
بسم الله الرحمن الرحيم

Software Requirement Engineering

LECTURER: SYED HASNAIN ABBAS BUKHARI

Functional and Non Functional Requirements

- **Functional Software Requirements**
  - Functional requirements express how the system behaves
  - These requirements are usually action oriented
    - "When the user does a, the system will do b.“
- **Non-functional Software Requirements**
  - These requirements are used most typically to express some of the "attributes of the system" or "attributes of the system environment“
  - Such as reliability
- Requirements = FURPS+ (Functionality, Usability, Reliability, Performance and Supportability model.)

Non-functional requirements (NFR)

- Non-functional requirements define the overall qualities or attributes of the resulting system
- Non-functional requirements place restrictions on the product being developed, the development process, and specify external constraints that the product must meet.
- Examples of NFR include safety, security, usability, reliability and performance requirements.

Functional and Non-functional requirements

- It is difficult to make a clear distinction between functional and non-functional requirements.
- Whether or not a requirement is expressed as a functional or a non-functional requirement may depend:
  - on the level of detail to be included in the requirements document
  - the degree of trust which exists between a system customer and a system developer.

Types of Non-functional requirements

- **The 'IEEE-Std 830 - 1993' lists 13 non-functional requirements to be included in a Software Requirements Document.**
  - Performance requirements
  - Interface requirements
  - Operational requirements
  - Resource requirements
  - Verification requirements
  - Acceptance requirements
  - Documentation requirements
  - Security requirements
  - Portability requirements
  - Quality requirements
  - Reliability requirements
  - Maintainability requirements
  - Safety requirements

ISO 9126 quality attributes refinement

Characteristic	Sub characteristics	Short definition
functionality	accuracy	provision of right or agreed results or effects
	compliance	adherence to application related standards or conventions
	interoperability	ability to interact with specified systems
	security	prevention to unauthorized access to data
	suitability	presence and appropriateness of a set of functions for specified tasks
reliability	fault tolerance	ability to keep a given level of performance in case of faults
	maturity	frequency of failure by faults in the software
	recoverability	capability of reestablish level of performance after faults
usability	learnability	users' effort for learning software application
	operability	users' effort for operation and operation control
	understandability	users' effort for recognizing sw. structure and applicability

ISO 9126 quality attributes refinement

efficiency	resource behavior	amount of resources used and the duration of such use
	time behaviour	response and processing times and throughput rates
maintainability	analysability	identification of deficiencies, failure causes, parts to be modified, etc.
	changeability	effort needed for modification, fault removal or environmental change
	stability	risk of unexpected effect of modifications
	testability	effort needed for validating the modified software
portability	adaptability	opportunity for adaptation to different environments
	conformance	adherence to conventions and standards related to portability
	installability	effort needed to install the software in a given environment
	replaceability	opportunity and effort of using software replacing other

Relationships between user needs, concerns and NFRs

User's need	User's concern	Non-functional requirement
Function	1. Ease of use	1. Usability
	2. Unauthorised access	2. Security
	3. Likelihood of failure	3. Reliability
Performance	1. Resource utilisation	1. Efficiency
	2. Performance verification	2. Verifiability
	3. Ease of interfacing	3. Interoperability
Change	1. Ease of repair	1. Maintainability
	2. Ease of change	2. Flexibility
	3. Ease of transport ?	3. Portability
	4. Ease of expanding or upgrading capacity or performance ?	4. Expandability

Testable NFRs

- **Stakeholders may have vague goals which cannot be expressed precisely**
- **Vague and imprecise ‘requirements’ are problematic**
- **NFRs should satisfy two attributes**
  - Must be objective
  - Must be testable (Use measurable metrics)
- **Product shall be easy to learn**
- **The product shall be able to be used by a member of the public without training**
- **90% of a panel representative of the general public shall successfully purchase a ticket from the product on their first encounter**

Examples of measurable metrics for NFRs

Property	Metric
Performance	1. Processed transactions per second 2. Response time to user input
Reliability	1. Rate of occurrence of failure 2. Mean time to failure
Availability	Probability of failure on demand
Size	Kbytes
Usability	1. Time taken to learn 80% of the facilities 2. Number of errors made by users in a given time period
Robustness	Time to restart after system failure
Portability	Number of target systems

Usability

- Usability encompasses several subdomains beyond the obvious ease of use, including ease of learning; memorability; error avoidance, handling, and recovery; efficiency of interactions; accessibility; and ergonomics.

Usability Indicators

- The average time needed for a specific type of user to complete a particular task correctly.
- How many transactions the user can complete correctly in a given time period.
- What percentage of a set of tasks the user can complete correctly without needing help.
- The number of interactions (mouse clicks, keystrokes, touch-screen gestures) required to get to a piece of information or to accomplish a task.

Reliability

- The probability of the software executing without failure for a specific period of time is known as reliability.
- **Reliability Indicators**
  - **Mean time to repair**
    - How long is the system allowed to be out of operation after it has failed?
  - **Accuracy**
    - What precision is required in systems that produce numerical outputs?
  - **Defect rate**
    - Bugs/KLOC or bugs per function-point

Constraints

- Limitations on the design or implementation of the system or the processes we use to build a system
- Restrictions on the design of a system, or the process by which a system is developed, that do not affect the external behavior of the system but that must be fulfilled to meet technical, business, or contractual obligations

Constraints

- **Operating environments**
  - Write the software in Visual Basic
- **Compatibility with existing systems**
  - The application must run on both our new and old platforms
- **Corporate "best practices" and standards:**
  - Compatibility with the legacy data base must be maintained
  - Use our C++ coding standards
- **Project Regulations and Standards**
  - Food and Drug Administration (FDA)
  - Federal Communications Commission (FCC)
  - ISO

How to Write Requirements

- A good requirement states something that is **necessary**, **verifiable**, and **attainable**.
- A good requirement should be clearly stated.
- Each requirement should express a single thought, be concise, and simple.
- It is important that the requirement not be misunderstood, it must be unambiguous.

Common Problems In Writing Requirements

- Making bad assumptions
- Writing implementation (HOW) instead of requirements (WHAT)
- Describing operations instead of writing requirements
- Using incorrect terms
- Using incorrect sentence structure or bad grammar
- Missing requirements
- Over-specifying

Requirement Structure/Grammar

- Requirements should be easy to read and understand.
- Each requirement can usually be written in the format.
  - *The System shall provide ...*
  - *The System shall be capable of ...*
  - *The System shall weigh ...*
  - *The Subsystem #1 shall provide ...*
  - *The Subsystem #2 shall interface with ...*
- Each of these beginnings is followed by *WHAT* the System or Subsystem shall do.
- State ***what*** the system must do, ***not how*** it must do it.

Bad Grammar

- If you use bad grammar you risk that the reader will misinterpret what is stated.

- If you use the requirements structure suggested previously, you will eliminate the bad grammar problems that occur when authors try to write complex sentences.

Examples

- **Functional Requirements:**
  - Customer should be able to list products
  - Customer shall be able to list products
  - Customer will be able to list products

More than one way to write requirements

  - Customer should be able to place order
  - Customer shall be able to place order
  - Customer will be able to place order

More than one way to write requirements

  - Customer should be able to pay by card
  - Customer shall be able to pay by card
  - Customer will be able to pay by card

More than one way to write requirements

- **Non-functional Requirements:**

1. The Site should be able to serve a 128 number of customers at the same time without problems..