Requirements Engineering

Notes by Mannan Ul Haq

Definition:

- **Process**: It's like making a shopping list. You figure out what you need (services) from a system and any rules (constraints) it needs to follow.
- **System Requirements**: These are detailed descriptions of what the system should do and how it should work, made during this process.

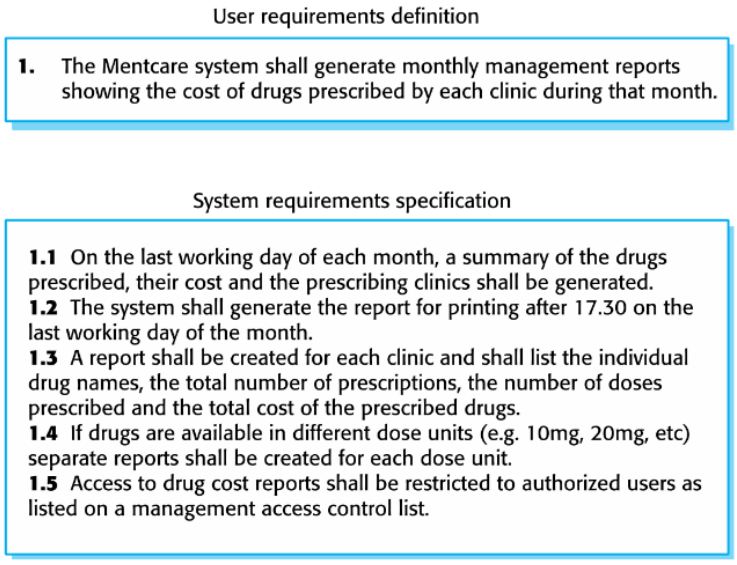
What's a Requirement?

A requirement is essentially a description of something that a system or product must do or a constraint it must adhere to. It can range from broad statements about what a system should achieve to very detailed specifications about how it should behave. Here's a breakdown:

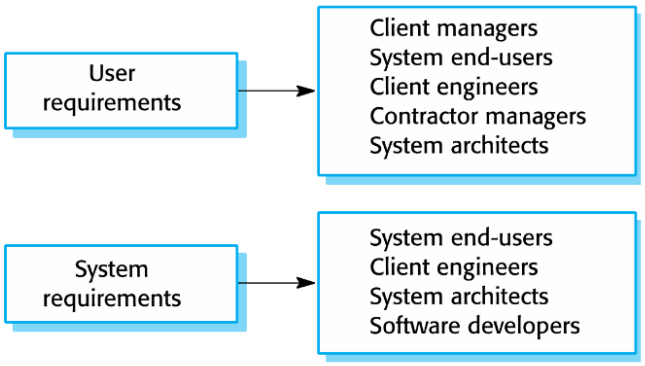
- **Broad Statements**: Sometimes requirements are like big-picture goals. For example, a requirement for a shopping app might be "Allow users to browse and purchase items online."
- **Detailed Specifications**: Other times, requirements get into the nitty-gritty details. For example, a detailed requirement for the same shopping app might specify how users should navigate through categories, how payment should be processed securely, and how orders should be tracked.

Types of Requirements

- **User Requirements**: What the user wants, written in simple language and maybe with pictures. It's like telling a friend what you want from a new phone.
- **System Requirements**: Detailed descriptions of how the system should work, like a blueprint. It's like giving instructions to a builder.



Readers of Different Types of Requirements Specification



System Stakeholders

Definition:

These are people or organizations who are affected by the system and have a genuine interest in it.

Stakeholder Types

1. **End Users**:

   People who directly use the system for their needs.

2. **System Managers**:

   Individuals responsible for overseeing the operation and performance of the system. They ensure everything runs smoothly.

3. **System Owners**:

   Individuals or entities who have ownership or control over the system. They make decisions about its development and use.

4. **External Stakeholders**:

   Individuals or organizations outside of the system but who are impacted by its operation. For example, IT staff responsible for system maintenance.

Example: Stakeholders in the Medical Care System

1. Patients
2. Doctors
3. Nurses
4. Medical Receptionists
5. IT Staff
6. Medical Ethics Manager
7. Health Care Managers

8. Medical Records Staff

Functional and Non-functional Requirements

Functional Requirements:

- **Definition**: These describe what the system should do, how it should respond to different inputs, and its behavior in various situations.
- They focus on defining services the system should provide and how it should function.
- They can also include statements about what the system should not do.

Example: Functional Requirements in the Medical Care System

1. **Appointment List Search**: Users should be able to search appointment lists for all clinics.
2. **Daily Patient Lists**: The system should generate daily patient lists for each clinic.
3. **Staff Identification**: Each staff member using the system should be uniquely identified by their 8-digit employee number.

Non-functional Requirements:

- **Definition**: These are constraints on the system's services or functions, such as time constraints or development process requirements.
- They often apply to the system as a whole rather than specific features.
- **Examples**: Non-functional requirements could include performance standards, security measures, or compatibility with existing systems.

Non-functional Classifications:

1. **Product Requirements:**

   Concerned with how the delivered product behaves, like speed and reliability.

2. **Organizational Requirements:**

   Arise from organizational policies and procedures, such as implementation standards.

3. **External Requirements:**

   Stem from external factors like interoperability demands or legal regulations.

Examples of Non-functional Requirements in the Medical Care System:

1. **Product Requirement:**

   All clinics during regular working hours (Mon-Fri, 0830-17:30). Downtime during these hours should not exceed five seconds per day.

2. **Organizational Requirement:**

   Users must verify their identity using their health authority identity card.

3. **External Requirement:**

The system must adhere to patient privacy provisions outlined in document HStan-03-2006-priv.

Property	Measure
Speed	Processed transactions/second User/event response time Screen refresh time
Size	Mbytes 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

Difference:

Parameters	Functional Requirement	Non-Functional Requirement
What it is	Verb	Attributes
End-result	Product feature	Product properties
Capturing	Easy to capture	Hard to capture
Objective	Helps you verify the functionality of the software.	Helps you to verify the performance of the software.
Area of focus	Focus on user requirement	Concentrates on the user's expectation.
Documentation	Describe what the product does	Describes how the product works
Type of Testing	Functional Testing like System, Integration, End to End, API testing, etc.	Non-Functional Testing like Performance, Stress, Usability, Security testing, etc.
Test Execution	Test Execution is done before non-functional testing.	After the functional testing
Product Info	Product Features	Product Properties

Domain Requirements:

- **Definition**: These are constraints specific to the operational domain of the system.
- They influence how the system operates within its specific industry or field.
- **Example**: In a medical system, domain requirements might include compliance with healthcare regulations or interoperability with other medical systems.

Usability Requirements

The system should be easy for user to use, minimizing user errors.

Example:

- Medical staff must master all system functions within four hours of training.
- Experienced users should make no more than two errors per hour after training.

Requirements Imprecision

Definition:

- **Problem**: Occurs when functional requirements lack precision or clarity.
- Ambiguous requirements can lead to different interpretations by developers and users.

Example: Consider the term 'search' in a requirement.

Interpretation Differences:

- **User Intention**:

  Wants to search for a patient's name across all appointments in all clinics, simplifying the process.

- **Developer Interpretation**:

  May understand it as searching for a patient's name within an individual clinic, requiring the user to choose a clinic first.

Writing a Good Requirement

Unambiguous:

Only one interpretation should be possible.

Ambiguity can arise from undefined acronyms.

- Example: "The system shall be implemented using ASP."
- Clarification: "The system shall be implemented using Active Server Pages (ASP)."

Testable (Verifiable):

Requirements should be measurable and quantifiable.

Avoid vague terms or acronyms.

- Example: "The system shall resist concurrent usage by many users."
- Clarification: Define what "many" means in terms of numbers.

Clear (Concise, Simple, Precise):

Avoid unnecessary verbiage or complexity.

Ensure requirements are stated clearly and simply.

- Example: Simplify complex sentences.

  Original: "Sometimes the user will enter Airport Code, which the system will understand, but sometimes the closest city may replace it, so the user does not need to know what the airport code is, and it will still be understood by the system."

Simplified: "The system shall identify the airport based on either an Airport Code or a City Name."

Correct:

Ensure that facts stated in requirements are accurate.

Validate information provided in requirements.

- Example: Make sure provided figures are correct.

  Original: "Car rental prices shall show all applicable taxes (including 6% state tax)."

  Correction: Ensure tax rates are accurate for all states.

Understandable:

Requirements should be grammatically correct and consistent.

Use standard conventions for clarity.

- Example: Use "shall" instead of "will," "must," or "may."

  Ensure requirements are written in a consistent style and format.

Feasible (Realistic, Possible):

Ensure requirements are achievable within existing constraints.

Consider time, budget, and available resources.

- Example: Ensure requirements are realistically implementable.

  Check if the requirement aligns with available resources and timeframes.

Independent:

Requirements should stand alone without relying on other requirements.

Avoid dependencies between requirements.

- Example: Ensure each requirement is self-contained.

  Avoid using pronouns like "it" that refer to other requirements.

Atomic:

Each requirement should address a single, traceable element.

Avoid combining multiple requirements into one.

- Example: Break down complex requirements into simpler, atomic ones.

  Avoid sentences containing "and" or "but" that combine multiple actions.

Necessary:

Ensure that each requirement adds value and is essential.

Remove unnecessary requirements that do not contribute to system functionality.

- Example: Remove requirements that are not needed by stakeholders.

  Ensure each requirement serves a valid purpose for stakeholders.

Implementation Free (Abstract):

Avoid including design or implementation details in requirements.

Keep requirements focused on functionality, not implementation.

- Example: Remove specifics about how something is achieved.

  Focus on what the system should do, not how it should do it.

Consistent:

Ensure requirements do not conflict with each other.

Check for conflicts, both direct and indirect, between requirements.

- Example: Ensure requirements related to the same functionality align.

  Resolve conflicts between requirements related to the same feature.

Nonredundant:

Each requirement should be unique and not duplicate others.

Avoid overlapping or repetitive requirements.

- Example: Ensure each requirement addresses a distinct aspect of functionality.

  Remove redundant requirements that duplicate others.

Complete:

Ensure all possible conditions are covered by requirements.

Address all scenarios and edge cases.

- Example: Ensure all possible scenarios are accounted for.

  Check for completeness by considering all possible user interactions.

Conflicts:

Resolve conflicts between different nonfunctional requirements.

Prioritize conflicting requirements based on criticality.

- Example: Resolve conflicts between performance and power consumption requirements.

  Determine which requirement is more critical and prioritize accordingly.