Tribhuvan University

Institute of Engineering

Pulchowk Campus

Department of Electronics and Computer Engineering

Software Engineering

Chapter One

Software Process and requirements

by

Er. Bishwas Pokharel

Lecturer, Kathford Int'l College of Engg. and Mgmt.

Date: 3rd jan, 2018

Chapter One: **Software Process and requirements**

Course Outline: **12 hours, 20 Marks**

1.1. Software crisis

1.2. Software characteristics

1.3. Software quality attributes

1.4. Software process model

1.5. Process iteration

1.6. process activities

1.7. Computer-aided software engineering

1.8. Functional and non –functional requirements

1.9. User requirements

1.10. System requirement

1.11. Interface specification

1.12. The software requirements documents

1.13. Feasibility study

1.14. Requirements elicitation and analysis

1.15. Requirements validation and management

Solve it!!

If a and b are positive integers, and $x = 2 \cdot 3 \cdot 7 \cdot a$, and $y = 2 \cdot 2 \cdot 8 \cdot b$, and the values of both x and y lie between 120 and 130 (not including the two), then $a - b =$

(A) –2

(B) –1

(C) 0

(D) 1

(E) 2

ans:

The only multiple of 42 in this range is $42 \times 3 = 126$. Hence, $x = 126$ and $a = 3$. The only multiple of 32 in this range is $32 \times 4 = 128$. Hence, $y = 128$ and $b = 4$. Hence, $a - b = 3 - 4 = -1$. The answer is (B).

What is requirement?

Requirements describe how a system should act, appear, or perform. For this, when users request for software, they possess an approximation of what the new system should be capable of doing. Requirements differ from one user to another user and from one business process to another business process.

What is software requirement?

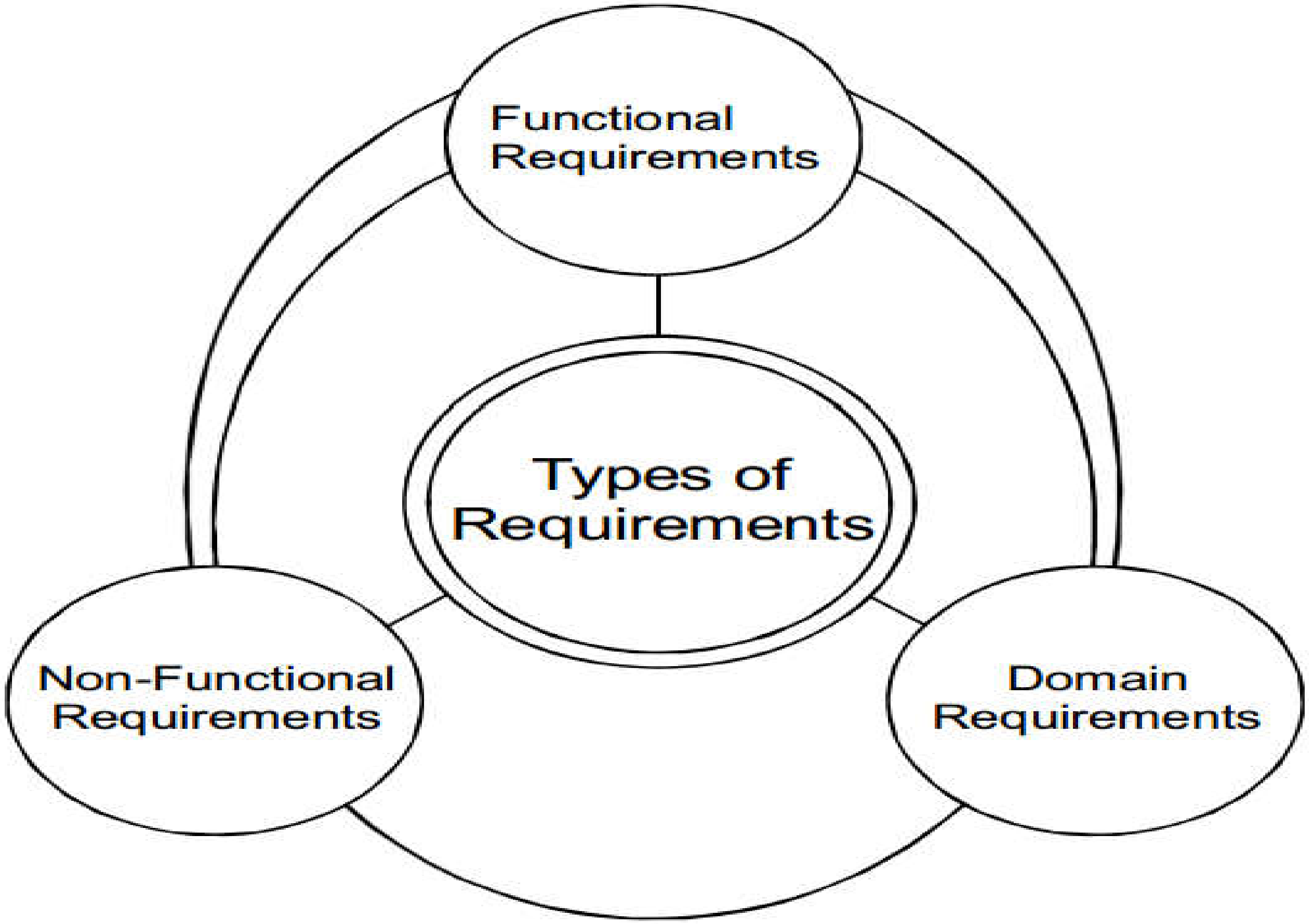
IEEE defines requirement as

"(1) A condition or capability needed by a user to solve a problem or achieve an objective. (2) A condition or capability that must be met or possessed by a system or system component to satisfy a contract, standard, specification, or other formally imposed documents. (3) A documented representation of a condition or capability as in (1)or (2)"

Guidelines for Expressing Requirements

- Sentences and paragraphs should be short and written in active voice. Also, proper grammar, spelling, and punctuation should be used.
- Conjunctions, such as ‘and’ and ‘or’ should be avoided as they indicate the combination of several requirements in one requirement.
- Each requirement should be stated only once so that it does not create redundancy in the requirements specification document

Types of Requirements



a. Functional Requirements

To understand this, let us consider an example of an online banking system, which is listed below:

- The user of the bank should be able to search the desired services from the available services.
- There should be appropriate documents for users to read. This implies that when a user wants to open an account in the bank, the forms must be available so that the user can open an account.
- After registration, the user should be provided with a unique acknowledgement number so that the user can later be given an account number

The above-mentioned functional requirements describe the specific services provided by the online banking system. These requirements indicate user requirements and specify that functional requirements may be described at different levels of detail in online banking system. With the help of these functional requirements, users can easily view, search, and download registration forms and other information about the bank. On the other hand, if requirements are not stated properly, then they are misinterpreted by the software engineers and user requirements are not met.

The functional requirements (also known behavioral requirements) describe the functionality or services that software should provide.

IEEE defines function requirements as

"a function that a system or component must be able to perform."

Functional requirements describe the interaction of software with its environment and specify the inputs, outputs, external interfaces, and the functions that should not be included in the software. .

The functional requirements should be complete and consistent.

Completeness:

Implies that all the user requirements are defined.

Consistency:

implies that all requirements are specified clearly without any contradictory

b. Non-functional Requirement

- These requirements are not related directly to any particular function provided by the system.
- The non-functional requirements (also known as **quality requirements**) relate to system attributes, such as reliability and response time.
- Non-functional requirements should be accomplished in software to make it perform efficiently. For example, if an aero plane is unable to fulfill reliability requirements, it is not approved for safe operation.

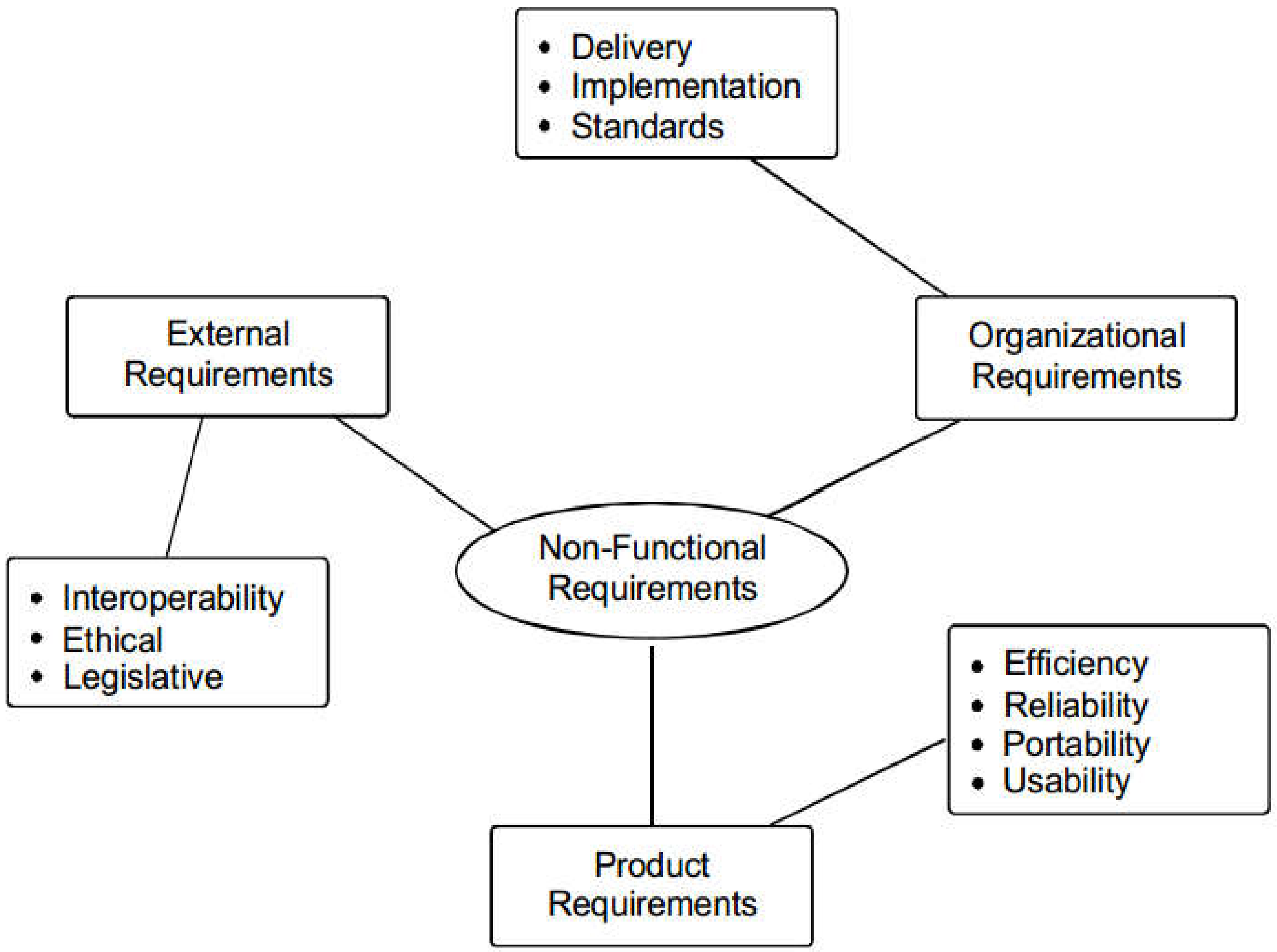


Fig: types of non functional requirements

Product requirements:

These requirements specify how software product performs.

- **Efficiency requirements**: Describe the extent to which software makes optimal use of resources, the speed with which system executes, and the memory it consumes for its operation. For example, system should be able to operate at least three times faster than the existing system.

Reliability requirements: Describe the acceptable failure rate of the software. For example, software should be able to operate even if a hazard occurs.

Portability requirements: Describe the ease with which software can be transferred from one platform to another. For example, it should be easy to port software to different operating system without the need to redesign the entire software.

Usability requirements: Describe the ease with which users are able to operate the software. For example, software should be able to provide access to functionality with fewer keystrokes and mouse clicks.

Organizational requirements:

These requirements are derived from the policies and procedures of an organization.

- **Delivery requirements:** Specify when software and its documentation are to be delivered to the user
- **Implementation requirements:** Describe requirements, such as programming language and design method.
- **Standards requirements:** Describe the process standards to be used during software development. For example, software should be developed using standards specified by ISO (International Organization for Standardization) and IEEE standards

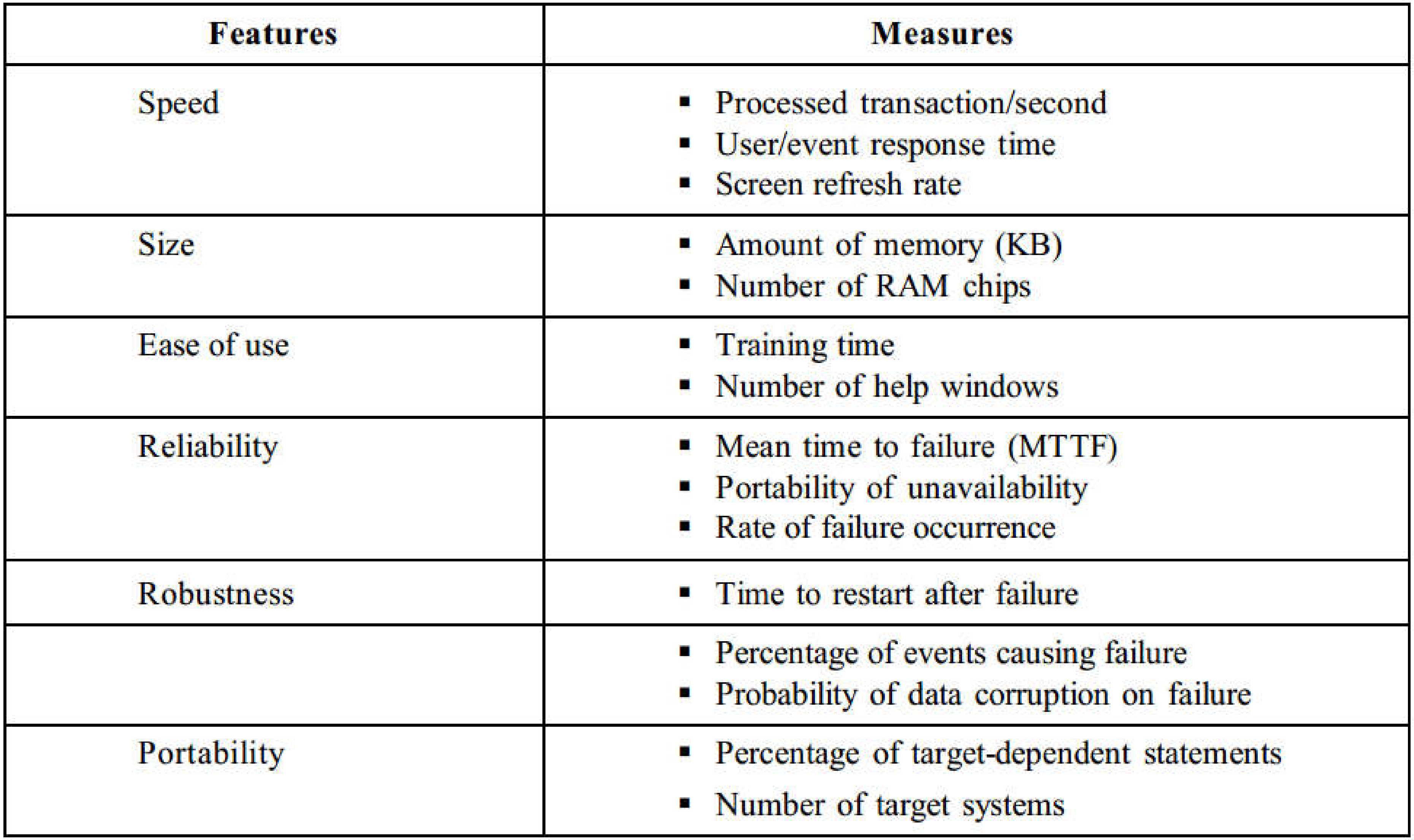
External requirements:

include all the requirements that affect the software or its development process externally.

Interoperability requirements: Define the way in which different computer-based systems interact with each other in one or more organizations.

Ethical requirements: Specify the rules and regulations of the software so that they are acceptable to users.

Legislative requirements: Ensure that software operates within the legal jurisdiction. For example, pirated software should not be sold.



Features	Measures
Speed	▪ Processed transaction/second ▪ User/event response time ▪ Screen refresh rate
Size	▪ Amount of memory (KB) ▪ Number of RAM chips
Ease of use	▪ Training time ▪ Number of help windows
Reliability	▪ Mean time to failure (MTTF) ▪ Portability of unavailability ▪ Rate of failure occurrence
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

Fig: Metrics for non functional requirements

Domain Requirements:

Requirements derived from the application domain of a system, instead from the needs of the users are known as domain requirements.

For example: It is important for a development team to create databases and interface design as per established standards. Similarly, the requirements requested by the user, such as copyright restrictions and security mechanism for the files and documents used in the system are also domain requirements.

Requirements Engineering Process

The requirements engineering (RE) process is a series of activities that are performed in requirements phase in order to express requirements in **software requirements specification (SRS) document**.

These steps include feasibility study, requirements elicitation, requirements analysis, requirements specification, requirements validation, and requirement management

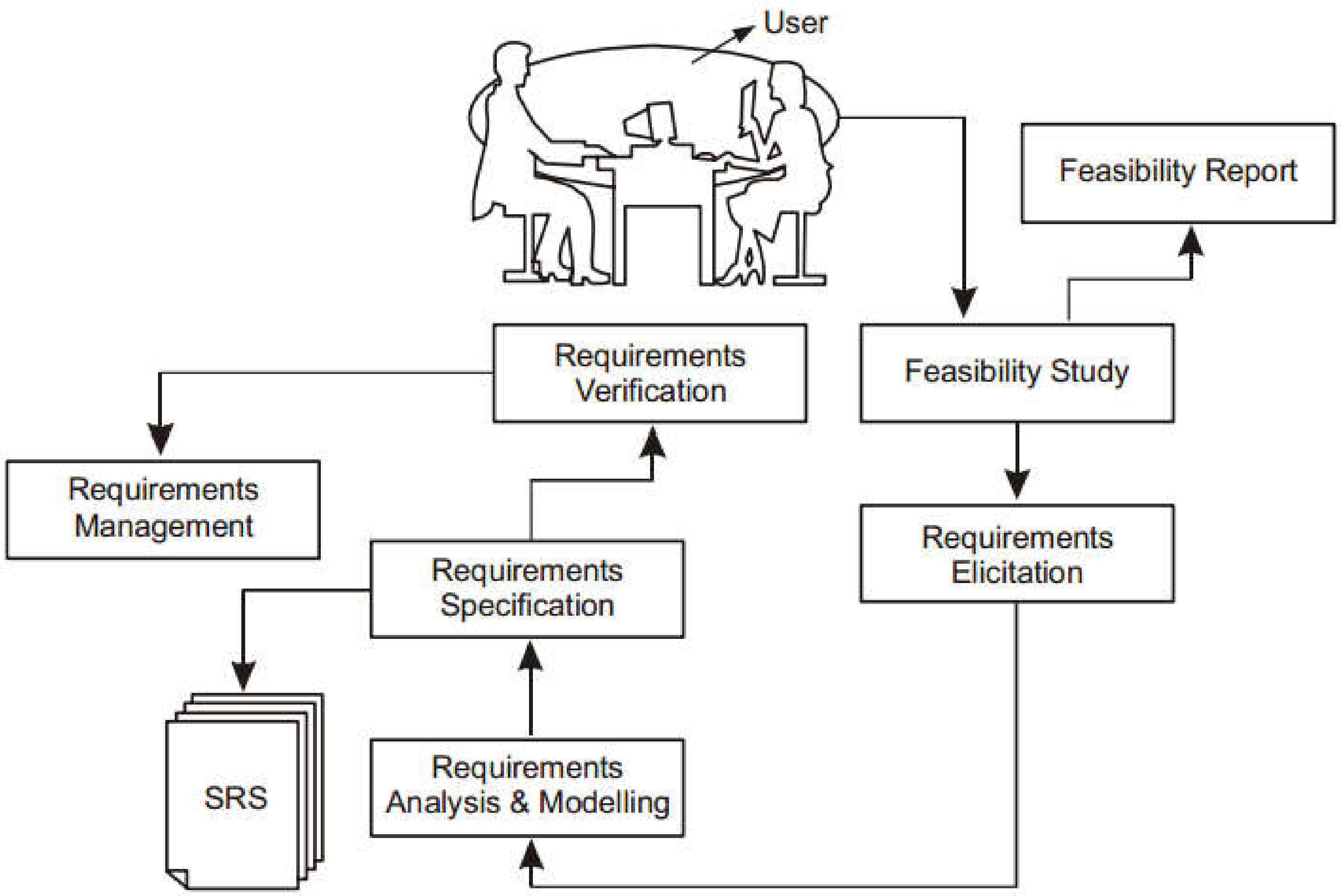


Fig: Requirement engineering process

STEP 1: FEASIBILITY STUDY

Feasibility is defined as the practical extent to which a project can be performed successfully

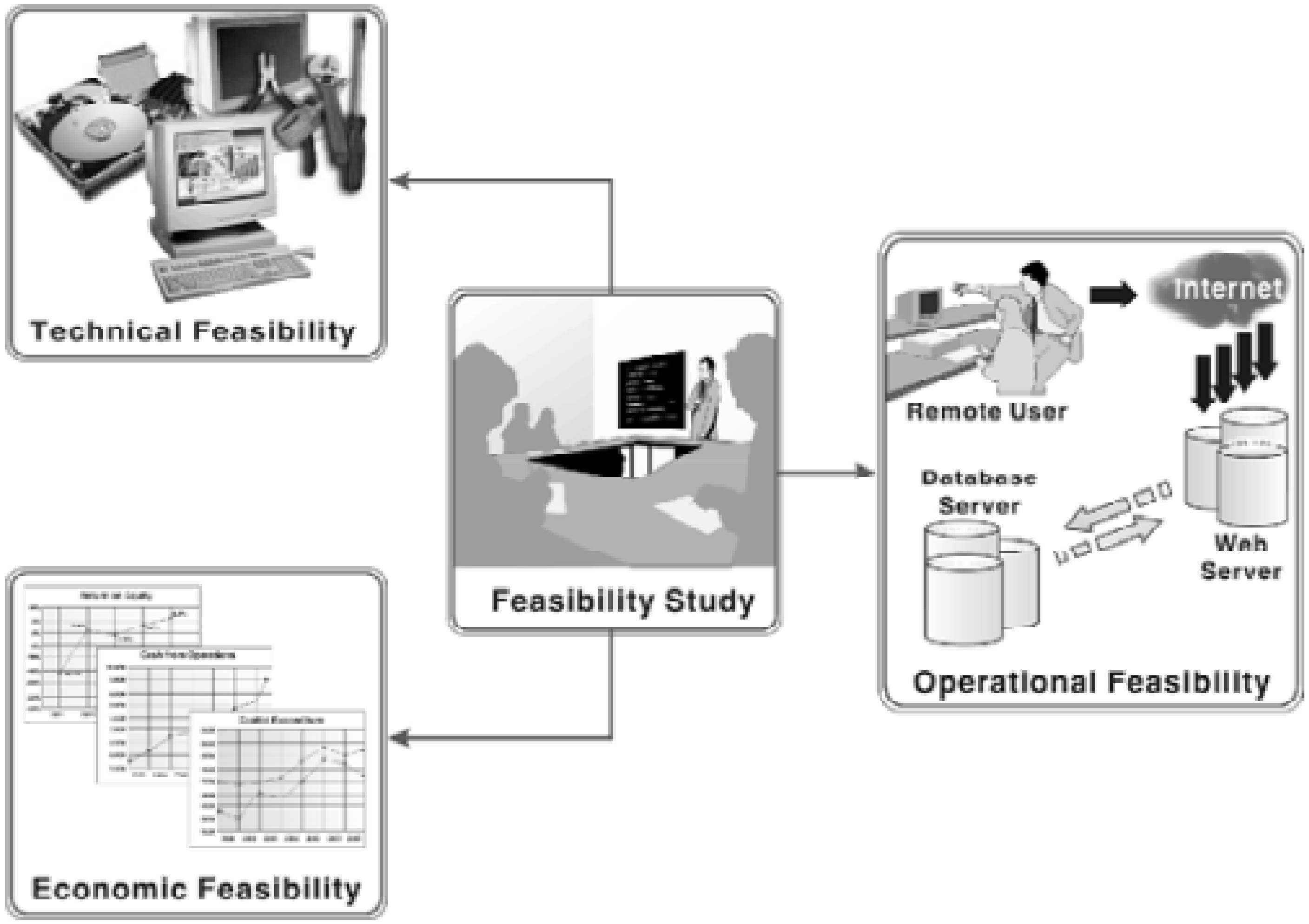
Objectives of feasibility study:

- To analyze whether the software will meet organizational requirements or not.

- To determine whether the software can be implemented using current technology and within the specified budget and schedule or not.

- To determine whether the software can be integrated with other existing software or not.
- To establish the reasons for developing software that is acceptable to users, adaptable to change, and conformable to established standards.
- To minimizes project failure.

Types of Feasibility



(a) Technical Feasibility:

- Analyzes the technical skills and capabilities of software development team members.

- Determines whether the relevant technology is stable and established or not.

- Ascertains that the technology chosen for software development has large number of users so that they can be consulted when problems arise or improvements are required

(*b*) ***Operational Feasibility*:**

- Determines whether the problems proposed in user requirements are of high priority or not.

- Determines whether the solution suggested by software development team is acceptable or not.

- Analyzes whether users will adapt to new software or not.

- Determines whether the organization is satisfied by the alternative solutions proposed by software development team or not.

(c)Economic Feasibility:

Economic feasibility determines: whether the required software is capable of generating financial gains for an organization or not. It involves the cost incurred on software development team, estimated cost of hardware and software, cost of performing feasibility study.

Feasibility Study Process

1. **Information assessment:**

- Identifies information about whether the system helps in achieving the objectives of the organization.

- verifies that the system can be implemented using new technology and within the budget.

- It also verifies whether the system can be integrated with the existing system

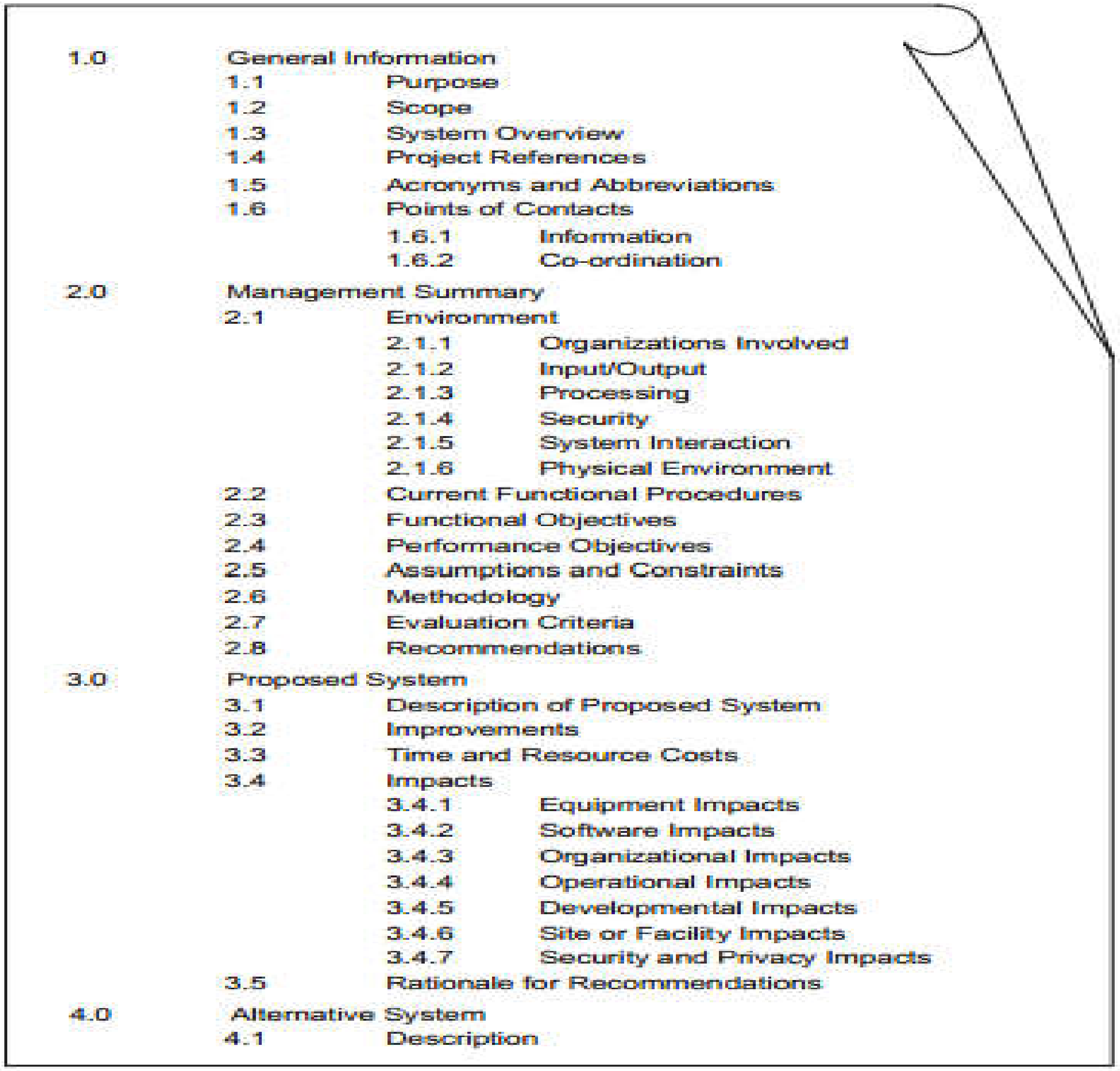
2. **Information collection:**

- Specifies the sources from where information about software can be obtained.

- Generally, these sources include users (who will operate the software), organization (where the software will be used), and software development team (who understands user requirements and knows how to fulfill them in software).

3. **Report writing**:

- Uses a feasibility report, which is the conclusion of the feasibility by the software development team. It includes the recommendation of whether the software development should continue or not. This report may also include information about changes in software scope, budget, schedule, and suggestion of any requirements in the system.



1.0 General Information
- 1.1 Purpose
- 1.2 Scope
- 1.3 System Overview
- 1.4 Project References
- 1.5 Acronyms and Abbreviations
- 1.6 Points of Contacts
  - 1.6.1 Information
  - 1.6.2 Co-ordination

2.0 Management Summary
- 2.1 Environment
  - 2.1.1 Organizations Involved
  - 2.1.2 Input/Output
  - 2.1.3 Processing
  - 2.1.4 Security
  - 2.1.5 System Interaction
  - 2.1.6 Physical Environment
- 2.2 Current Functional Procedures
- 2.3 Functional Objectives
- 2.4 Performance Objectives
- 2.5 Assumptions and Constraints
- 2.6 Methodology
- 2.7 Evaluation Criteria
- 2.8 Recommendations

3.0 Proposed System
- 3.1 Description of Proposed System
- 3.2 Improvements
- 3.3 Time and Resource Costs
- 3.4 Impacts
  - 3.4.1 Equipment Impacts
  - 3.4.2 Software Impacts
  - 3.4.3 Organizational Impacts
  - 3.4.4 Operational Impacts
  - 3.4.5 Developmental Impacts
  - 3.4.6 Site or Facility Impacts
  - 3.4.7 Security and Privacy Impacts
- 3.5 Rationale for Recommendations

4.0 Alternative System
- 4.1 Description

Fig: feasibility study plan

STEP2:REQUIREMENTS ELICITATION

Requirements elicitation (also known requirements capture or requirements acquisition) is a process of collecting information about software requirements from different individuals, such as users and other stakeholders.

Various issues may arise during requirements elicitation and may cause difficulty in understanding the software requirements.

Some of the problems are listed below:

- **Problems of scope**: This problem arises when the boundary of software (that is, scope) is not defined properly. Due to this, it becomes **difficult to identify objectives** as well as functions and features to be accomplished in software.
- **Problems of understanding:** This problem arises **when users are not certain about their requirements and thus are unable to express what they require in software** and which requirements are feasible.

This problem also arises when users have no or little knowledge of the problem domain and are unable to understand the limitations of computing environment of software.

- **Problems of volatility:** This problem arises when requirements change over time.

Elicitation Techniques

The commonly followed elicitation techniques are listed below:

Interviews: These are conventional ways for eliciting requirements, which help software engineer, users, and software development team to understand the problem and suggest solution for the problem.

An effective interview should have characteristics listed below:

- Individuals involved in interviews should be able to accept new ideas. Also, they should focus on listening to the views of stakeholders related to requirements and avoid biased views.
- Interviews should be conducted in defined context to requirements rather than in general terms. For this, users should start with a question or **a** ***requirements proposal***.

Scenarios: These are descriptions of a sequence of events, which help to determine possible future outcome before the software is developed or implemented.

Generally, a scenario comprises of the information listed below:

- Description of what users expect when scenario starts.
- Description of how to handle the situation when software is not operating correctly.
- Description of the state of software when scenario ends

Prototypes: Prototypes help to clarify unclear requirements. Like scenarios, prototypes also help users to understand the information they need to provide to software development team.

Quality function deployment (QFD): This deployment translates user requirements into technical requirements for the software. QFD identifies some of the common user requirements, which are listed below: **General requirement,Expected requirement Unexpected_requirement**

STEP3: REQUIREMENT ANALYSIS

- IEEE defines requirements analysis as

"(1) the process of studying user needs to arrive at a definition of a system, hardware, or software requirements. (2) the process of studying and refining system, hardware, or software requirements".

Various tasks performed using requirements analysis are listed below:

- Detect and resolve conflicts that arise due to unclear and unstated requirements.
- Determine operational characteristics of software and how it interacts with its environment.
- Understand the problem for which software is to be developed.
- Develop analysis model to analyze the requirements in the software.

STEP4: REQUIREMENTS SPECIFICATION

Develop software requirement specification document (known as requirement document).

IEEE defines software requirement specification as *"a document that clearly and precisely describes each of the essential requirements (functions, performance, design constraints, and quality attributes) of the software and the external interfaces. Each requirement is defined in such a way that its achievement can be objectively verified by a prescribed method, for example, inspection, demonstration, analysis, or test."*

Characteristics of SRS are listed below:

Correct: SRS is correct when all user requirements are stated in the requirements document. The stated requirements should be according to the desired system. This implies that each requirement is examined to ensure that it (SRS) represents user requirements. Note that there is no specified tool or procedure to assure the correctness of SRS. Correctness ensures that all specified requirements are performed correctly.

Characteristics of SRS are listed below:

Unambiguous: SRS is unambiguous when every stated requirement has only one interpretation. This implies that each requirement is uniquely interpreted. In case there is a term used with multiple meanings, the requirements document should specify the meanings in the SRS so that it is clear and easy to understand.

Complete: SRS is complete when the requirements clearly define what the software is required to do. This includes all the requirements related to performance, design and functionality.

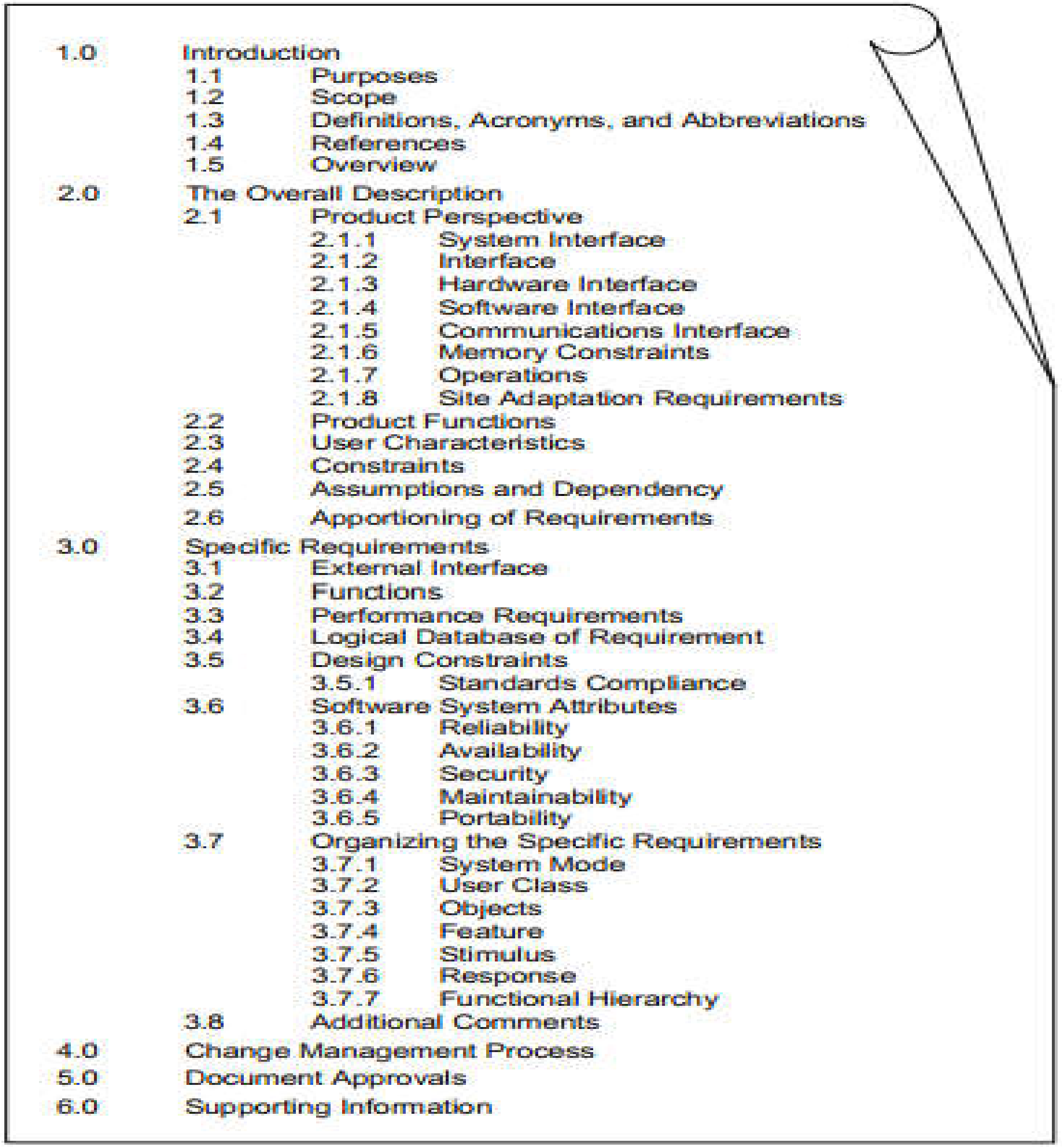
Modifiable: The requirements of the user can change, hence, requirements document should be created in such a manner where those changes can be modified easily, consistently maintaining the structure and style of the SRS.

Ranked for importance and stability: All requirements are not equally important, hence, each requirement is identified to make differences among other requirements. For this, it is essential to clearly identify each requirement. Stability implies the probability of changes in the requirement in future.

Verifiable: SRS is verifiable when the specified requirements can be verified with a cost-effective process to check whether the final software meets those requirements or not. unambiguity is essential for verifiability.

Consistent: SRS is consistent when the subset of individual requirements defined does not conflict with each other. For example, there can be a case when different requirements can use different terms to refer to the same object. There can be logical or temporal conflicts between the specified requirements and some requirements whose logical or temporal characteristics are not satisfied. For instance, a requirement states that an event ‘a’ is to occur before another event ‘b’. But then another set of requirements states (directly or indirectly by transitivity) that event ‘b’ should occur before event ‘a’.

Traceable: SRS is traceable when the source of each requirement is clear and it facilitates the reference of each requirement in future. For this, forward tracing and backward tracing are used. Forward tracing implies that each requirement should be traceable to design and code elements. Backward tracing implies defining each requirement explicitly referencing its source.



1.0 Introduction
- 1.1 Purposes
- 1.2 Scope
- 1.3 Definitions, Acronyms, and Abbreviations
- 1.4 References
- 1.5 Overview

2.0 The Overall Description
- 2.1 Product Perspective
  - 2.1.1 System Interface
  - 2.1.2 Interface
  - 2.1.3 Hardware Interface
  - 2.1.4 Software Interface
  - 2.1.5 Communications Interface
  - 2.1.6 Memory Constraints
  - 2.1.7 Operations
  - 2.1.8 Site Adaptation Requirements
- 2.2 Product Functions
- 2.3 User Characteristics
- 2.4 Constraints
- 2.5 Assumptions and Dependency
- 2.6 Apportioning of Requirements

3.0 Specific Requirements
- 3.1 External Interface
- 3.2 Functions
- 3.3 Performance Requirements
- 3.4 Logical Database of Requirement
- 3.5 Design Constraints
  - 3.5.1 Standards Compliance
- 3.6 Software System Attributes
  - 3.6.1 Reliability
  - 3.6.2 Availability
  - 3.6.3 Security
  - 3.6.4 Maintainability
  - 3.6.5 Portability
- 3.7 Organizing the Specific Requirements
  - 3.7.1 System Mode
  - 3.7.2 User Class
  - 3.7.3 Objects
  - 3.7.4 Feature
  - 3.7.5 Stimulus
  - 3.7.6 Response
  - 3.7.7 Functional Hierarchy
- 3.8 Additional Comments

4.0 Change Management Process

5.0 Document Approvals

6.0 Supporting Information

Fig : SRS Document

!! Brain wash !!

Suppose we have 3 glasses and 10 coins. The problem is to place odd number of coins in each glass i.e. each glass should contain coins and the number of coins in each glass must be odd and total coins which will be used must be equal to 10.

STEP 5 : REQUIREMENTS VALIDATION

WHY VALIDATION ?

Errors present in the SRS will adversely affect the cost if they are detected later in the development process or when the software is delivered to the user. Hence, it is desirable to detect errors in the requirements before the design and development of the software begin. To check all the issues related to requirements, requirements validation is performed.

STEP 5 : REQUIREMENTS VALIDATION

WHY VALIDATION ?

Errors present in the SRS will adversely affect the cost if they are detected later in the development process or when the software is delivered to the user. Hence, it is desirable to detect errors in the requirements before the design and development of the software begin. To check all the issues related to requirements, requirements validation is performed.

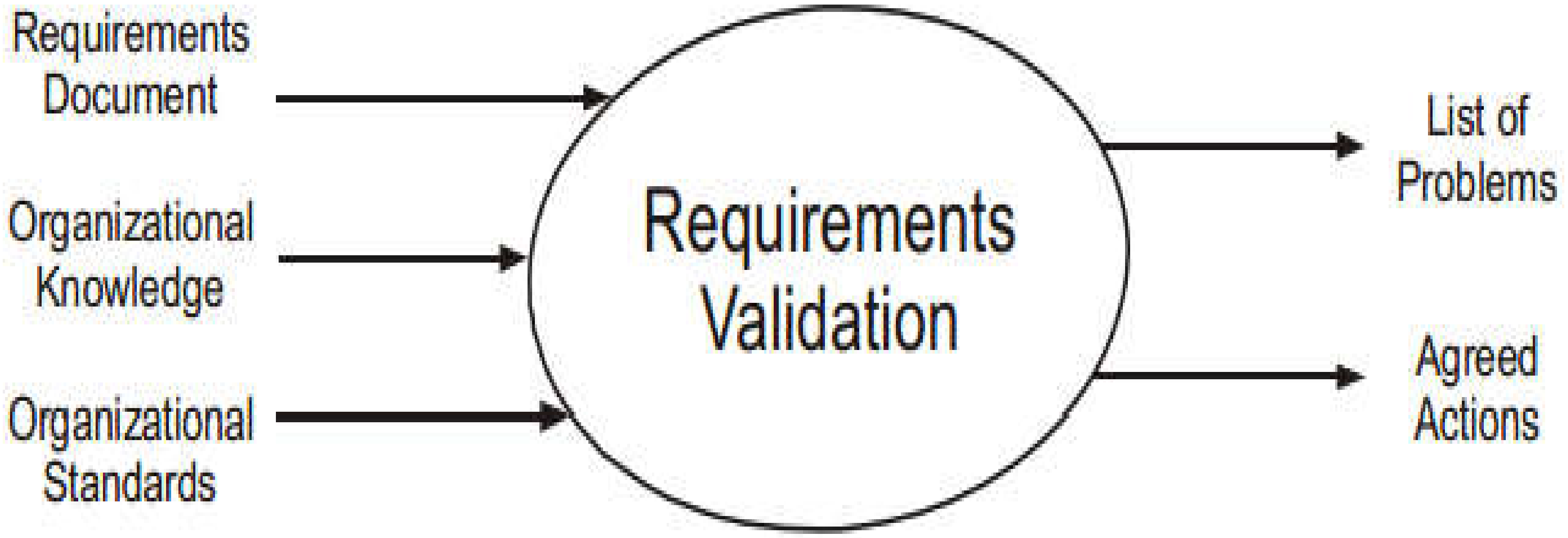


Fig: Requirement Validation

Various problems are encountered during requirements validation. These problems are

- Unclear stated requirements.

- Conflicting requirements are not detected during requirements analysis.

•Errors in the requirements elicitation and analysis.

- Lack of conformance to quality standards.

Requirement Validation Techniques

Requirement review:

The review team consists of software engineers, users, and other stakeholders who examine the specification to ensure that the problems associated with consistency, omissions and errors detected and corrected. In addition, the review team checks whether the work products produced during requirements phase conform to standards specified for the process, project and the product or not.

Test case generation: The requirements specified in the SRS document should be testable. The test in the validation process can reveal problems in the requirement. In some cases test becomes difficult to design, which implies requirement is difficult to implement and requires improvement.

Prototyping: This helps to interpret assumptions and provide an appropriate feedback about the requirements to the user. Ex: if users have approved a prototype, which consists of graphical user interface, then the user interface can be considered validated.

Automated consistency analysis: If the requirements are expressed in the form of structured or formal notation, then computer aided software engineering (CASE) tools can be used to check the consistency of the system. A requirements database is created using a CASE tool that checks the entire requirements in the database using rules of method or notation. The report of all inconsistencies is identified and managed.

STEP 6: REQUIREMENTS MANAGEMENT

WHY ??

It is difficult for the users to anticipate the effect of these new requirements (if a new system is developed for these requirements) on the organization. Thus, to understand and control changes to system requirements, requirements management is performed.

Advantages of requirements management:

Better control of complex projects: Provides the development team with a clear understanding of what, when and why software is to be delivered. The resources are allocated according to user-driven priorities.

Improves software quality: Ensures that the software performs according to requirements to enhance software quality. This can be achieved when the developers and testers have a concise understanding of what to develop and test.

Reduced project costs and delays: Minimizes errors early in the development cycle, as it is expensive to ‘fix’ errors at the later stages of the development cycle. As a result, the project costs also reduce.

Improved team communications: Facilitates early involvement of users to ensure that their needs are achieved.

Requirements Management Process

- Requirements management starts with planning,

- After planning, each requirement is assigned a unique ‘identifier’ so that it can be crosschecked by other requirements. Once requirements are identified, requirements tracing is performed.

- The objective of requirement tracing is to ensure that all the requirements are well understood and are included in test plans and test cases

- Traceability techniques facilitate the impact of analysis on changes of the project, which is under development.

- Traceability information is stored in a **traceability matrix**, which relates requirements to stakeholders or design module.

- Traceability matrix refers to a table that correlates requirements with the detailed requirements of the product.

Req. ID	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2
1.1		U	R					
1.2			U			R		U
1.3	R			R				
2.1			R		U			U
2.2								U
2.3		R		U				
3.1								R
3.2							R	

Fig: Traceability matrix

‘**U**’ in the row and column intersection indicates the dependencies of the requirement in the row on the column and ‘**R**’ in the row and column intersection indicates the existence of some other weaker relationship between the requirements.

Requirements change management

Requirements change management is used when there is a request or proposal for a change to the requirements. The advantage of this process is that the changes to the proposals are managed consistently and in a controlled manner.

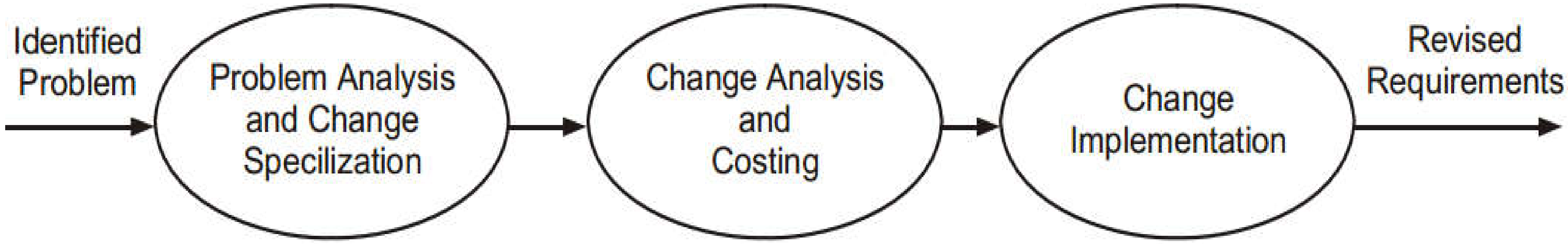

Identified Problem
Problem Analysis and Change Specilization
Change Analysis and Costing
Change Implementation
Revised Requirements

Fig: Required change management

These stages are listed below:

- **Problem analysis and change specification:** The entire process begins with identification of problems to the requirements. The problem or proposal is analyzed to verify whether the change is valid or not. The outcome of the analysis is provided to the 'change requester' and a more specific requirements change proposals is then made.

- **Change analysis and costing:**. The cost for this can be estimated on the basis of modification made to the design and implementation. After analysis is over, a decision is made as to whether changes are to be made or not.

- **Change implementation:** Finally, the changes are made to the requirements document, system design and implementation. The requirements document is organized in such a manner so that changes to it can be made without extensive rewriting.

Chapter 1: Finished (20 marks !!)

!! Attendance please !!

Thank You!!!