CHAPTER

3

Software Requirements

Learning Outcomes

In this chapter you will learn:

- Basic terminology in software requirements
- Product vision and project scope document
- Difference between functional and non-functional requirements (NFRs)
- Types of NFRs
- Requirements engineering process steps
- Techniques for requirements elicitation
- Development of a use case model
- Validation of software requirements
- IEEE standard for writing a software requirements specifications document

Elicitation of NFRs

- Two types: generic and use case specific
- Generic (system wide – applies to all use cases) NFRs like overall performance, memory requirements, standards, legal, …
- Use case specific NFRs like security for certain functions
- Technical and non-technical NFRs
- Quantitative or qualitative NFRs
- To address user or developer concerns

Table 3.6 Types of non-functional requirements

	Mainly a user concern	**Mainly a developer concern**
Technical NFRs	Interoperability Performance Reliability Availability Robustness Security Safety Usability User friendliness	Implementation Installation Maintainability Operational Portability Reusability Scalability Testability Traceability
Non-technical NFRs	Business integrity Cultural Political	Legal Conformity to standards

Interoperability Requirement

- imposes constraints on the types of systems to interface and communicate with.
  - other software systems and hardware devices.
- “The system must be able to interface with all printers manufactured by company X.”.
  - This requirement imposes some dependability on the proper specification and implementation of the interface with these printers.
- indicative of some potential external risks to the development process.

Maintainability Requirement

- imposes constraints related to the ease with which the software can be fixed, adapted to new environments and technologies, or expanded.
- To meet these requirements, various technical and non-technical measures need to be taken.
  - development environment and tools used and the development methodologies and models adopted.
- Examples of non-technical measures include hiring decisions and appropriate developers training programs.

Operational Requirement

- imposes constraints on the physical and logical environments in which the software operates.
- can include
  - the characteristics of the environment in which the servers are physically located,
  - minimum speed of the network connections,
  - the operating systems that are compatible, and
  - the hardware devices the software interfaces with.

Performance Requirement

- Imposes constraints on the response time delays, throughput, and memory requirements.
  - Response time delays can be either system wide or use case specific.
  - Throughput requirements by putting lower bounds on the acceptable rate of completed transactions.
- ‘The Update Salaries use case should process at least one million records in 10 seconds’.
- ‘The maximum memory needed to run the system shall not exceed 1 Mbyte of RAM’

Portability Requirement

- Imposes conditions related to the future deployment and evolution of the software
- Ease with which the software can be made to run on a different hardware platform or software environment.
- “The system should be able to run under Mac OS with the least development effort”.
  - To meet this requirement, many constraints related to the selected development environment and design choices, and the implementation programming language used have to be imposed.

Reliability and Availability Requirements

- Impose some values related to the reliability of the software under development.
- Typical values include the mean time between failures (MTBF), or the maximum time allowed for failures over a period of time.
- Can be used to impose an upper limit on the down time of the software, indicating an acceptable level of failures.
- Availability requirements are also part of the security requirements.

Reusability Requirement

- Imposes constraints on the development of the system related to the degree of reuse.
- 2 types of reuse:
  - Development **with reuse** aims at producing software faster by **reusing existing** software components.
  - Development **for reuse** aims at producing highly maintainable software so it can be reused in future software

Robustness Requirement

- Imposes constraints related to the way the software handles abnormal and erroneous inputs and operational conditions.
  - should not state how to achieve robustness or to suggest a particular technological solution to do it.
- Addresses the software behavior with respect to error recoverability and fault tolerance.

Robustness Requirement

- System wide or use case specific requirements.
  - 'The system must recover from power failures and continue operating without any side effect on the currently executing transactions'.
  - 'Completing the Create Customer use case shall be guaranteed even in the case of a database server failure'.
  - 'in case of software or hardware failure, all committed transactions will not be affected'

Safety Requirement

- Needed when the software deals with safety critical issues such aviation software, power and chemical plants control software.
- Addresses the enforcement of safety standards known in the particular application domain.
- Needed expertise in the domain of the safety critical application to avoid legal consequences.
  - in chemical plant control software, a fail-safe safety requirement would state that: ‘in the case of software failure, a safety procedure XYZ must be activated immediately to ensure no chemical leakage occurs’.

Scalability Requirement

- Imposes constraints on how the software system should scale up to a high input load at all interfaces.
- “The system must be able to accommodate a maximum of 100 users logged on and performing all types of transactions”.
  - May force specific architectural design choices on the development team.
- Scalability requirements are ideally quantifiable and are normally verified during load testing

Security Requirement

- Very critical nowadays and need to be identified earlier in the software development process.
  - can be either a system wide or use case specific
- Addresses 4 main types of security concerns, namely, confidentiality, integrity, availability and accountability (CIAA).
- Dealt with by imposing and adhering to
  - identification, authentication, authorization, integrity, immunity, privacy, non-repudiation, survivability, physical protection, and security standards conformity requirements.

Security Requirement

- **Access control** related requirements
  - **identification, authentication and authorization** (IAA) requirements are used to address **confidentiality** concerns.
  - Physical protection requirements are useful to address confidentiality concerns at the physical level.
- IAA requirements: system wide or use case based.
  - Client may require that certain functionalities be only accessible by identified, authenticated and authorized users, and others to be publicly and unanimously accessible.

Security Requirement

- **Integrity** concerns are addressed using immunity and privacy requirements.
  - **privacy** requirement: ‘the software shall allow a registered user to have control over its own private information’.
  - **immunity** requirement: ‘no files can be attached to a user profile without being disinfected by the software’.
- Can be system wide or use case specific.
  - ‘all files saved by the software must be tamperproof so that any malicious and unauthorized modification of a file shall be detected’.

Security Requirement

- **Availability** concerns are dealt with using the survivability and physical protection requirements.
- A **survivability** requirement could be
  - system wide: ‘the system shall continue to provide its functions even if one of its nodes fails’.
  - use case specific: ‘the user validation shall always be performed regardless of some network nodes failures’.
- A **physical protection** requirement that addresses availability concerns: ‘the database servers shall be protected in fireproof data centers’.

Security Requirement

- **Accountability** concerns are dealt with using non-repudiation, and standards conformity requirements.
- **Non-repudiation** requirement: 'The software shall preserve a tamper-proof record of every privileged access to the system'.
  - Can be either system wide or applicable to specific functions or use cases of the system.
- **Standards conformity** requirements are normally system wide: 'The software and its operating environment shall conform to the ISO

Security-related FRs

- Some of them must be implemented by first identifying the appropriate security related functional requirements.
  - IAA requirements are considered by introducing a logon use case as a functional requirement, also called **security use cases**.
  - A **misuse case**, in which the involved actors are called **misusers**. It represents the abnormal behavior of an attacker or misuser, and specifies the software reaction to the specified misuse.

Testability Requirement

- Imposes constraints on the design choices and future testing of the developed software.
- Defined as the ease with which testing is performed.
- Design for testability: observability & controllability
  - Observing the execution inside the software - debugging
  - Controlling the execution inside the software

Testability Requirement

- The ease of maintenance (maintainability) is positively correlated with the ease of testing.
- Quantifiable requirement: “Testing the software must be completed in 24 hours”.
  - This requirement imposes constraints on the test automation and the tools used for testing.

Traceability Requirement

- Imposes some constraints on the ease with which the software is traceable.
- Related to the traceability of the different parts of the software deliverables
  - ability to link forward and backward every aspect of the software deliverables.
  - Each module in the software design can be traced backward to a requirement specification element, or forward to a particular piece of code or test cases, hence, enhancing the software maintainability
- “Every functional requirement must be traced forward in the design, code and testing docs”.

Traceability Requirement

- Related to the traceability of the software during its execution.
  - traces collected during the execution of the software may be needed. Software execution traces are used for testing and debugging purposes, hence enhancing the software testability (i.e., observability) to meet the software testability requirements.
  - helps meeting the auditability and non-repudiation aspects of security requirements.
- “All activities of an admin user session must be recorded by the system”

Usability Requirement

- Imposes constraints by the client in its representation of the user community.
  - aim at making the software easy to use by the different types of users
- Elicited by first knowing the intended users and their backgrounds and characteristics.
  - first to undertake in a usability engineering process.
- ‘Software must be easy to use by 7-9 year old children’.
- “80% of Students between ages 10 and 14 must be able to create a demo within 1 hour”.

User friendliness Requirement

- Imposes constraints on the user experiences when interacting with the software.
  - have implications on the functional requirements and the graphical user interface design decisions.
- The availability of context sensitive help versus generic help, a forgiving and courteous interface, and the ease of navigability are examples of user friendliness requirements.

User friendliness Requirement

- Usability and user friendliness requirements complement each other and are often being confused as equivalent.
- A system may be easy to use or usable but may not be user friendly.
- However, ideally, a highly usable system is normally a user friendly system.
- Issues related to graphical user interface design addressing the user friendliness and usability requirements will be studied in Chapter 5.

Cultural and Political Requirements

- Address the cultural and political constraints to consider when developing the software.
- The analyst and client must be aware of the cultural sensitivities of the countries in which the software will be deployed and used.
  - language use issues, the use of symbols, and politically offensive contents, among other things
- Failure to elicit them affects its acceptance.
- May conflict for different countries or societies
  - many versions should be developed and deployed.

Legal and Regulatory Requirements

- Address the legal and regulatory issues related to the software.
  - Adherence to local and international laws and regulations may have to be observed by the software and the process by which it is developed.
  - For national security concerns, the team must not include people who have resided in certain countries.
  - software be in line with copyright laws and regulations.
- Overlooking them may lead to lawsuits and criminal investigations affecting the economic feasibility and reputation of the development company.

Conformity to Standards Requirement

- Indicates the standards that must be followed while developing the software system.
- **Internal standards** that are developed by the software development company may include coding and testing standards, and documentation standards and templates.
- Specific **country standards** may have to be followed and the developer and client must be aware of them.

Conformity to Standards Requirement

- **Military standards** exist and need to be followed if the military is one of the software stakeholders. Industry standards may have to be adhered to.
- **Specific standards** exist for different sectors such as the health, finance, construction, and education sectors.
- **Professional standards** can also be referred to in this type of requirements. (like IEEE standards)

Conformity to Standards Requirement

- **International standards** such as those developed by the international standardization organization (ISO) and the International Telecommunications Union (ITU)
- The software analyst must be aware of the different standards related to the application domain.
- Non-adherence to known and relevant standards may lead to unusable software.

Example 3.2

Examples of generic system-wide NFRs for the OPS system are:

a. Must be accessible from PCs, Sun Workstations, or Macintoshes using at least Internet Explorer and Netscape web browsers
b. Must be modular software made of reusable components that are easily modifiable
c. Must have speed of access to OPS and must be, as much as possible, location-transparent
d. Must be portable to additional platforms with the least effort
e. Must be highly available with only an average of one hour a week of downtime to allow for upgrades
f. Must allow for 100 simultaneous user accesses during the first year of operations and 300 thereafter
g. Must allow encryption for private user data and only users or programs with the right access control are allowed to decrypt this data
h. Must log all critical transactions, including ordering, modifying order, or canceling order for future use
i. Must be easy to use by novice computer users by providing contextual guidance throughout the interface
j. Must allow language support, depending on the country from which it is accessed

Documenting NFRs

- NFRs can be documented separately in the software requirement document if they are generic.
- However, if they are specific to a particular use case, they can be attached closer to the use case description.
  - The constraints section of the use case can include specific NFRs applicable to that use case.

Alternative flows	Use case can be interrupted at any point by the use cases E1 to E7. Use case is discontinued when interrupted by the use cases: E1 to E4. However, the use case continues after being interrupted by use cases E5 to E7.
Used use cases (include)	Add book to basket Remove book from basket Modify order Clear order Save order Send order Provide payment information Check credit information Check book availability
Interrupting or extending use cases (extend)	E1: Handle timeout E2: Handle communications failure E3: Handle server failure E4: Handle cancel E5: Handle invalid payment information E6: Handle book unavailability E7: Help place order
Non-functional requirements	See Section 3.5
Remarks and issues	

Example 3.3

Examples of NFRs for the Place Order use case are:

a. Private data in the Place Order form must be encrypted in the order database
b. Timeout should occur if the Place Order form is not submitted within 20 minutes
c. Confirmation to the Place Order form, once submitted, must occur within 30 seconds
d. Cancellation and help should be available at any time during the ordering process

Requirements validation

- Using review meetings, prototyping, checklists, walkthroughs
- Checking the desirable properties of requirements
- Verifiability – Is it possible to verify the conformity of the software product to the requirement ? (both FR and NFR)
  - FR are visible, NFRs should be as quantifiable as possible

Desirable properties

- correctness
- completeness
- clarity and comprehensibility
- non-ambiguity
- consistency
- feasibility
- modifiability and maintainability
- testability
- traceability
- verifiability
- reusability
- ranking for importance and criticality

Requirements maintenance

- Automating the process
- System to deal with requirement change
  - Change request and approval
  - Tracability
- Requirements maintenance metrics
- Part of a configuration management plan and tool

IEEE 830 – SRS document

- The heart of requirements and specifications in Section 3 (Chapters 3 & 4)
- Organize it by Use Cases
- Use case can include use case specific NFRs
- Section 3 includes also system wide NFRs
- Data model, behavior model and process model – Section 3 can be completed after finishing Chapter 4 (Specification)

Table 3.7 Template used in the IEEE recommended practice for software requirements specifications

Title page
Change history
Preface
Table of contents
1. Introduction
    1.1 Purpose and intended audience
    1.2 Scope
    1.3 Definitions, acronyms, and abbreviations
    1.4 References
    1.5 Overview of the document
2. Overall description
    2.1 Product perspectives and context
        2.1.1 System interfaces
        2.1.2 User interfaces
        2.1.3 Hardware and software interfaces
        2.1.4 Communications interfaces
        2.1.5 Memory constraints
        2.1.6 Operational context
    2.2 Product functions
    2.3 User characteristics
    2.4 General developmental constraints
    2.5 Project assumptions and dependencies

3. Specific requirements
    3.1 External interfaces
    3.2 Functional requirements
    3.3 Performance requirements
    3.4 Database requirements
    3.5 Design constraints and standards compliance
    3.6 Software system attributes (Non-functional requirements)
4. Supporting information
    4.1 Appendices
    4.2 Index