Pamantasan ng Lungsod ng Maynila
(University of the City of Manila)

College of Information Systems and Technology Management

REQUIREMENTS ENGINEERING

Report Transcript

Block 1 - Group 2

Members:

Abenoja, Stephen Lloyd C.	2022-34002
Alcantara, Paolo Mackenzie C.	2022-34005
Alibin, Harley Usher C.	2021-11477
Carpina, Christine Joy I.	2022-34030
Levita, Jehan Chaiz P.	2022-34072
Ong, Alexandra A.	2021-11491
Tolentino. Michelle N. *[Leader]*	2022-34103

Submitted to:

Prof. Jonathan C. Morano

September 16, 2024

TABLE OF CONTENTS

4.1 Functional & Non-Functional Requirements 3

4.1.1 Functional Requirements 3

4.1.2 Non-functional Requirements 5

4.2 Requirements Engineering Processes 7

4.3 Requirements Elicitation 8

4.3.1 Requirements Elicitation Techniques 13

4.3.1.1 Interviewing 13

4.3.1.2 Ethnography 15

4.3.1.3 Stories & Scenarios 16

4.4 Requirements Specification 18

4.4.1 Writing Notations 19

4.4.2 Use-Cases 20

4.4.3 Software Requirements Document 20

4.5 Requirements Validation 22

4.6 Requirements Change 23

4.6.1 Requirements management planning 24

4.6.2 Requirements Change Management 24

REQUIREMENTS ENGINEERING

Report Transcript

Contents

4.1 Functional and non-functional requirements
4.2 Requirements engineering processes
4.3 Requirements elicitation
4.4 Requirements specification
4.5 Requirements validation
4.6 Requirements change

4.1 Functional & Non-Functional Requirements

4.1.1 Functional Requirements

Functional requirements describe what a system should do. They vary based on the software, users, and organizational approach. User requirements are written in simple language for easy understanding, while system requirements provide detailed descriptions for developers, including functions, inputs, outputs, and exceptions.

A functional requirement should be:

- **Specific and unambiguous**: The requirement should be clear and leave no room for interpretation.
- **Measurable and observable**: It should have criteria that can be tracked or seen.
- **Testable**: There must be a way to verify if the requirement has been met through testing.

Figure 4.1 MOSCOW acronym

MOSCOW is a prioritization technique used in project management and requirements engineering to categorize and prioritize requirements. The acronym "MOSCOW" stands for:

- **(MO) Must Have**: Requirements categorized as "Must have" are considered essential for the project's success. These are non-negotiable features or functionalities that must be delivered for the product to be considered complete.

  Example:

  User Authentication: Users must be able to create accounts, log in, and manage their profiles securely.

- **(S) Should Have**: These are important requirements that are not as critical as "Must have" but significantly contribute to the project's overall success. "Should have" requirements are desirable and should be included if possible, but their omission would not jeopardize the project's viability.

  Example:

  Shopping Cart: Users should be able to add products to a shopping cart and proceed to checkout.

- **(Co) Could Have**: "Could have" requirements are features or functionalities that would be nice to have but are not critical for the project's success. Their inclusion may enhance the product, but they can be deferred to a later phase or release if necessary.

  Example:

  Wishlist: Users could have the option to create a wishlist for products they want to save for later.

- **(W) Won't Have (But Would Like)**): This category includes features or functionalities that stakeholders have discussed or considered but have explicitly decided not to include in the current project scope. These items may be considered for future releases or projects.

  Example:

  Augmented Reality Try-On: While it would be exciting, the team has decided not to include a feature allowing users to virtually try on clothes using augmented reality in this release due to resource constraints.

4.1.2 Non-functional Requirements

- are requirements that are not directly concerned with the specific services delivered by the system to its users. These non-functional requirements usually specify or constrain characteristics of the system as a whole.
- They may relate to emergent system properties such as reliability, response time, and memory use. Alternatively, they may define constraints on the system implementation, such as the capabilities of I/O devices.

Non-functional requirements are often more critical than individual functional requirements. System users can usually find ways to work around a system function that doesn't really meet their needs. However, failing to meet a non-functional requirement can mean that the whole system is unusable.

Types of non-functional requirements

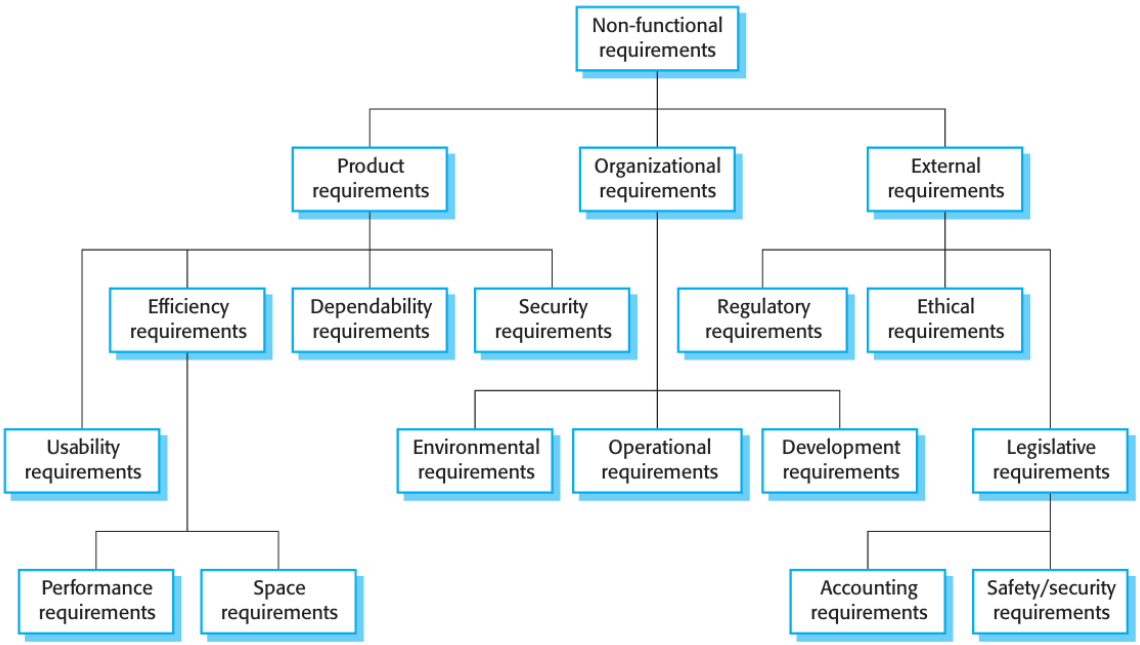

Figure 4.2 *Types of non-functional requirements*

Non-functional requirements may come from required characteristics of the software (*product requirements*), the organization developing the software (*organizational requirements*), or *external sources.*

1. **Product requirements** - specify or constrain the runtime behavior of the software.

   Examples:

   - Performance requirements: how fast the system must execute and how much memory it requires.
   - Reliability requirements: set out the acceptable failure rate.

- Security requirements
- Usability requirements

2. **Organizational requirements** - are broad system requirements derived from policies and procedures in the customer's and developer's organizations.

   Examples:

   - Operational process requirements: define how the system will be used.
   - Development process requirements: specify the programming language, development environment, or process standards to be used.
   - Environmental requirements: specify the operating environment of the system.

3. **External requirements** - all requirements that are derived from factors external to the system and its development process.

   Examples:

   - Regulatory requirements: set out what must be done for the system to be approved for use by a regulator.
   - Legislative requirements: ensure that the system operates within the law
   - Ethical requirements: ensure that the system will be acceptable to its users and the general public.

Table 4.1 *Metrics for specifying non-functional requirements*

Property	Measure
Speed	Processed transactions/second User/event response time Screen refresh time
Size	Megabytes/Number of ROM chips
Ease of use	Training time Number of help frames
Reliability	Mean time to failure Probability of unavailability Rate of failure occurrence Availability
Robustness	Time to restart after failure Percentage of events causing failure Probability of data corruption on failure
Portability	Percentage of target dependent statements

	Number of target systems

Whenever possible, you should write non-functional requirements quantitatively so that they can be objectively tested. **Table 4.1** shows metrics that you can use to specify non-functional system properties. You can measure these characteristics when the system is being tested to check whether or not the system has met its non functional requirements.

4.2 Requirements Engineering Processes

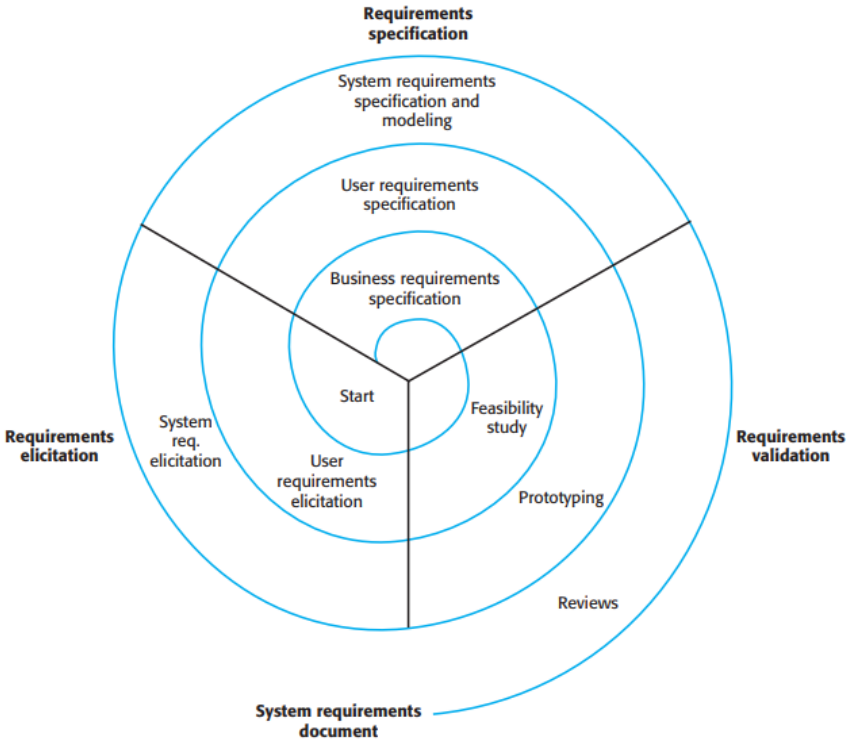

Figure 4.3 *A spiral view of the requirements engineering process*

In Chapter 2, requirements engineering was explained to involve three main activities: 1.) gathering requirements from stakeholders (known as **elicitation** and **analysis**), 2.) transforming these requirements into a standard format (**specification**), and 3.) ensuring the requirements match what the customer wants (**validation**). Although these activities are often presented as sequential, in reality, they are part of an ongoing, **iterative process**.

As shown in Figure 4.3, the iterative nature of requirements engineering is represented as a **spiral**. The **output of this process is a system requirements document**, with the amount of

time spent on each activity varying depending on the project's stage, the type of system being developed, and the available budget.

In the early stages, the focus is on understanding high-level business needs and user requirements. As the process advances, more attention is given to detailed system and non-functional requirements. The number of iterations around the spiral can vary so that the spiral can be exited after some or all of the user requirements have been elicited. This spiral model allows for flexibility, adapting to development approaches where requirements can be defined at varying levels of detail. For instance, Agile development may replace prototyping to allow simultaneous development of both the requirements and the system.

Changes to requirements are inevitable in most systems due to evolving understanding from stakeholders, shifts in the purchasing organization, or modifications to the system's hardware, software, or environment. These changes must be carefully managed to assess their impact on other requirements and overall project costs. The process of handling these changes is further discussed in Section 4.6.

4.3 Requirements Elicitation

- **Elicit** - *(verb)* to call forth or draw out (something, such as information or a response)
- **Requirements Elicitation** - to understand the work that stakeholders do and how they might use a new system to help support that work.

  Challenges to eliciting requirements:

  1. Unclear or Unrealistic Expectations
  Stakeholders often have only a vague idea of what they want from a system and may struggle to clearly articulate their needs. Because they lack technical expertise, they might make unrealistic demands, not understanding what is technically feasible or within budget. This can lead to challenges for requirements engineers in translating these general expectations into specific, achievable system requirements.

  Sample: Instant Feature Implementation
  - **Requirement:** *"We need a system that can process data in real-time."*
  - **Stakeholder version:** *"It should process everything instantly—like the moment I press a button, I want the report done, no matter how much data we have."*
  - **Challenge:** The expectation for instantaneous processing is unrealistic for large datasets or complex operations, which require processing time depending on system capacity.

  2. Domain-Specific Jargon

Each stakeholder typically expresses their requirements using their own work-related terminology, assuming implicit understanding of their job's processes. Requirements engineers, unfamiliar with the specific domain, may misinterpret these needs. This communication gap makes it harder to accurately capture the true requirements of the system unless domain knowledge is gained.

Sample: Healthcare Industry

- **Stakeholder version:** *"The system should ensure accurate triage and escalate to a higher acuity level when the patient's SOFA score goes up."*
- **What they mean:** They want the system to detect changes in a patient's condition, using a specific scoring method (Sequential Organ Failure Assessment or SOFA), and automatically raise the priority of care based on certain medical indicators.
- **Challenge:** The requirements engineer may not be familiar with medical scoring systems like SOFA and might need additional clarification to understand how it should affect system behavior.

3. Diverse and Conflicting Requirements

Different stakeholders often have diverse, sometimes conflicting requirements based on their roles and perspectives. They may express these needs in different ways, making it challenging for requirements engineers to consolidate and identify commonalities. The engineer must carefully navigate these differences to balance conflicting priorities while ensuring that all necessary requirements are addressed.

Sample: Banking System

- **Customer Service Team's Requirement:** *"We need a system that's easy to navigate and allows us to process customer requests quickly without going through too many steps."*
- **Compliance Team's Requirement:** *"We need a system that enforces strict approval workflows and logs every transaction detail for regulatory auditing."*
- **Conflict:** Customer service wants a fast and streamlined process, while the compliance team requires additional layers of approval and detailed documentation, which slows things down. The challenge is finding a balance between speed and regulatory compliance.

4. Influence of Organizational Politics

In some cases, managers or influential stakeholders may push for certain system requirements that are driven by organizational politics rather than genuine technical or business needs. This can skew the focus of the project toward features that enhance the manager's power within the organization rather than addressing the broader needs of the system or the users.

Sample: Manufacturing System

- **CEO’s Requirement:** *"The system should provide real-time financial data to monitor profitability and production costs."*
- **Factory Manager’s Requirement:** *"We need detailed tracking of production metrics, machine performance, and worker shifts to ensure smooth factory operations."*
- **Political Influence:** The CEO, with final decision-making authority, might prioritize financial oversight, whereas the factory manager may need tools for managing the production floor. As a result, the system might overemphasize financial data, potentially undermining the operational efficiency of the factory.

5. Changing Economic and Business Environment

The dynamic nature of the business and economic environment means that requirements may change throughout the project. As the analysis process unfolds, new stakeholders may emerge, and priorities may shift, leading to new or revised requirements. Requirements engineers must stay flexible and adapt to these evolving needs while considering the impact on the overall system design and budget.

Sample: E-commerce Platform

- **Initial Requirement:** *"We need a system that can handle local online orders and payments for our domestic market."*
- **New Requirement Due to Economic Change:** *"Due to a shift in market strategy, we now need the platform to support international transactions, multiple currencies, and shipping integrations as we expand into global markets."*
- **Impact:** Initially focused on local business, the platform must now support complex international logistics, payment gateways, and compliance with global regulations to keep pace with economic expansion.

Elicitation and Analysis process

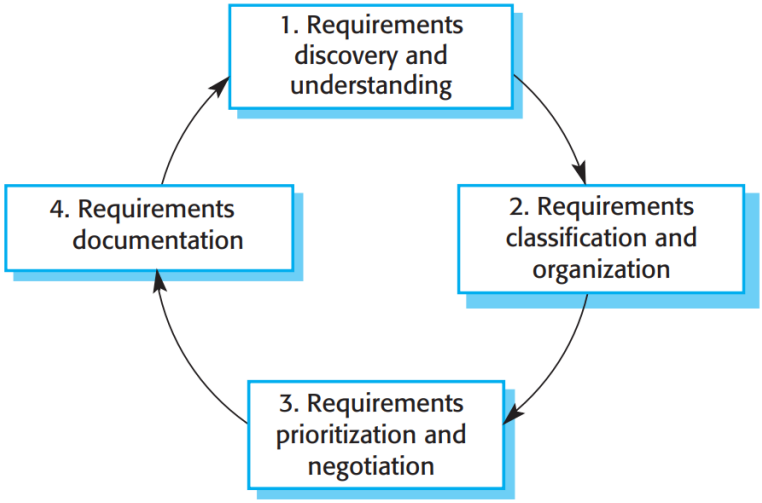

Figure 4.4 *The requirements elicitation and analysis process*

The requirements elicitation and analysis process **(Fig 4.4)** is a structured approach used to gather and refine system requirements. Different organizations may tailor this general model depending on factors such as the team's expertise, the type of system being developed, and organizational standards. The process can be broken down into four main activities:

1. **Requirements Discovery and Understanding**
   This is the initial phase where stakeholders are engaged to uncover what they need from the system. The process involves direct interaction with stakeholders to discover functional and non-functional requirements. Additionally, domain requirements are gathered from both stakeholders and existing documentation, which helps identify what the system should achieve within its specific environment.

   Goal: Understand what stakeholders need and clarify the high-level system expectations.

2. **Requirements Classification and Organization**
   After gathering a broad set of unstructured requirements, the next step is to categorize and organize them. This activity involves grouping related requirements and creating structured clusters, helping to make sense of the diverse inputs received. This organization ensures that the requirements are easier to manage and reference throughout the development process.

   Goal: Transform a raw collection of requirements into a clear and structured set of organized needs.

3. **Requirements Prioritization and Negotiation**

Since multiple stakeholders are typically involved, conflicting requirements are inevitable. The prioritization and negotiation process is key to resolving these conflicts. Stakeholders work together to rank the importance of different requirements and reach agreements through compromise. This step helps to focus development on the most critical features and ensures that conflicting demands do not derail progress.

Goal: Resolve conflicting requirements and prioritize based on stakeholder consensus.

4. **Requirements Documentation**
   Once the requirements have been identified, organized, and prioritized, they are documented. This documentation could take the form of formal software requirements documents or informal notes maintained on shared spaces like whiteboards or wikis. These documents are then used as a reference for future iterations of the project, ensuring that everyone is aligned on what has been agreed upon so far.

   Goal: Provide clear, accessible documentation of the agreed-upon requirements for future use.

4.3.1 Requirements Elicitation Techniques

Requirements elicitation involves meeting with various stakeholders to gather information about the proposed system. This process may also include examining existing systems, their usage, and relevant documentation. It is essential to spend time understanding how people work, what they produce, how they interact with other systems, and how their workflows may need to adapt to a new system. There are two primary approaches to requirements elicitation: 1.) **interviewing**, where you talk to people about their roles and tasks, and 2.) **observation** or **ethnography**, where you observe people at work to understand the tools and processes they use. A combination of both approaches is recommended to collect comprehensive information, which forms the foundation for deriving system requirements and serves as the basis for further discussions.

4.3.1.1 Interviewing

Interview- is a structured or semi-structured conversation between a requirements engineer and one or more stakeholders. The goal is to gather detailed information about the stakeholders' needs, expectations, and constraints regarding the system being developed.

In requirements engineering, interviews are a key method for gathering information from system stakeholders. These interviews involve asking stakeholders questions about both the system they currently use and the system being developed. The answers help derive system requirements. Interviews can be classified into two types:

1. **Closed interviews:** In this format, stakeholders respond to a predefined set of questions.
2. **Open interviews:** These have no set agenda, allowing the requirements engineering team to explore various topics to better understand stakeholder needs.

In practice, interviews are usually a blend of both closed and open formats. While certain questions must be answered, they often lead to further unstructured discussions. Fully open-ended interviews tend to be less effective, as some structure is needed to keep the conversation focused on the system being developed.

To be an effective interviewer in the requirements elicitation process, you need to follow two key guidelines:

1. **Be open-minded-** It's important to approach each interview without preconceived notions or rigid ideas about what the system requirements should be. Listening to stakeholders with an open mind allows you to adapt to their input. If stakeholders present surprising or unexpected requirements, you must be willing to reconsider your initial assumptions and potentially shift your understanding of the system. For example, a stakeholder might propose a feature you hadn’t anticipated because

it's outside the usual technical scope, like integrating social media feeds into a project management system. Instead of dismissing it as unfeasible, you should explore the rationale behind this idea and whether it aligns with the business goals or enhances the user experience. An open-minded approach helps build trust with stakeholders and encourages them to share candidly, knowing that their insights will be considered.

2. **Use prompts and provide context-** Simply asking stakeholders, *"Tell me what you want"* is rarely effective. Stakeholders often struggle to articulate specific needs when faced with a blank slate. To facilitate productive discussions, you should introduce tools like springboard questions, requirements proposals, or even prototype systems to guide the conversation.

Springboard questions

These are open-ended questions designed to spark discussion by focusing on specific areas of interest or pain points. Rather than asking a vague, general question like *"What features would you like in the system?"*, you might ask, *"What tasks take the most time in your daily workflow?"* or "*Which processes in your current system cause the most frustration?"* By focusing on areas that stakeholders are familiar with, you allow them to frame their responses based on real experiences. For example, a springboard question might lead a marketing manager to talk about the difficulty in tracking customer engagement across multiple platforms, which could evolve into a requirement for centralized tracking tools.

Requirements proposals

In many cases, it's more effective to present stakeholders with an initial set of suggested system requirements based on prior research or industry best practices. For instance, if you're eliciting requirements for a sales management platform, you could propose features like automated lead generation, sales tracking dashboards, or client communication logs. This allows stakeholders to respond with clarifications or modifications: they may agree with certain proposals, reject others, or suggest additional features. A stakeholder might say, *"I like the idea of automated lead generation, but we also need customizable lead scoring to prioritize clients."* This approach gives them a framework to react to, making it easier for them to express more specific needs than if they had been asked to start from scratch.

Prototype systems

A prototype is a visual or interactive model of the system, which serves as a practical tool for gathering feedback. Rather than simply discussing abstract concepts, stakeholders can see and interact with a working model of the system to provide concrete input. For example, if you're developing a customer support portal, showing a prototype where users can submit, track, and escalate their

issues helps stakeholders visualize the flow of interactions. They might then suggest modifications like adding a live chat feature, making it more mobile-friendly, or enhancing the ticket categorization process. The act of using a prototype allows stakeholders to identify usability issues or gaps in functionality that might not have surfaced in a verbal discussion.

Prototypes are particularly useful when dealing with non-technical stakeholders who may struggle to conceptualize what the system could look like from a requirements list alone. By providing a hands-on experience, they can better understand how the system will meet their needs, and can express detailed requirements they might otherwise have missed.

4.3.1.2 Ethnography

- Ethnography is an observational technique that can be used to understand operational processes and help derive requirements for software to support these processes.
- It helps discover implicit system requirements that reflect the actual ways that people work, rather than the formal processes defined by the organization.
- People often find it very difficult to articulate details of their work because it is second nature to them, which is why an analyst immerses themselves in the working environment where the system will be used.

Ethnography is effective for discovering two types of requirements:

1. Requirements derived from the way in which people actually work, rather than the way in which business process definitions say they ought to work.
2. Requirements derived from cooperation and awareness of other people's activities.

Ethnography can be combined with the development of a system prototype to lessen the cycles of prototype refinement required.

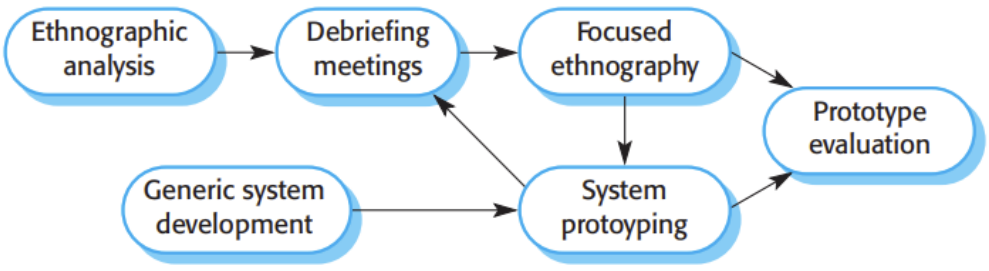

Figure 4.5 *Ethnography and prototyping for requirements analysis*

Ethnographic studies can reveal critical process details that are often missed by other requirements elicitation techniques, but this understanding does not always help with innovation for new product development. Because of its focus on the end user, this approach is not effective for discovering broader organizational or domain requirements or for suggesting innovations.

4.3.1.3 Stories & Scenarios

Stories and scenarios are ways to describe how particular situations are handled. They are not good at telling you the system requirements. Stories and scenarios are essentially the same thing, you can use these when interviewing groups of stakeholders to discuss the system and to develop more specific system requirements. The difference is in the ways that descriptions are structured and in the level of detail presented.

Stories are written as narrative text and present a high-level description of system use. They are effective in setting out the big picture. The advantage is that everyone can easily relate to them, and it is useful to get information from a wider community than you could realistically interview.

Scenarios are usually structured with specific information collected such as inputs and outputs; they are descriptions of example user interaction sessions. A scenario starts with an outline of the interaction, and at its most general, may include the following:

1. A description of what the system and users expect when the scenario starts.
2. A description of the normal flow of events in the scenario.
3. A description of what can go wrong and how resulting problems can be handled.
4. Information about other activities that might be going on at the same time.
5. A description of the system state when the scenario ends.

Jack is a primary school teacher in Ullapool (a village in northern Scotland). He has decided that a class project should be focused on the fishing industry in the area, looking at the history, development, and economic impact of fishing. As part of this project, pupils are asked to gather and share reminiscences from relatives, use newspaper archives, and collect old photographs related to fishing and fishing communities in the area. Pupils use an iLearn wiki to gather together fishing stories and SCRAN (a history resources site) to access newspaper archives and photographs. However, Jack also needs a photo-sharing site because he wants pupils to take and comment on each other's photos and to upload scans of old photographs that they may have in their families.

Jack sends an email to a primary school teachers' group, which he is a member of, to see if anyone can recommend an appropriate system. Two teachers reply, and both suggest that he use KidsTakePics, a photo-sharing site that allows teachers to check and moderate content. As KidsTakePics is not integrated with the iLearn authentication service, he sets up a teacher and a class account. He uses the iLearn setup service to add KidsTakePics to the services seen by the pupils in his class so that when they log in, they can immediately use the system to upload photos from their mobile devices and class computers.

Figure 4.6 *Example structure of a Stories-type Requirements Description*

Initial assumption: A user or a group of users have one or more digital photographs to be uploaded to the picture-sharing site. These photos are saved on either a tablet or a laptop computer. They have successfully logged on to KidsTakePics.

Normal: The user chooses to upload photos and is prompted to select the photos to be uploaded on the computer and to select the project name under which the photos will be stored. Users should also be given the option of inputting keywords that should be associated with each uploaded photo. Uploaded photos are named by creating a conjunction of the user name with the filename of the photo on the local computer.

On completion of the upload, the system automatically sends an email to the project moderator, asking them to check new content, and generates an on-screen message to the user that this checking has been done.

What can go wrong: No moderator is associated with the selected project. An email is automatically generated to the school administrator asking them to nominate a project moderator. Users should be informed of a possible delay in making their photos visible.

Photos with the same name have already been uploaded by the same user. The user should be asked if he or she wishes to re-upload the photos with the same name, rename the photos, or cancel the upload. If users choose to re-upload the photos, the originals are overwritten. If they choose to rename the photos, a new name is automatically generated by adding a number to the existing filename.

Other activities: The moderator may be logged on to the system and may approve photos as they are uploaded.

System state on completion: User is logged on. The selected photos have been uploaded and assigned a status "awaiting moderation." Photos are visible to the moderator and to the user who uploaded them.

Figure 4.7 *Example structure of a Scenarios-type Requirements Description*

4.4 Requirements Specification

The process of writing down/document User Requirements and System Requirements**.** Ideally clear, unambiguous, easy to read, and consistent

User Requirements

- Often written in *Natural Language* notation.
  User requirements should contain basic information, and details that are easy to understand without the need for technical experience and knowledge. Thus, it is commonly written using basic, natural language, listed line by line.

- Describes functional & non-functional requirements of the system.
  Without delving into the internal functions of the system/project, user requirements enumerate the baseline/minimum requirements that must be achieved by the developers. However, its contents and complexity are limited to the external behaviour of the system: the results of its processes, and its outputs.

- Should not contain software jargon.
  The user requirements are written with the client, and non-developer individuals in mind. To avoid confusion, the developers must refrain from including technical jargon and terminologies in this section.

System Requirements

- Expands on User Requirements.
  System requirements delve further into the project's details, and are typically more technical than the user requirements. Here, it is explained **how** the user requirements will be provided, and expands on what needs to be achieved.

- Complete, detailed specification of the system.
  In some cases, the written system requirements may be used as a 'checklist' of sorts, as a part of the project's contract. Therefore, it should be a complete, and detailed specification of the whole system.

- Used as a starting point by the software developers.
  Because the system requirements detail how the user requirements are to be achieved, it is often used as a starting point of development. It gives the software engineers a direction to follow, and serves as a basis to build the system off of.

4.4.1 Writing Notations

1. **Natural Language**

   Requirements written in Natural Language notation are jotted down in numbered sentences, written with plain, natural language. Each sentence/line contains one requirement, and may be supplemented by images, or simple charts.

   Natural language is intuitive, fast to write, and can be as expressive as the writer wishes, which is why it's a very common medium of writing requirements. It does however, bear the pitfall of potentially being vague, ambiguous, and easy to misinterpret. The following are a number of guidelines to ensure that a message written in natural language may be delivered concisely and clearly.

   i. Use a standard format.

      Devise a standard, consistent format for writing in natural language, and adhere to it. Using a standard format will lessen ambiguity, and make the requirements easier to check.

   ii. Distinguish between mandatory and "desirable" features.

      To ensure that the development of the system is steered in the correct direction, use language that distinguishes necessary, uncompromisable requirements from soft, 'desirable' requirements. The use of words like 'Should', and 'Shall' are an option, though other forms of distinguishment, such as symbols and font alterations may also be used.

   iii. Use text highlighting to pick out key parts of the requirement.

      The use of text formatting tools, such as bold fonts, font colours, and font sizes allow key points to be emphasized in writing. This will draw the readers' attention to important information, and away otherwise.

   iv. Keep it simple.

      Do not assume that the reader understands technical, software engineering language. Avoid using esoteric terms, complex concepts, and complicated jargon.

   ` v. Attach a rationale with each requirement.

      Explain why each requirement is present, and what the goal is with its inclusion. This will make it easier for readers– especially the engineers to understand what is trying to be achieved, and to adjust accordingly if changes are necessary.

2. **Structured Natural Language**

   Similar to Natural Language notation, Structured Natural Language majorly employs the use of simple sentences written in natural language. However, rather than being completely free-form, Structured Natural Language notation uses standardized templates and formats in order to impose uniformity and consistency unto the requirements, while maintaining the expressiveness and understandability of Natural Language.

   When using a standard format for system requirements using Structured Natural Language, the following information must be included:

   1. The function, or entity being specified

2. The inputs of the function or entity
3. The outputs of the function or entity
4. The prerequisites of the function or entity
5. The action that the function or entity takes
6. The precondition (what must be true before the action is taken)
7. The postcondition (what becomes true after the action is taken)
8. The side-effects (if any) of the action.

In cases where information remains vague, or ambiguous under the use of Structured Natural Language notation, the use of tables, or other similar visual templates may be employed.

3. **Graphical Notation**
   When describing figures and concepts far too specific to explain with words, the use of a Graphical Notation may be employed. Here, the focus is shifted to the use of charts, models, diagrams, and other forms of visual representations of data and information. The graphical models are supplemented by text labels and annotations for clarity, but the concept is mainly delivered through intuitive models.

4. **Mathematical Specifications**
   Another means of conveying information and data graphically is through the use of Mathematical models and diagrams: graphics such as venn diagrams, finite state machines, state transition diagrams, etc. or even through the use of mathematical equations. This is a notation with a relatively obscure, and specific use-case, and is less likely to be encountered, especially because most readers would fail to understand the esoteric concepts, yet it is still a valuable tool to keep in mind.

4.4.2 Use-Cases

Use cases describe interactions between users and the system, usually using a graphical model and structured text. They aid in enumerating and identifying the actors involved in the transactions that will occur in the system, and how they will interact with each other, and the system itself. Defining use-cases allow engineers to optimize interactions, and lay out fail safes in preparation for edge-cases, and potential failures.

4.4.3 Software Requirements Document

The Software Requirements Document– sometimes called the Software Requirements Specification (SRS), is where the requirements of a system are compiled for documentation. It is the official statement of what the engineers and developers must implement, and is accessed by a diverse set of actors in the project. In projects where development is split into teams, an SRS is particularly essential, in order to keep track of the requirements that must be fulfilled.

The structure of an SRS may vary between projects, and settings, but what follows is a sample structure of what an SRS may look like.

Table 4.2 *Sample SRS Structure*

Chapter	Description/Content
Preface	Defines the expected readership of the document and enumerates its version history, including changes committed in between them (if any).
Introduction	Describes the goal for the system, and briefly describes the system's functions, including how it'd work with other systems. Also describes how the system fits into the client, and business' goals and objectives.
Glossary	Defines the terms used in the document– especially useful for clarifying jargon and technical terms.
User Requirements	Describes the services provided for the user, and the nonfunctional requirements. Often uses natural language,and other simple diagrams and notations.
System Architecture	Presents a high-level overview of the anticipated system architecture, and shows the distribution of functions across system modules.
System Requirements	Describes the functional, and nonfunctional requirements in more detail.
System Models	Presents a graphical model showing the relationships and interactions between the system components, the system, and its environment.
System Evolution	Describes the anticipated changes that may occur throughout the course of the system, possibly due to hardware upgrades, or changes in the user base. Particularly useful for system designers, to avoid making design decisions that would constrain growth
Appendices	Provides detailed, specific information related to the systems' development.
Index	A structured index of the document.

4.5 Requirements Validation

Requirements validation is the process of checking that requirements define the system that the customer really wants. It involves identifying and addressing issues with requirements, which can lead to significant rework costs and require system changes, often requiring retesting and alterations to design and implementation.

1. **Validity Checks:** The requirements are reviewed to ensure they accurately reflect the actual needs of system users, as they may have evolved due to evolving circumstances.
2. **Consistency Checks:** The system function should be accurately described and not conflict with any contradictory constraints or different descriptions in the document.
3. **Completeness Checks:** The requirements document should clearly outline all functions and constraints intended by the system user.
4. **Realism Checks:** The requirements should be checked to ensure that they can be implemented within the proposed budget for the system.
5. **Verifiability:** The system requirements should always be written so that they are verifiable.

Requirements Validation Techniques:

1. **Requirements Reviews:** A team performs a systematic analysis of the needs of reviewers who look for errors and inconsistencies.
2. **Prototyping:** The process involves creating an executable system model, testing it with end-users and customers, and incorporating stakeholder feedback to ensure it meets their needs and expectations.
3. **Test-case Generation:** Requirements should be testable, and if tests are part of validation, they reveal problems. Test-driven development involves developing user requirements tests before code is written.

4.6 Requirements Change

- In the development of large software systems, requirements are constantly evolving. One of the main reasons is that these systems often address **wicked problems**—complex issues that cannot be fully defined at the outset.
- Initial requirements are incomplete, and as stakeholders gain a better understanding of the problem throughout the development process, system requirements must adapt accordingly.

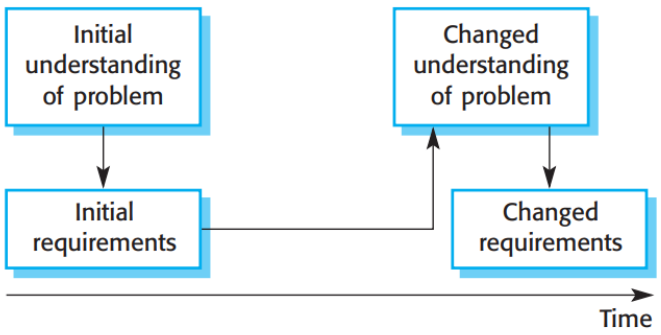

Figure 4.8 *Requirements Evolution*

Once a software system has been installed and is regularly used, new requirements inevitably emerge as shown in Figure 4.8. While some changes result from errors and omissions in the original requirements, most adjustments arise due to shifts in the business environment of the system:

1. **Dynamic Business Environment**: New hardware may be introduced, or existing hardware may be updated. Systems may need to interface with other platforms, business priorities can shift, and new legislation or regulations may demand system compliance.
    - For example, a bank may upgrade its ATMs, requiring software updates to integrate with its backend systems, while also adjusting for new security regulations.

2. **Diverse Stakeholder Needs**: The individuals funding a system often differ from the system's end users. Customers may impose requirements based on organizational and budgetary constraints, which may conflict with the needs of the users. After deployment, new features may need to be added to ensure the system meets its intended goals.
    - For instance, a school district may fund an online learning platform, but teachers find it difficult to use, leading to post-launch feature updates for better usability.

3. **Stakeholder Conflicts**: Large systems typically involve a diverse group of stakeholders, each with different requirements and priorities, some of which may conflict. The final

system requirements are often a compromise, but as the system is used, it may become apparent that this balance needs to shift, leading to a re-prioritization of requirements.

- In particular, a hospital's management may want an affordable system, while doctors prioritize ease of use. After launch, the system may be updated to better meet doctors' needs without exceeding budget constraints.

4.6.1 Requirements management planning

Requirements management planning focuses on how to handle changing requirements during a project. During this planning, there are several issues:

1. **Requirements Identification**: Each requirement needs a unique ID so it can be easily referenced and tracked throughout the project.
2. **Change Management Process**: A defined process is needed to assess the impact and cost of any changes to the requirements.
3. **Traceability Policies**: These policies outline how to track the relationships between requirements and the system design, and how to maintain those records.
4. **Tool Support**: Managing requirements involves handling a lot of information, so tools like requirements management systems or even simple spreadsheets and databases can be used to help organize and track everything.

4.6.2 Requirements Change Management

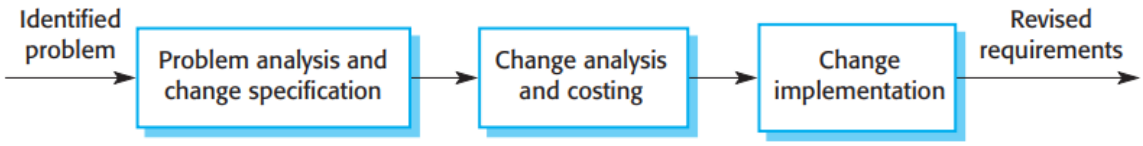

Figure 4.9 *Requirements Change Management*

Requirements change management shown in Figure 4.9 is essential for ensuring that any proposed changes to the system's requirements are justified and controlled. This formal process has three principal stages:

1. **Problem Analysis and Change Specification**: A requirements problem or change proposal is identified and analyzed to ensure its validity. Feedback is provided to the requestor, who may revise the proposal or withdraw it.
2. **Change Analysis and Costing**: The impact of the proposed change is assessed using traceability information. The cost of making the change is estimated, including necessary updates to the requirements, system design, and implementation. A decision is then made on whether to proceed with the change.
3. **Change Implementation**: If approved, the requirements document and possibly the system design are modified. To make this process efficient, the document

should be structured in a modular way, minimizing external references so that sections can be updated without requiring extensive rewriting or affecting other parts of the document.

External References

Haas, L. (2021, November 2). *Secure Authentication – everything you need to know*. SecureCoding. https://www.securecoding.com/blog/secure-authentication/

TestCaseLab, O. H. F. (2023, August 2). Testing Shopping Cart & Checkout Process - Olha Holota from TestCaseLab - Medium. *Medium*. https://medium.com/@case_lab/testing-shopping-cart-checkout-process-444d08a09a91

What is Wishlist? - Why Wishlist is Important Feature in Ecommerce. (n.d.). Oberlo. https://www.oberlo.com/ecommerce-wiki/wishlists

How augmented reality (AR) impacts the fashion industry. (2024, September 5). *HQSoftware*. https://hqsoftwarelab.com/blog/augmented-reality-in-fashion/