

UNIT-2

Part A(Multiple Choice Questions) Understanding

1. The goal of requirement phase is to produce____
 - A. To produce System design document
 - B. To produce high quality and stable SRS**
 - C. To produce Detailed design document
 - D. To produce review report

2. A requirement process involves ____phases
 - A. four
 - B. three**
 - C. five
 - D. two

3. The author of SRS document is____
 - A. developer
 - B. client
 - C. end user
 - D. analyst**

4. Expand SRS
 - A. Software requirement structure
 - B. Structured requirements software
 - C. Software requirement specification**
 - D. System requirement specification

5. An SRS establishes the basis for agreement between _____and_____on what the software product will do.
 - A. the Client and the Developer**
 - B. the Client and the Designer
 - C. the User and the Analyst
 - D. the Client and the Analyst

6. DFD are used in____phase of requirement process
 - A. requirement review
 - B. requirement validation
 - C. structured problem analysis**
 - D. informal problem analysis

7. _____represents Flow of data in a system
 - A. DFD**

- B. Flow chart
- C. Finite State Automata
- D. SRS

8. In a DFD, a process is represented by _____
- A. **circle(bubble)**
 - B. rectangle
 - C. source/sink
 - D. Oval

9. The system defined in multiple point of view is referred as
- A. state
 - B. function
 - C. Object
 - D. **Projection**

10. Net originator or Consumer of data in DFD is represented by _____
- A. circle(bubble)
 - B. parallel lines
 - C. **source/sink**
 - D. named arrows

11. _____ specifies the repeated occurrence of a regular expression
- A. Composition
 - B. **Closure**
 - C. Alteration
 - D. Atoms

12. An Evolutionary prototype leads to _____ model
- A. Prototype model
 - B. Waterfall model
 - C. **Iterative enhancement model**
 - D. Spiral model

13. A SRS is _____, if requirement state only one interpretation.
- A. consistent
 - B. complete
 - C. correct
 - D. **unambiguous**

14. if requirement stated doesn't conflict with another, then SRS is said to be ____
- A. **Consistent**
 - B. traceable
 - C. unambiguous
 - D. verifiable
15. Static and dynamic are the types observed in _____ requirement
- A. functional
 - B. **performance**
 - C. design constraint
 - D. external user interface
16. Omission directly effects the ____ of SRS
- A. Consistency
 - B. **Completeness**
 - C. traceability
 - D. modifiability
17. what is the error type ,if some requirements are not included in SRS
- A. Inconsistency
 - B. **Omission**
 - C. Incorrect Fact
 - D. Ambiguity
18. ____ requirement describe the relationship between the input and output of a system.
- A. Performance requirement
 - B. Design constraints
 - C. External interface requirement
 - D. **Functional requirement**
19. Informal approach, structured analysis and _____ are the approaches to problem analysis
- A. DFD
 - B. Finite state automata
 - C. **Prototyping**
 - D. Decision tables
20. Regular expressions, Finite state automata ,decision tables are part of
- A. Prototyping
 - B. **Specification language used for SRS**
 - C. Structured analysis
 - D. Requirement review
21. Which symbol represents A source / sink in DFD
- A. Circle

- B. **Rectangle**
 - C. Parallel lines
 - D. Named Arrows
22. Redundancy is a major issue in_____
- A. Consistency
 - B. Completeness
 - C. traceability
 - D. **modifiability**
23. Which of the following is true about External interface requirement
- A. It specifies the performance constraints on the software system
 - B. It specify which output should be produced from the given inputs
 - C. It specifies the requirements for the standards the system must follow
 - D. **It specifies all the details of hardware,software support and other requirement to be stated**
24. Which interface in SRS specifies the software communication with entities in the other machines
- A. Hardware interface
 - B. Software interface
 - C. **Communication interface**
 - D. User interface
25. The _____section in SRS document contains the purpose, scope, overview etc. of the requirements document
- A. **Introduction**
 - B. Specific requirements
 - C. Functional requirements
 - D. Performance requirements

Part B (4 marks) Understanding

1. Explain the need for SRS.

Client originates the requirements. The software is developed by software engineers and delivered to clients. Completed system will be used by the end-user. There are three major parties involved: client, developer and the end-user. The problem here is, the client usually does not understand software or the software development process and the developer often does not understand the client's problem and application area. This causes a communication gap between the client and the developer. A basic purpose of SRS is to bridge this communication gap. SRS is the medium with which the client and user needs are identified. Another important purpose of developing the SRS is helping the clients to understand their own needs. In order to satisfy the client, he has to be made aware about the requirements of his organization. The process of developing an SRS helps here.

2. **Explain the phases of the requirement process with a diagram.**

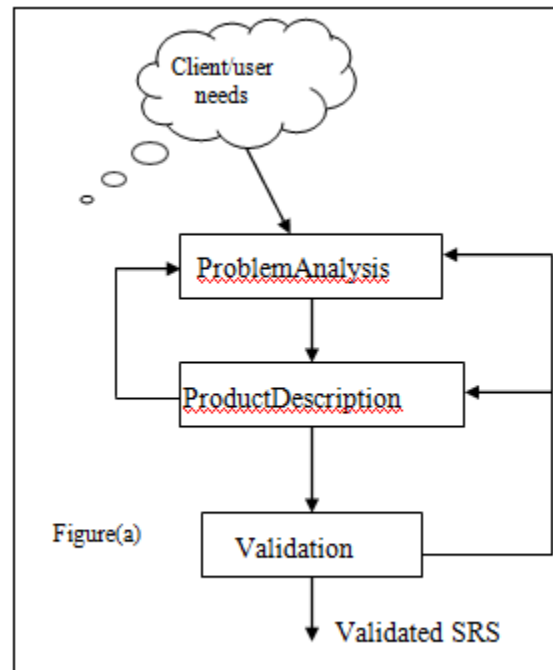
The requirement process is the sequence of activities that need to be performed in the requirement phase. There are three basic activities in case of requirement analysis. They are: 1. Problem analysis or requirement analysis. 2. Requirement specification. 3. Requirement validation. Problem analysis is initiated with some general statement of needs. Client is the originator of these needs. During analysis, the system behavior, constraints on the system, its inputs, and outputs are analyzed. The basic purpose of this activity is to obtain the thorough understanding of what the software needs to provide. The requirement specification clearly specifies the requirements in the form of a document. The final activity focuses on validation of the collected requirements. Requirement process terminates with the production of the validated SRS.

Though it seems that the requirement process is a linear sequence of these activities, in reality it is not so. The reality is, there will be a considerable overlap and feedback between these activities. So, some parts of the system are analyzed and then specified while the analysis of some other parts is going on. If validation activities reveal some problem, for a part of the system, analysis and specifications are conducted again.

The requirement process is represented diagrammatically in figure (a). As shown in the figure, from specification activity we may go back to the analysis activity. This happens because

the process specification is not possible without a clear understanding of the requirements. Once the specification is complete, it goes through the validation activity. This activity may reveal problems in the specification itself, which requires going back to the specification step, which in turn may reveal shortcomings in the understanding of the problem, which requires going back to the analysis activity.

During requirement analysis, the focus is on understanding the system and its requirements. For complex systems, this is the most difficult task. Hence the concept “divide-and-conquer” i.e., decomposing the problem into sub-problems and then understanding the parts and their relationship.



3. Explain briefly the structured analysis technique.

The structured analysis technique uses function-based decomposition while modeling the problem. It focuses on the functions performed in the problem domain and the data consumed and produced by these functions. This method helps the analyst decide what type of information to obtain at different points in analysis, and it helps to organize information.

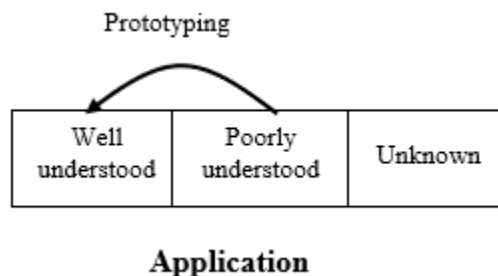
Data Flow Diagrams and Data Dictionary Data flow diagrams (DFD) are commonly used during problem analysis. DFDs are quite general and are not limited to problem analysis. They were in use before software engineering discipline began. DFDs are very useful in understanding a system can be effectively used during analysis. DFD shows the flow of data through the system. It views a system as a function that transforms the input into desired output. Any complex system will not perform this in a single step and the data will typically undergo a series of transformations before it becomes an output. The DFD aims to capture the transformations that take place within a system to the input data so that eventually the output data is produced. The agent that performs the transformation of data from one state to another is called a process and is represented in the form of a circle (or bubble) in the DFD. The processes are shown by named circles and data flows are represented by named arrows entering or leaving the bubbles. Rectangles represent a source or sink and is a net originator or consumer of data. An example of DFD is given in figure given below. This diagram represents the basic operations that are taking place while calculating the pay of employees in an organization. The source and sink both are worker here. Some conventions used in DFDs are: a labeled arrow represents an input or output. The need for multiple data flows by a process is represented by “*” between the data flows. This symbol represents AND relationship. For example, if “*” is there between two inputs A and B for a process, it means that A and B are needed for the process. Similarly the “OR” relationship is represented by a “+” between the data flows. It should be pointed out that a DFD is not a flowchart. A DFD represents the flow of data, while a flow chart shows the flow of control. A DFD does not include procedural information. So while drawing a DFD, one must not get involved in procedural details and procedural thinking is

consciously avoided. For examples, consideration of loops and decisions must be avoided. There are no detailed procedures that can be used to draw a DFD for a given problem. Only some directions can be provided. For large systems, it is necessary to decompose the DFD to further levels of abstraction. In such cases, DFDs can be hierarchically arranged.

4. **Describe the prototyping technique and its types used for problem analysis.**

Prototyping is another method that can be used for problem analysis. It takes a very different approach to problem analysis as compared to structured analysis. In prototyping, a partial system is constructed, which is then used by the clients, developers and end users to gain a better understanding of the problem and the needs. There are two features to prototyping: throwaway and evolutionary. In the throwaway approach, the prototype is constructed with the idea that it will be discarded after the analysis is complete. In the evolutionary approach, the prototype is built with the idea that it will be used in the final system. Determining the missing requirements is an advantage of prototyping. In case of evolutionary prototyping, more formal techniques need to be applied since the prototype is retained in the final system.

Throwaway prototype leads to prototype model and the evolutionary prototype leads to iterative enhancement model. It is important to clearly understand when a prototype is to be used and when it is not to be used. The requirements can be divided into three sets — those that are well understood those that are poorly understood, and those that are unknown. In case of throwaway prototype, poorly understood ones that should be incorporated. Based on the experience with the prototype, these requirements then become well understood as shown in figure below.



5. **Explain the role of DFD and data dictionary. Also explain different symbols with purposes used in DFD.**

Data flow diagrams (DFD) are commonly used during problem analysis. DFDs are quite general and are not limited to problem analysis. They were in use before software engineering discipline began. DFDs are very useful in understanding a system can be effectively used during analysis.

DFD shows the flow of data through the system. It views a system as a function that transforms the input into desired output. Any complex system will not perform this in a single step and the

data will typically undergo a series of transformations before it becomes an output. The DFD aims to capture the transformations that take place within a system to the input data so that eventually the output data is produced. The agent that performs the transformation of data from one state to another is called a process and is represented in the form of a circle (or bubble) in the DFD. The processes are shown by named circles and data flows are represented by named arrows entering or leaving the bubbles. Rectangles represent a source or sink and is a net originator or consumer of data. En example of DFD is given in figure given below. This diagram represents the basic operations that are taking place while calculating the pay of employees in an organization. The source and sink both are worker here. Some conventions used in DFDs are: a labeled arrow represents an input or output. The need for multiple data flows by a process is represent by “*” between the data flows. This symbol represents AND relationship. For example, if “*” is there between two inputs A and B for a process, it means that A and B are needed for the process. Similarly the “OR” relationship is represented by a “+” between the data flows. It should be pointed out that a DFD is not a flowchart. A DFD represents the flow of data, while a flow chart shows the flow of control. A DFD does not include procedural information. So while drawing a DFD, one must not get involved in procedural details and procedural thinking is consciously avoided. For examples, consideration of loops and decisions must be avoided. There are no detailed procedures that can be used to draw a DFD for a given problem. Only some directions can be provided. For large systems, it is necessary to decompose the DFD to further levels of abstraction. In such cases, DFDs can be hierarchically arranged.

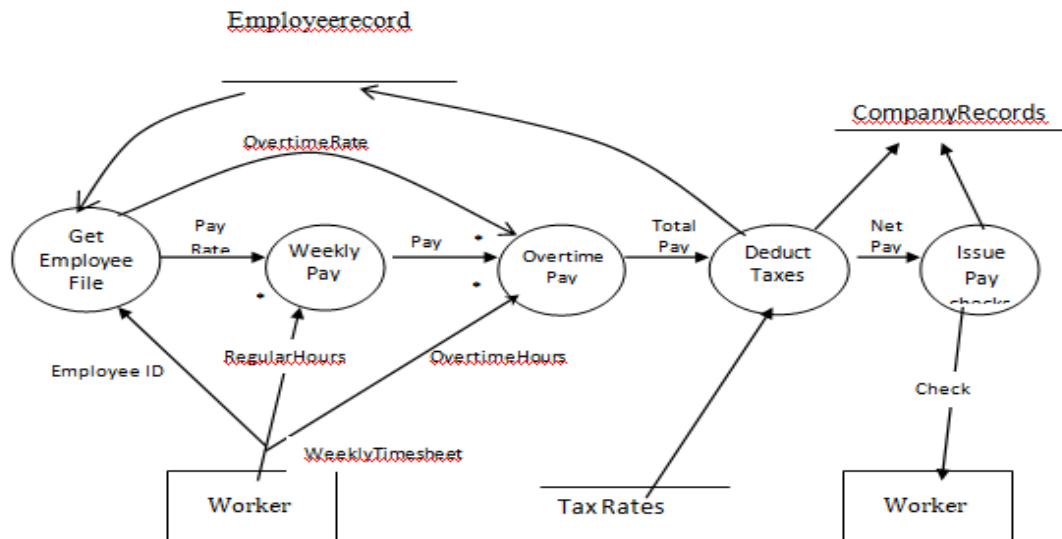


Figure: DFD of a system that pays workers

In a DFD, data flows are identified by unique names. These names are chosen so that they convey some meaning about what the data is. However, the precise structure of the data flows is not specified in a DFD. The data dictionary is a repository of various data flows defined in a DFD. Data dictionary states the structure of each data flow in the DFD. To define data structure, different notations are used. A composition is represented by +, selection is represented by / (i.e., either or relationship), and repetition may be represented by *. Example of a data dictionary is given below:









Weekly timesheet= employee_name +
employee_id+[regular_hrs+Overtime_hrs]*

Pay_rate= [hourly_pay/daily_pay /weekly_pay]

Employee_name=Last_name+First_name+Middle_nameEm

mployee_id= digit+ digit+ digit + digit

The symbols used in DFD are

Notation	De Marco & Yourdon	Gane and Sarson
External Entity		
Process		
Data Store		
Data Flow		

6. **Explain the characteristics of SRS.**

A SRS is correct if every requirement included in SRS represents something required in the final system. An SRS is complete if everything the software is supposed to do and the responses of the software to all classes of input data are specified in the SRS. Completeness and correctness go hand-in-hand. An SRS is unambiguous if and only if every requirement stated one and only one interpretation. Requirements are often written in natural language, which are inherently ambiguous. If the requirements are specified using natural language, the SRS writer should ensure that there is no ambiguity. One way to avoid ambiguity is to use some formal requirement specification language. The major disadvantage of using formal languages is large effort is needed to write an SRS and increased difficulty in understanding formally stated requirements especially by clients.

A SRS is verifiable if and only if every stored requirement is verifiable. A requirement is verifiable if there exists some cost effective process that can check whether the final software meets that requirement. Un-ambiguity is essential for verifiability. Verification of requirements is often done through reviews.

A SRS is consistent if there is no requirement that conflicts with another. This can be explained with the help of an example: suppose that there is a requirement stating that process A occurs before process B. But another requirement states that process B starts before process A. This is the situation of inconsistency. Inconsistencies in SRS can be a reflection of some major problems. Generally, all the requirements for software need not be of equal importance. Some are critical. Others are important but not critical. An SRS is ranked for importance and/or stability if for each requirement the importance and the stability of the requirement are indicated. Stability of a requirement reflects the chances of it being changed. Writing SRS is an iterative process. An SRS is modifiable if its structure and style are such that any necessary change can be made easily while preserving completeness and consistency. Presence of redundancy is a major difficulty to modifiability as it can easily lead to errors. For example, assume that a requirement is stated in two places and that requirement later need to be changed. If only one occurrence of the requirement is modified, the resulting SRS will be inconsistent. An SRS is traceable if the origin of each requirement is clear and if it facilitates the referencing of each requirement in future development. Forward traceability means that each requirement should be

traceable to some design and code elements. Backward traceability requires that it is possible to trace the design and code element to the requirements they support.

7. **Explain various components of an SRS.**

The basic issues an SRS must address are:

1. Functional Requirements
2. Performance Requirements
3. Design constraints imposed on implementation
4. External interface requirements.

Functional Requirements

Functional requirements specify which output should be produced from the given inputs. They describe the relationship between the input and output of a system. All operations to be performed on the input data to obtain the output should be specified. This includes specifying the validity checks on the inputs and output data. Care must be taken not to specify any algorithm. An important part of the specification is, the system behavior in abnormal situations like invalid inputs or error during computation. The functional requirements must clearly state what the system should do if such situations occur. It should specify the behavior of the system for invalid inputs and invalid outputs. And also, the behavior of the system where the input is valid but normal operation cannot be performed should also be specified. An example of this situation is an airline reservation system, where the reservation cannot be made even for a valid passenger if the airplane is fully booked. In short, system behavior for all foreseen inputs and for all foreseen system states should be specified.

Performance Requirements

This part of the SRS specifies the performance constraints on the software system. There are two types of performance requirements—static and dynamic. Static requirements do not impose constraints on the execution characteristics of the system. These include requirements like number of terminals to be supported, the number of simultaneous operations to be supported etc. These are also called capacity requirements of the system. Dynamic requirements specify constraints on the execution behavior of the system. These typically include response time and throughput constraints on the system. All these requirements must be stated in measurable terms. Requirements like “response time must be good” are not desirable because they are not verifiable.

Design Constraints

There are a number of factors in the client’s environment that may restrict the choices of the designer. An SRS should identify and specify all such constraints.

Standard Compliance: This specifies the requirements for the standards the system must follow. The standards may include the report format and accounting procedures.

Hardware Limitations:

the software may have to operate on some existing or predetermined hardware, thus, imposing restrictions on the design. Hardware limitations can include the type of machines

to be used, operating systems available, languages supported and limits on primary and secondary storage.

Reliability and Fault Tolerance: These requirements can place major constraints on how the system is to be designed. Fault tolerance requirements make the system more complex. Requirements in the system behavior in face of certain kinds of faults are to be specified. Recovery requirements deal with the system behavior in case of failure.

Security: These requirements place restriction on the use of certain commands, control access to data, provide different kinds of access requirements for different people etc. External Interface Requirements All the possible interactions of the software with the people, hardware and other software should be clearly specified. User interface should be user friendly. To create user friendly interface one can use GUI tools

8. **Write the various factors considered in design constraints imposed on implementation.**

There are a number of factors in the client's environment that may restrict the choices of the designer. An SRS should identify and specify all such constraints. Standard Compliance: This specifies the requirements for the standards the system must follow. The standards may include the report format and accounting procedures. Hardware Limitations: the software may have to operate on some existing or predetermined hardware, thus, imposing restrictions on the design. Hardware limitations can include the type of machines to be used, operating systems available, languages supported and limits on primary and secondary storage.

9. **Explain the general structure of SRS.**

All the requirements for the system have to be included in a document that is clear and concise. For this, it is necessary to organize the requirements document as sections and subsections. There can be many ways to structure requirements documents. The general structure of an SRS is given below.

1. Introduction

1.1.1 Purpose

1.1.2 Scope

1.1.3 Definitions, Acronyms, and Abbreviations

1.1.4 References

1.2 Overview

2. Overall Description

2.1 Product Perspective

2.2 Product Functions

2.3 User Characteristics

2.4 General Constraints

2.5 Assumptions and Dependencies

3. Specific Requirements

3.1 External Interface Requirements

3.1.1 User Interfaces

3.1.2 Hardware Interfaces

3.1.3 Software Interfaces

3.1.4 Communication Interfaces

3.2 Functional Requirements

3.2.1 Mode 1

3.2.1.1 Functional Requirement 1.1

. .

. .

3.2.1.n Functional Requirement 1.n . .

3.2.m Mode m

3.2.m.1 Functional Requirement m.1

3.3 Performance Requirements

3.4 Design Constraints

3.5 Attributes

3.6 Other Requirements

The introduction section contains the purpose, scope, overview, etc. of the requirements document. It also contains the references cited in the document and any definitions that are used. Section 2 describes the general factors that affects the product and its requirements. Product perspective is essentially the relationship of the product to other products. Defining if the product is independent or is a part of a larger product. A general abstract description of the functions to be performed by the product is given. Schematic diagrams showing a general view of different functions and their relationship with each other. Similarly, characteristics of the eventual end user and general constraints are also specified.

The specific requirements section describes all the details that the software developer needs to know for designing and developing the system. This is the largest and most important part of the documents. One method to organize the specific requirements is to first specify the external interfaces, followed by functional requirements, performance requirements, design constraints and system attributes.

The external interface requirements section specifies all the interfaces of the software: to people, other software, hardware, and other systems. User interfaces are clearly a very important component;

they specify each human interface the system plans to have, including screen formats, contents of menus, and command structure. In hardware interfaces, the logical characteristics of each interface between the software and hardware on which the software can run are specified. In software interfaces, all other software that is needed for this software to run is specified, along with the interfaces. Communication interfaces need to be specified if the software communicates with other entities in other machines. In the functional requirements section, the functional capabilities of the system are described. For each functional requirement, the required inputs, desired outputs, and processing requirements will have to be specified.

The performance section should specify both static and dynamic performance requirements. The attributes section specifies some of the overall attributes that the system should have. Any requirement not covered under these is listed under other requirements. Design constraints specify all the constraints imposed on design

