**No.1 User Requirements & System Requirements**

User requirements describe the services a system should provide and the constraints under which it operates. Example:

* User Requirement: The MHC-PMS shall generate monthly reports showing drug costs for each clinic.

System requirements give more detailed descriptions of the software’s functions and constraints. They define exactly what will be implemented and may be part of a contract between the buyer and developers. Example:

* System Requirement: A summary of drugs prescribed, their cost, and clinics shall be generated monthly and restricted to authorized users. The system will automatically generate the report after 17:30 on the last working day of the month.

**No.2 Identify and briefly describe four types of requirements that may be defined for a computer-based system.**

1. User Requirements: These describe what the user expects from the system in simple, non-technical terms. For example, "Users should be able to easily reset their passwords."
2. System Requirements: These detail the technical needs, such as hardware and software. For example, "The system must run on Windows 10 or higher and require at least 8GB of RAM."
3. Functional Requirements: These define what the system must do, such as specific features. For example, "The system must allow users to log in and view their account details."
4. Non-Functional Requirements: These describe how the system should perform, focusing on things like speed, security, and usability. For example, "The system should load the homepage within 2 seconds."

**No.3 Functional and Non-Functional Requirements**

Software system requirements are divided into functional and non-functional categories:

1. Functional Requirements: These describe what the system should do, including services provided, responses to inputs, and behavior in various situations. For example, in the MHC-PMS:
   * Users should be able to search appointment lists for all clinics.
   * The system must generate a daily list of patients for each clinic.
   * Each staff member should have a unique 8-digit employee number.
2. Non-Functional Requirements: These specify constraints on the system, like reliability, response time, and storage needs. They often impact the overall system architecture rather than individual features. For example:
   * Non-functional requirements might affect overall system design and lead to related functional requirements, such as security needs.

Non-functional requirements can be classified into:

* Product Requirements: Attributes of the final system.
* Organisational Requirements: Constraints related to the organization.
* External Requirements: External factors influencing the system.

**No.4 Requirements Engineering Process**

The processes used for RE vary widely depending on the application domain, the people involved and the organization developing the requirements. However, there are common generic activities to all processes.

1. Requirements elicitation
2. Requirements analysis
3. Requirements validation
4. Requirements management

Requirement Engineering is an iterative activity in which these processes are interleaved.

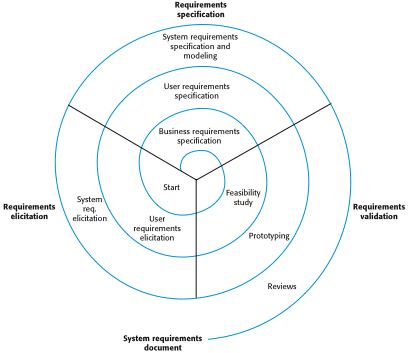


Figure. A spiral view of the requirements engineering process

**No 5 Requirements Elicitation and Analysis**

Requirements elicitation, also known as requirements discovery, involves technical staff working with stakeholders to understand the application domain, required services, and operational constraints. Stakeholders include end-users, managers, maintenance engineers, domain experts, and others.

The process includes:

1. Requirements Discovery
2. Requirements Classification and Organization
3. Requirements Prioritization and Negotiation
4. Requirements Specification

Problems in this process include:

* Stakeholders may not know exactly what they want.
* Requirements may be expressed in vague or varied terms.
* Conflicting requirements can arise among different stakeholders.
* Requirements can change during analysis due to new stakeholders or shifts in the business environment.

**No.6 Scenarios**

Scenarios are practical examples of how a system is used and should include:

1. Starting Situation: Describes the initial state.
2. Normal Flow of Events: Details the usual sequence of actions.
3. What Can Go Wrong: Lists potential issues and errors.
4. Other Concurrent Activities: Information on simultaneous actions by other users.
5. State When Finished: Describes the final system state.

Scenario for Collecting Medical History in MHC-PMS:

* Starting Situation: A nurse is logged into the system and collecting medical history.
* Normal Flow: The nurse searches for the patient by family name, uses the date of birth if needed, selects the option to add medical history, and follows prompts to enter consultation details.
* What Can Go Wrong: If the patient’s record is missing, the nurse must create a new record and input personal information.
* Other Activities: Other staff may view but not edit the record while the nurse enters data.
* State When Finished: The nurse remains logged in, the patient's medical history is saved in the database, and a log record is created with the session details and nurse information.

No.7 Requirements Validation

Requirements validation ensures that the system meets the customer's true needs.

Requirements Checking:

1. Validity Checks: Identify if additional or different functions are needed beyond initial thoughts.
2. Consistency Checks: Ensure no conflicting requirements in the document.
3. Completeness Checks: Verify the document covers all necessary functions and constraints.
4. Realism Checks: Confirm that requirements can be implemented with current technology.
5. Verifiability: Ensure requirements are clear and testable to avoid disputes.

Validation Techniques:

1. Requirements Reviews: A team systematically examines requirements for errors.
2. Prototyping: An executable model is shown to users and customers for feedback.
3. Test-Case Generation: Create tests to check if requirements are testable and identify issues.

No.8 Requirements Management Planning

Planning is a crucial first step in managing requirements. Key decisions include:

1. Requirements Identification: Each requirement must have a unique ID for easy reference and tracking.
2. Change Management Process: Activities to assess the impact and cost of changes.
3. Traceability Policies: Define and record the relationships between requirements and the system design.
4. Tool Support: Use tools like specialized systems, spreadsheets, or databases for managing requirements.

Automated support and software tools are essential:

1. Requirements Storage: Maintain requirements in a secure, accessible data store for all involved.
2. Change Management: Tools simplify managing changes.
3. Traceability Management: Tools help track related requirements.

No.9 Interviewing

Interviews with stakeholders are common in requirements engineering. Types include:

1. Closed Interviews: Use a pre-set list of questions.
2. Open Interviews: Explore various issues with stakeholders.

Effective Interviewing:

1. Be open-minded and avoid preconceived notions. Listen carefully to stakeholders.
2. Use prompts like springboard questions, requirements proposals, or prototypes to start discussions.

Interviews in Practice:

* Typically use a mix of closed and open-ended questions.
* Useful for understanding stakeholders' roles and system interactions but less effective for domain-specific knowledge:
  + Requirements engineers might not grasp specialized terminology.
  + Some domain knowledge is so familiar that stakeholders may find it hard to express or consider it unimportant.

No.10 Requirements Change Management

Change management is essential for handling any changes to a system's requirements after the requirements document is approved. The process includes three main stages:

1. Problem Analysis and Change Specification: Analyze the problem or change proposal to ensure its validity. Feedback is given to the requester, who can then refine or withdraw the proposal.
2. Change Analysis and Costing: Assess the impact of the proposed change using traceability information and system knowledge. Decide whether to proceed with the change based on this analysis.
3. Change Implementation: Update the requirements document and, if needed, the system design and implementation. The document should be structured to facilitate easy changes.

4.18 ReqChangeMan.eps

Fig. Requirements change management

1 **Evolutionary Prototyping**

Evolutionary prototyping is a way of developing software where a basic version of the system is created, and then improved through feedback from users over several stages. This process continues until the system is complete. There are two main benefits of using this approach:

**1. Faster system delivery**: In fast-changing businesses, it’s important to provide software quickly, and this method helps speed up delivery.

1. **User involvement**: Involving users in the development process ensures the system better matches their needs.

However, there are some challenges:

1. **Management issues**: Traditional management methods might not work well with this fast-paced approach, especially if new technologies are used. It can be hard to find the right people with the necessary skills.
2. **Maintenance problems**: Constant changes to the system can mess up its structure, making it harder for anyone other than the original developers to maintain it.
3. **Contract issues**: Without a clear plan or specification, it can be difficult to create a fair contract. Customers may not want to pay for time spent, and developers may not agree to a fixed price because they cannot predict all the changes users might request.

2 **Rapid Prototyping Techniques**

Rapid prototyping techniques focus on delivering software quickly, prioritizing speed over performance, maintainability, or reliability. There are three main methods:

1. **Dynamic high-level language development**
2. **Database programming**
3. **Component and application assembly**

In this approach, database programming is used to extract data, and reusable components handle detailed processing. Visual programming is often used to design user interfaces. Tools like Lotus Notes support database programming with high-level languages and reusable components linked to database tasks.

Most prototyping systems now use visual programming, where developers manipulate graphical icons instead of writing code, making development faster and reducing costs.

3 **Dynamic High-Level Language Development**

Dynamic high-level languages are programming languages with strong run-time data management features that simplify development by handling storage and management tasks. Examples include Lisp, Prolog, and Smalltalk. These languages weren't commonly used for large systems due to the need for a large run-time support system. The figure shows the dynamic languages commonly used for prototyping. When selecting a language, consider:

1. **Application domain**: Different languages suit different application types.
2. **User interaction**: Some languages offer better support for user interaction.
3. **Support environment**: Mature environments with tools and reusable components make prototyping easier.

Dynamic languages can be used together in one prototype, with each part of the system programmed in the best-suited language. While this approach speeds up development, it can be challenging to establish communication between different languages.

4 **User Interface Prototyping**

Graphical user interfaces (GUIs) are now standard for interactive systems, and developing them represents a significant part of application costs. Designers should involve users in the design process to ensure the interface meets their needs, rather than imposing their own views.

Prototyping is essential for user interface design, as textual descriptions and diagrams are insufficient to capture dynamic interface requirements. Interface generators allow designers to visually select and arrange components like menus and buttons.

Web-based interfaces can be prototyped using website editors, where elements are defined, positioned, and actions are linked using HTML, Java, or CGI scripts.

1. The constructor methods are overloaded in definition.
2. All constructors share the same method names.
3. Any identifiers define within the blocks.
4. A String Tokenizer is a predefined Java Class found in the java.util package.
5. A method is a modularization of data and statements.
6. A class is a modularization of data and methods.
7. By creating a new class that is a subclass of a class in the API via the inheritance mechanism.
8. Screen output is an output stream.
9. A BufferedReader object makes use of the InputStreamReader object, providing for the efficient reading of data.
10. Java has four access modifiers to control the visibility of fields and methods : public, protected, default access (when no access modifier is specified, also known as friendly) and private.
11. A superclass contains common features of its subclasses.
12. An abstract method is a method signature without implementation (i.e. without the braces{..}). The implementation of abstract methods is provided by subclasses.
13. Polymorphism therefore promotes generic programming.
14. Polymorphism is the ability of objects to respond to the same message but with the appropriate method based on their class definition.
15. Java does support Multiple Class Inheritance.
16. The implementation of the method will be provided at run-time and decided by the subclass( such as Class Z). Java provides for this mechanism via the Interface construct.
17. All methods in an interface are abstract methods.
18. An interface body contains constant and method declarations.
19. A String Tokenizer is a predefined JAVA Class found in the java.util package

1. Prototyping is becoming increasingly used for system development where rapid \_development\_\_\_\_\_\_ is essential.

2. The \_prototype\_\_\_\_\_\_ may serve as a basis for deriving a system specification.

3. Throw-away prototyping is used to understand the \_system\_\_\_\_\_\_ requirements.

4. In evolutionary prototyping, the system is developed by \_evolving\_\_\_\_\_\_ an initial version to the final version.

5. Evolutionary prototyping is to deliver a working system to \_\_\_end-users\_\_\_\_.

6. Throw-away prototyping is to validate or derive the system \_\_requirements\_\_\_\_\_.

7. Database programming languages are sometimes known as a \_\_fourth\_\_\_\_\_\_\_\_\_-generation language (4GL).

8. The requirements engineering process is an iterative process including requirements elicitation, specification, and \_validation\_\_\_\_\_\_.

9. Requirements \_management\_\_\_\_\_\_ is the process of managing and controlling these changes.

10. Functional requirements state what the system should \_not\_\_\_\_\_\_ do and should provide, how the system should react to particular inputs.

11. Functional system requirements should describe the system \_\_services\_\_\_\_\_ in detail.

12. Non-functional requirements often apply to the system as a whole\_\_\_\_\_\_\_ rather than individual features or services.

13. Domain requirements are constraints on the system from the domain of \_operation\_\_\_\_\_\_.

14. Non-functional requirements often \_\_\_\_\_\_\_ the system being developed and the development process being used.

15. \_Use-cases\_\_\_\_\_\_ are a scenario-based technique in the Unified Modelling Language.

16. The fourth generation language is used to refer to both the database programming language and its supporting environment\_\_\_\_\_\_\_.

17. A \_\_prototype\_\_\_\_\_ can be used to give end-users a concrete impression of the system's capabilities.

18. Prototyping can be considered as a risk \_\_\_\_\_\_\_ reduction activity which reduces requirements risks.

19. A prototype can be used to give end-users a concrete \_\_impression\_\_\_\_\_ of the system's capabilities.

20. Requirement \_discovery\_\_\_\_\_\_ is the process of gathering information about the required and existing systems.

21. Requirement \_\_validation\_\_\_\_\_ is the process of checking that the requirements actually define the system that the customer really wants.

22. Structured languages notations use \_templates\_\_\_\_\_\_ to specify system requirements.

23. Non-functional requirements arise because of \_budget\_\_\_\_\_\_ constraints, organizational constraints, the need for interoperability with other software or hardware systems, and external factors.

24. The software requirement document includes both user documents for a system and a detailed \_specification\_\_\_\_\_\_ of the system requirements.