove his superiority in sales and other aspects of office life.
- Gain recognition for contributing to the company's success.

Interests:

- Agriculture: Runs a family beet farm, Schrute Farms.
- Security: Maintaining strict order and security protocols.
- Sales Performance: Focused on achieving top sales and proving his superiority

Bio:

Dwight is a dedicated and hardworking senior sales representative at Dunder Mifflin. His unique personality and intense approach to his work make him a valuable, though unconventional, team member. He is skeptical towards the adoption of new technology, but is proficient with traditional office productivity applications. His ambition and commitment drive him to support the development of a new order management system that will enhance his sales efficiency.

Personas: examples

Micheal Scott

Age: 45
Location: Scranton, PA
Marital status: single
Job title: Regional Manager, Dunder Mifflin Scranton Branch
Personal traits: optimistic, attention-seeking, childlike and naive

Goals:

- Create a fun and engaging work environment.
- Improve team morale and cohesion through better order management.
- Enhance customer satisfaction and business performance.

Interests:

- Comedy: Loves quoting movies and TV shows
- Social Activities: Enthusiastic about office parties
- Personal Relationships: Focused on finding love and maintaining friendships

Bio:

Michael Scott has been with Dunder Mifflin for many years, rising to the position of regional manager. Known for his enthusiastic and creative approach, Michael is committed to making the office a fun place to work while achieving business success. He is not very tech-savy, and gets easily frustrated over complex user interfaces. His optimistic outlook and genuine care for his team drive him to support the development of a new order management system to enhance efficiency and customer satisfaction.

Personas: examples

Toby Flenderson

Age: 43
Marital status: divorced
Job title: Human Resources Representative, Dunder Mifflin Scranton Branch
Personal traits: reserved, calm, introverted, patient

Goals:

- Ensure company policies and procedures are followed.
- Improve the workplace environment and employee satisfaction.
- Enhance customer satisfaction and business performance.

Interests:

- Writing: Enjoys writing fiction and is an avid reader
- Social Activities: Enthusiastic about office parties
- Family: Values time spent with his daughter

Bio:

Toby Flenderson has been with Dunder Mifflin for several years as the Human Resources representative. His calm and empathetic nature makes him a reliable mediator and support system for employees. Toby's attention to detail and dedication to compliance ensure that company policies are upheld. He is a proficient office applications user. He supports the development of a new order management system to streamline operations and improve the overall workplace environment.

Personas in RE

Sometimes, we do not even have specific stakeholders to interview!

- For example, when we're developing an off-the-shelf software for the general public
- When we're developing a custom-made software for a customer, and some of the end-users are the general public

- In these cases, Personas can be a powerful tool to start discovering requirements

Example: dental practice management system

Suppose we are doing Requirement Elicitiation for a software to manage appointments at a dental practice. By interviewing with the staff, we discovered the following requirements:

- Patients shall be able to signup and login into the system
- Patients shall be able to book an appointment in a free slot
- Doctors shall see the scheduled appointments for the current week
- Receptionists shall be able to delete appointments
- Patients shall receive an email reminder the day before their appointment
- Patients shall receive an email reminder if their appointment is cancelled

Personas: example

Concetta (Tina) Aiello

Age: 68
Marital status: widowed
Job: Retired
Personal traits: assertive, impatient

Bio:

Tina is a retired primary school teacher, with more than 40 years of work experience. She is very sweet and patient around children, but she is short-tempered and impatient when dealing with things outside her comfort zone. She loves following her daily routine and doen't like changing it. Doesn't really like technology. Last year, her sons gifted her her first smartphone with an internet plan. To date, she only uses it to regularly video-call her grand-daughters. She is generally skeptical around the internet, and perceives it as a dangerous place.

Interests:

- Knitting: Is a skilled knitter, loves making sweathers for her grand-daughters
- Local church: Devout catholic and involved in her local church community
- Family: Values time spent with her grand-daughters

Goals:

- Book regular appointments to check on her dental prosthetics
- Get a reminder so that she does not miss her appointments
- Change the date of an appointment if conflicting with another event

New requirements arise from the Persona

- Receptionists shall be able to create appointments for patients that called the dental practice to schedule an appointment
- The day before an appointment, receptionists shall receive a notification telling them to call patients that scheduled an appointment via telephone to remind them of the appointment.

Stories

- It is easier to relate to real-life examples rather than abstractions
- Stakeholders may not be very good at telling us system requirements, but they are generally more effective at describing how they actually do things or at envisioning how they might do things in a new way of working
- Stories are descriptions of how the system can be used for a particular task
  - What users do, what information they use, what output they need…

Stories

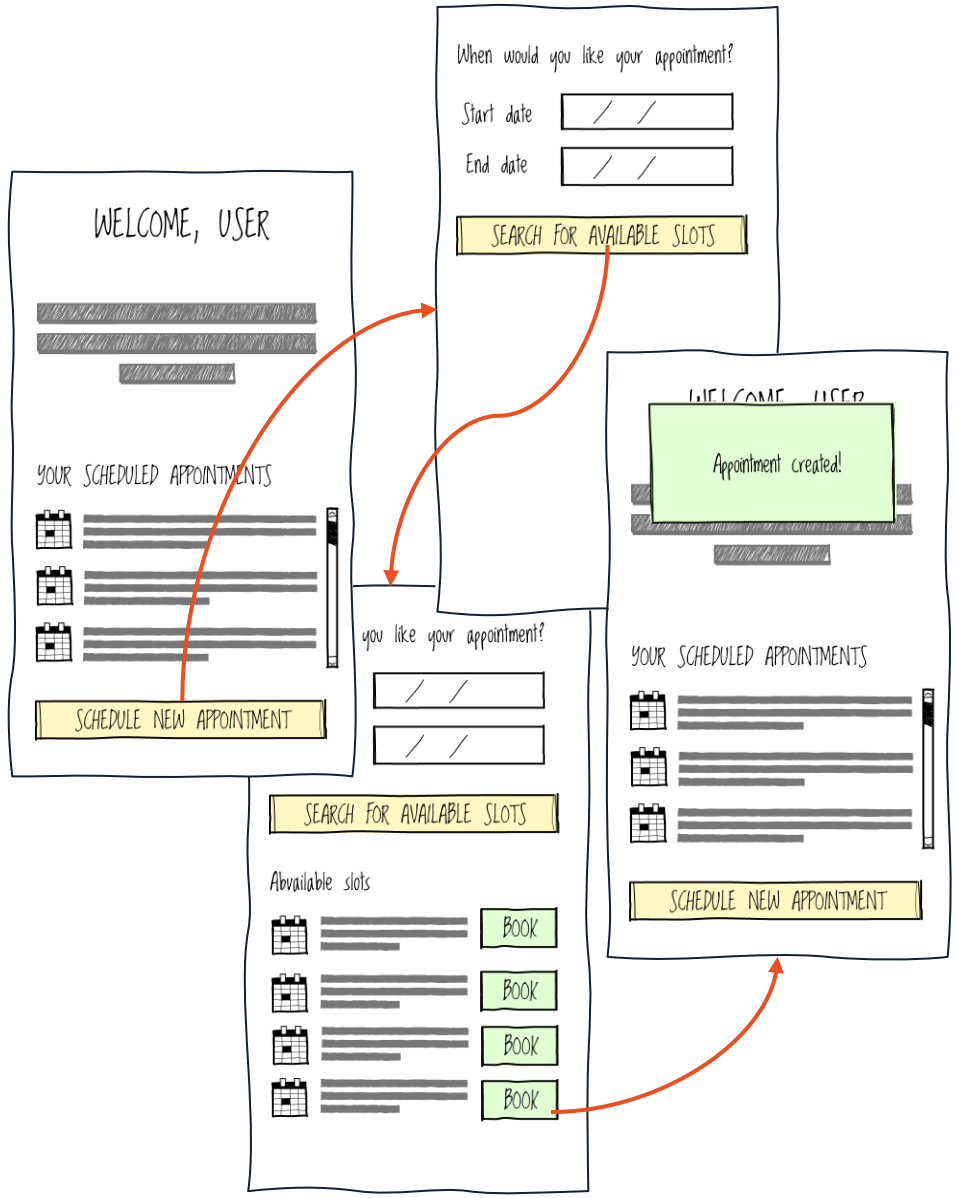
- Stories are written as narrative text and present an high-level description of the system use
- They are effective in presenting relatable “big picture” goals

Story: Scheduling an appointment by using the app

Pam needs to schedule a dental care appointment because her wisdom tooth is starting to hurt. She logs into the system and searches for available slots in the same day or tomorrow. The system shows a list of available slots. Pam selects one of the proposed slots and confirms the booking. Pams receives a confirmation message via email, with the details of the booked appointment.

Low-fidelity mockups

- Low-fidelity mockups (or **wireframes**) are **simplified sketches** of the System user interfaces
- Quick and cost-effective way to visualize the structure and basic flows of the system
- Focus is functionality and flow, not aesthetics!



Low-fidelity mockups: benefits

- **Facilitate Understanding**: Help stakeholders grasp the system's basic functionality. Serve as a starting point for discussions, ensuring clear communication of initial ideas.
- **Help discovering new requirements**: it is easier to reason about concrete interfaces rather than abstract statements.
- **Engagement and Iterative Feedback**: Promotes stakeholder engagement and allows for rapid iteration and feedback without significant investment in design.
- **Foundation for Development:** Provides a solid foundation for moving towards higher-fidelity designs and eventual development.

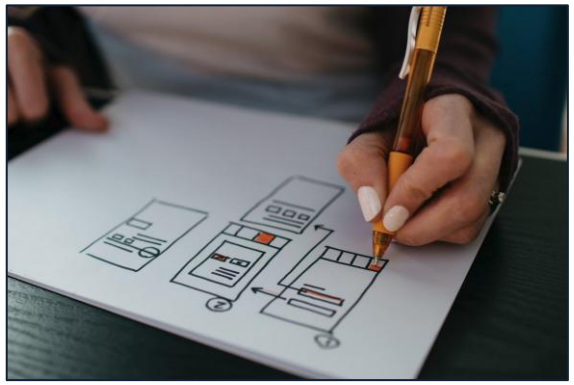
Low-fidelity mockups: tools

Paper and Pencil:

- Quick and accessible for initial sketches. May get messy for complex systems.

Digital Tools:

- Commercial software like Balsamiq or Figma allows for creating and sharing wireframes electronically
- Open-source alternatives such as mydraft.cc, draw.io or Pencil Project also exist



Readings and references

- *The inmates are running the asylum*, Alan Cooper