UNIT-3

Requirement Analysis

: Introduction

Requirement Analysis is the activity that consists of clear and through understanding of the product to be developed with all ambiguities and inconsistencies removed from customer perception of the problem. It is very difficult to find a clear and unambiguous understanding of the problem in absence of working model of computerization of an existing manual activity of an institution, the system Analyst can easily study and understand the input data, output data, the formats for both input/output and existing procedure for an activity. On the contrary, if the project is to be developed a fresh for which no working model exists, then the system Analyst job becomes more difficult as the task of gathering requirements becomes complex.

The System Analyst must aquaint himself with:

(i) Problem Identification
(ii) Reasons for solving the problem
(iii) Possible solutions of the problem
(iv) Data inputs to the system and data outputs from the system.
(v) Complexity of the problem / system.

fter performing the above information regarding the problem is called by performing the following activities:

1.1 Collection of Requirements

This activity consists of studying the perception of end user and customer eg if existing procedure is to be automized then system Analyst job becomes easier as input and output data formats and operational procedure details can easily be obtained from existing activity.

1.2 Analysis of Gathered Requirements

This activity involves the understanding of exact requirements. An inconsistency arises when there is difference in different end users requirements.

2. Requirement engineering

Requirement engineering is understanding ~~a reasonable~~ what the customer wants, analyzing need, assessing feasibility, negotiating a reasonable solution, specifying the solution unambiguously, validating the specification and managing the requirements as they are transformed into an operational system. The requirements engineering process can be described in five distinct steps.

(i) Requirement Gathering
(ii) Requirement Analysis and Negotiation.
(iii) Requirement Specification
(iv) System Modeling
(v) Requirement Validation

2.1 Requirement Gathering / Fact Finding

It certainly seems simple enough - ask the customer the users and others what the objectives for the system or product are, what is to be accomplished, how the system or product fits

Requirement Analysis

Types of Requirements :- The requirement definition and specification document describe everything about how the system is to interact with its environment.

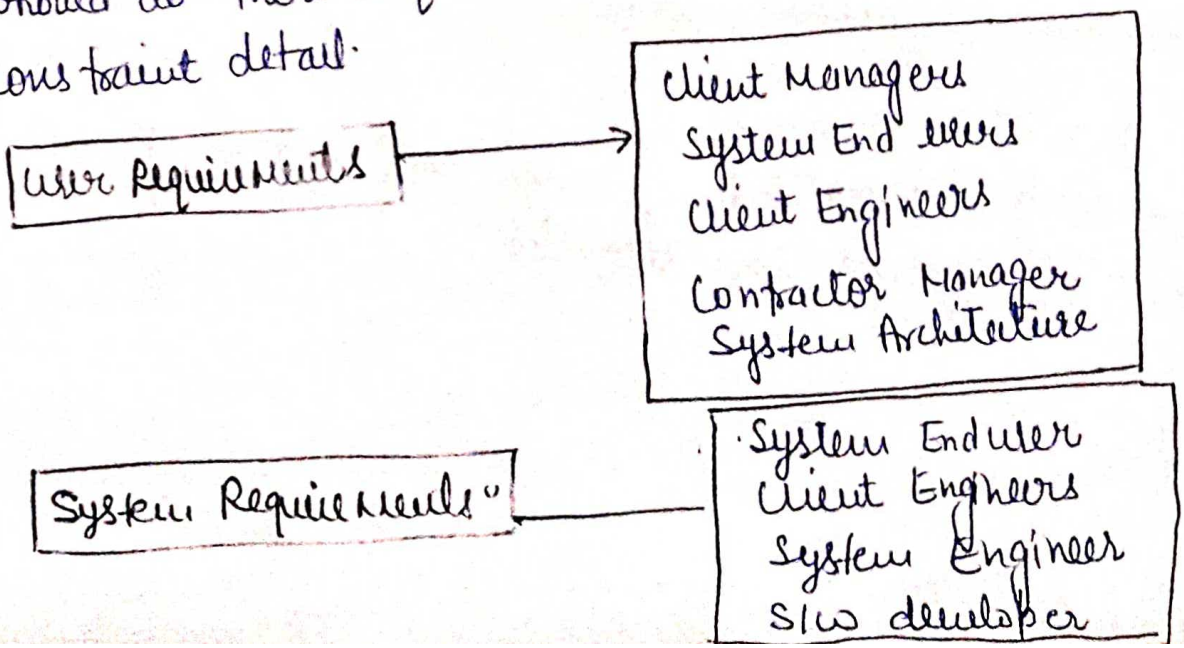
1) Physical Environment
2) Interfaces
3) User and Human Factors
4) functionality
5) Documentation
6) Data
7) Resources
8) Security
9) Quality Assurance

* Differentiating the User Requirements and System Requirements :-

* **User Requirement :-** User Requirement refers to the high level abstract requirements. These are the statement in a natural language plus diagram of what services the system is expected to provide and the constraints under which it must operate.

* **System Requirements :-**

System Requirement mean the details description of what the system (S/W) should do. These requirements set out the system services and constraint detail.



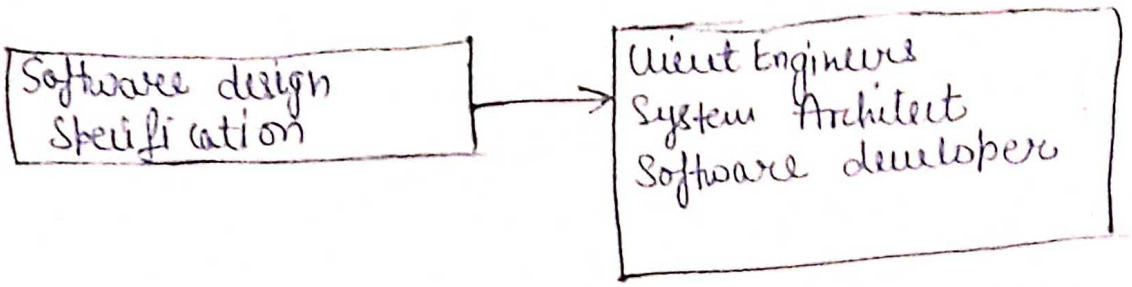


fig:- Readers of different types of Specification

Software Requirement Analysis :-

Software Requirement Analysis :-

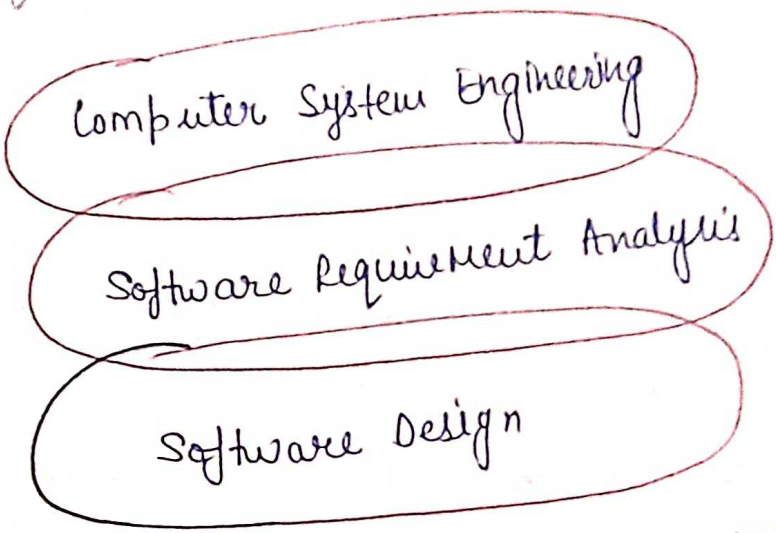


figure:- Analysis A bridge between computer System Engineering and Software Design.

→ Software Requirement Analysis is a Software Engineering task which bridges of the gap between computer System Engineering and Software design.

→ The Software Requirement Analysis activity is the next step of Software Project planning in the Software development.

into the needs of the business, and finally, how the system a product is to be used on a day-to-day basis.

Why because?

(i) Confusion of customer about the scope or the over all objectives of system.

(ii) Trouble of communicating the needs properly to software engineer.

(iii) Requirements are volatile i.e they keep on Changing.

There are certain guidelines to overcome these problems:

(i) Identify the people who will specify requirements

(ii) Assess the technical and other feasibilities.

(iii) Define the mode of gathering. e.g
- Interviews;
- team meeting

2.2 Requirements Analysis and Negotiation

Once requirements have been gathered, the work products noted earlier from the basis for requirements analysis. Analysis categorizes requirements and organizes them into related subsets; explores each requirement in relationship to others:-

For the requirements analysis activity, the followi questions are asked.

(1) Have all requirements been specified at the proper level of abstraction?

(2) Is each requirement consistent with the overall objective for the system.

(3) Is the requirement really necessary or does it represent an add-on features that may not be essential to the object

(4) Do any requirements conflict with other requirements?

(5) Is each requirement testable, once implemented

2.3 Requirement Specifications

A specification can be a written document, a graphical model, a formal prototype or any combination of these. For large systems, a written document, combining natural language description and graphical models represent its specification.

2.4 System Modeling

In order to fully specify what is to be built, we need a meaningful model by which it would be relatively easy, to assess the efficiency of work flow or effect. This determines how specification fit into a conceivable model and why?

2.5 Requirements Validation

The work products produced, as a consequence of requirements engineering, are assessed for quality during a validation step. Requirements validation examines the specification to ensure that all system requirements have been stated unambiguously.

3. SOFTWARE REQUIREMENT SPECIFICATION (SRS)

Requirement may be definedas:
A condition or capability that must be satisfied by a system to satisfy a contract, standard or specification.
Software are requirement phase deals with the requirements of the proposed system, The capabilities that the said system should posses.

4. SOFTWARE REQUIREMENT SPECIFICATION (SRS)

- SRS is a document that completely and precisely describes what the proposed software should do without describing ho

- SRS is a reference document that acts as a contract between developer and customer and is used to resolve any disagreement which may arise in future. Once the customer is agreed to the SRS document, the developer starts developing the product as per the requirements mentioned in SRS document.

4.1 Author of SRS Document

(i) SRS written by customer of the system.
(ii) SRS written by developer of the system.

4.2 Nature of SRS Document

SRS document should clearly and precisely documer the following:

(i) Functional requirements of the system.
(ii) Non-Functional requirements of the system.
(iii) Design constraints imposed on implementation of the system.
(iv) External interfaces.
(v) Performance requirements
(vi) Validation criteria
(vii) Other information pertinent to requirements.

4.2.1 Functional Requirement of the system

Every system performs a set of functions. Each function of the system can be considered as performing a transformation of a set of input data to the corresponding set of output da

4.2.2 Non-functional Requirements

Non-functional Requirements constitute the characteristics of the system that cannot b expressed functionally e.g maintainability, portability, security, operating and physical constraints. etc.

4.2.3 Design - Constraints Imposed on Implementation of the system.

Design - constraints on the functions of a system deals with what the system should or should not do. These include implementation, language policies for database integrity.

4.2.4 External Interfaces

It includes interaction of software with people/user, and with systems hardware.

4.2.5 Performance Requirement

Performance Requirement includes speed, availability, response time, recovery time of software.

4.2.6 Validation Criteria.

It is most important part of software requirement specification, which is most often neglected as it requires a complete understanding of softwdre. The successful implementation of a software depends on the effective and appropriate validation criteria.

5. CHARACTERISTICS OF GOOD SRS

1 Correctness

SRS is said to be correct if and only if every requirement stated in it is met by software. e.g if a software is supposed of perforing a particular task at a particular instant of time and it is responsible at some other instant of time then the requirement is incorrect.

2. Completeness

It constitutes the following elements:

(i) All important requirements relating to functionality performance design constraints and external interfaces.

software should do without describing h

(ii) Responses/output to both valid and invalid input values.

3. Unambiguousness :- SRS is said to be unambiguous if and only if every requirement stated in it has only one interpretation. Each written sentence in the SRS should have unique interpretation. SRS should unambiguous for both author and user, irrespective of their technical background. Requirements are generally written in natural language such as English.

The limitations of language is, the time required to learn the language which may not be understandable to the customers and users. Futhur these languages can only express certain types of requirements and can address certain types of systems.

4. Consistant :- SRS is consistent if, and only if individual requirements documented in it don't have any conflict.

Types of conflicts:

(i) The specified characteristics of objects may conflicts e.g the format of an output report may be tabular in one requirement and textual in another requirement.

(ii) There may be logical conflict between two specified actions.

(iii) Two or more requirements may describe the same object but the different terms for that object.

5. Modifiable :- SRS should be well structured and easily modifiable. SRS is said to be modifiable if, and only, if its structure is such that any changes to the requirements can be made easily completely and consistently without disturbing the structur

The requirements should not be redundant. Redundancy itself is not an error but it can easily lead to errors.

6. Verifiable :- SRS is verifiable, if and only if every requirement documented in it can be easily verified. For a SRS to be verifiable, it should be unambiguous. Non verifiable requirements include statements such as "work fine", "good human interface" and "generally happens".

7 Traceable :- SRS is said to be traceable if the origin of each of the requirement is clear and if facilities the referencing of each requirement in future.

(i) Backward Traceability :- It includes every requirement explicitly referencing its source in previous documents.

(ii) Forward Traceability : It includes each requirnt explicitly referencing its source in previous documents.

8. Maintainability, Portability and Adaptability :- SRS should record references to Maintainability, Portability and Adaptability. Maintainability is the collection of qualities which deals with how much easily the system can be changed. Portability ensures using a system under a new operating system or a new hardware plateform. Adaptability means the flexibility of the system to meet with predictable new requirements.

The advantages of SRS are :-

1. It works as a reference for the testing and validation of the final product.
2. It helps in getting high quality software as per requirements.
3. It acts as a guidelines for development.

software should do without reasoning role

4. It also reduces the development cost. 9

FORMAL SPECIFICATION TECHNIQUES

6. DATA FLOW DIAGRAMS (DFD'S)

The DFD (also known as pubble chart) is a graphical notation that represents a system in terms of input data to the system, processing carried out on these data and the output data generated by the system. DFD technique of representing flow of data through a system is popular and simple to use.

Symbols used in DFD:-

S.NO	Symbol	Name	Function
1.		Data flow	Used to connect processes to each other, to external entity
2.		Processor	Perfoms transformation of input data to output data.
3.		External Entity (Source or Sink)	Source of system inputs or sink of system outputs.
4.		Data Store	Place for storage of data, the arrow heads indicate net inputs and net output
5		Output	Represents data output

(i) Process:- Function is represented using a circle. symbol is also known as a process or bubble.

(ii) External Entity Symbol:- It is represented by a rectangle. The external entities are those physical entities which are external to the software system.

(iii) Data flow Symbol:- A directed arc or an arrow is used as a data flow symbol. It show the data flow between two processes or between an external entity and a process in the

The requirements should not be redundant.

10

direction of data flow arrow.

(iv) Data store symbal :- Open boxes are used to represent the data stores. A data store represents a logical file a data structure or a physical file on disk. Each data store is connected to a process by means of a data flow symbol.

(v.) Output symbal :- The box represents data output of the system.

6-1 Advantages of DFD

(i) DFD model is based on the hierarchical approch which is very easy to understand. Human mind can eadily understand this hierachical approch. This type of model stats with a very simple and abstract model of a system.

(ii) DFD technique constitutes simple set of rules and concepts.

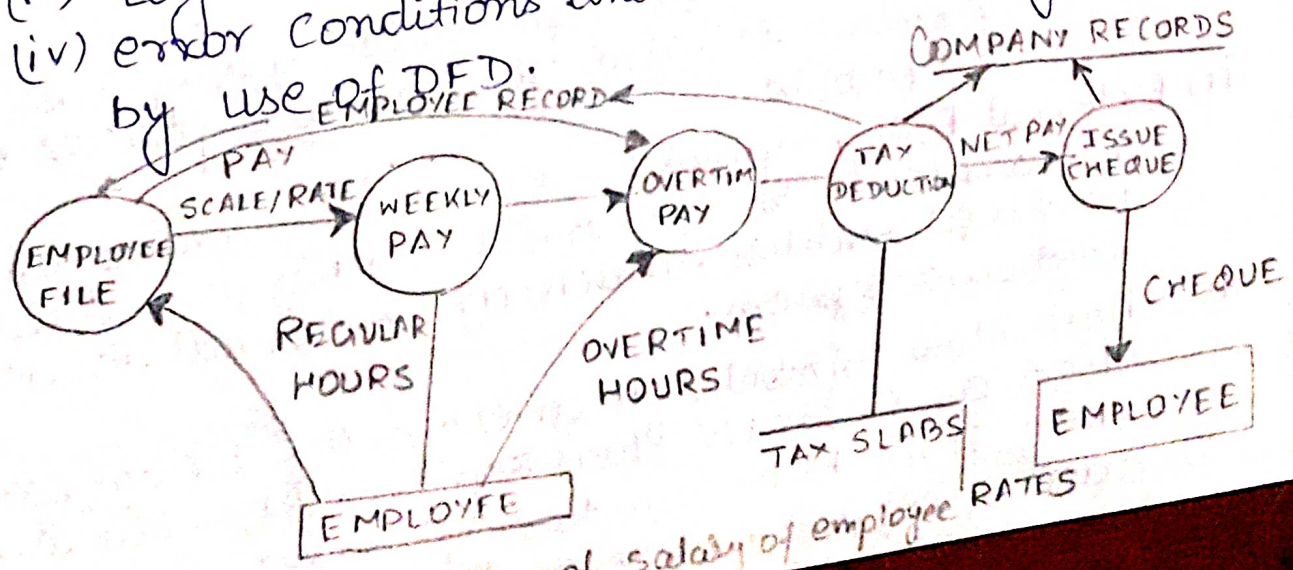
6-2. Checklist for DFD

(i) All names used should be unique which results in easy reference to iteams.

(ii) DFD is difficult from a flow chart. Arrows in flowchart represents the order of events but arrows in DFD represents data flow.

(iii) Logical decisions are suppressed in DFD.

(iv) error conditions and error handling are deferred by use of DFD.



DFD Representation of salary of employee

UNIT-3

7. DATA DICTIONARY

In a DFD, data flows are identified by unique names. These names specify the information regarding data. The data dictionary is an organized listing of all data elements that are important to the system will precise and clear definations so that both user and system analyst have a common understanding of inputs, outputs, components of stores and intermediate calculations. In other words it is data about data.

- The data dictionary is a repository of various data-objects data flows defined in a DFD.
- The data dictionary is used to provide an organized approach for representing the characteristics of each data object and control them.

7.1. Data Dictionary Format

- **Name:-** the primary name of data iteam/control iteam/data store/external entity.
- **Alias:-** other names by which data iteam/control item/data store is called for Annual Confidential Report DB for Database.
- **Description/purpose:** defines a list of processes that use the data or control item and how it is used.
- **Related data items:** ensures relationship between data items.
- **Range of values:** records all possible combination of values.
- **Content description:** a notation for representing content.
- **Supplementry:** information about types, present values, limitations etc.

Once a data object or control item name and its aliases are entered into data dictionary, consistency in naming can be enforced. System Analyst names a new data items as 'x' but 'x' is already in dictionary then the CASE (Computer Aided software engineering) tool associated with dictionary places a warning to indicate duplicate names.

When a dictionary entry is created, the CASE tool associated with dictionary scans the DFD's to determine which processes use the data or control information and how it is used.

The notation enables a software engineer to represent composite data in one of the following three ways:-

(i) As a sequence of data items
(ii) As a selection from among a set of data items.
(iii) As repeated grouping of data items.

Name = title + first_name + middle_name + last_name
title = [Mr. | Miss | Mrs. | Er. | Dr. | Prof.].
first name = {legal character}
middle name = {legal character}
last name = {legal character}
legal_character = [A-Z | a-z]

8. DATA MODELLING

Data modelling deals with a set of specific questions.

— What are the primary data objects to be processed by the system?
— Where do the objects exits?
— What are the relationship between each object and other objects?

DFD Representation of salary of employee RATES

UNIT-3

Software Prototyping And Specification

* **Software Requirements :-**

→ The requirements for as system is defined as what the system should do,
The service that its operation. These requirements refers needs of the customer in a system.

* Types of requirements :-

1 :- functional Requirements :-

→ These are statements of services the system should provide how the system should react to particular input and how the system should behave in particular situation.

2 :- Non-functional Requirements :-

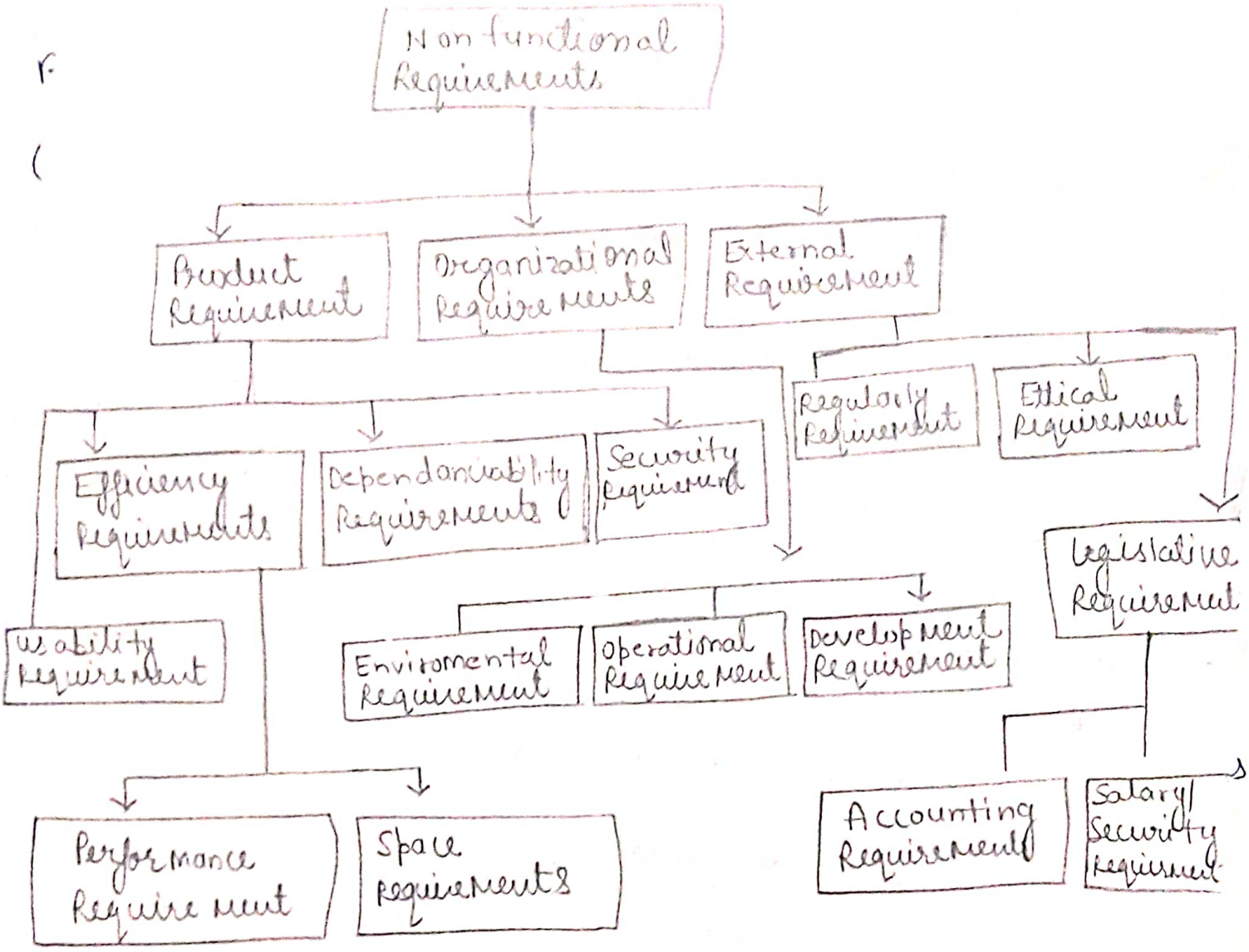
→ These are constraints on the services or functions offered by the system.

→ They include timing constraints, constraints on the develop - ments proces and constraints.

... control item name ... dictionary, ...

One

(2)



* Requirements Engineering Process :-

→ Requirement Engineering process may include four level activities

→ There focus on assessing if the system is useful to business (feasibility study), discovering requirement (elicitiation and analysis), converting these requirements into some standard form (Specification), and checking that the Requirement actually defines

. n . the following description :

UNIT-3

Finite State Machine

* Finite State Machine :-

→ Finite State Machine is used to recognized Patterns

→ Finite automata Machine takes the string of Symbol as input and changes its state acording.
In the input, when a desired Symbol is found then the transition occurs.

→ FA has two States : accept state or reject state. When the input string is successfully processed and the automata reached its final State then it will accept.

* A finite Automata consists of following :-

Q : finite Set of States

Σ : finite Set of Input Symbol

q_0 : ~~finite State~~ initial State

F : Final State

δ : Transition function.

The Transition function can be define as

$$\delta : Q \times \Sigma \rightarrow Q$$

* Finite Automata is characterized into two ways :-

1. DFA (Finite Automata) Deterministic
2. NDFA (Non Deterministic finite Automata)

* DFA (Deterministic Finite Automata) :-

→ DFA Stands For Deterministic Finite Automata. Deterministic refers to the uniqueness of the computation.

→ In DFA, the input character goes to one state only.

→ DFA doesn't accept the null value more that means the DFA cannot change state without any input character.

DFA has Five Tuples $\{Q, \Sigma, q0, F, \delta\}$

Q : Set of all States

Σ : Finite Set of Input Symbol where $\delta : Q \times \Sigma \rightarrow Q$

q0 : Initial State

F : final State

δ : Transition Function

Example :-

See an example of determinisics finite auto mata :-

Example 2 :- Draw a FSM Model for the following description.

$\Gamma = \{S, S_1\}$

$\Sigma = \{a\}$

S = Starting State

$\delta(S, a) = S_1$

The FSM model is shown

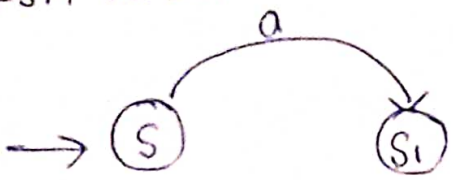


fig:- Transition among states

When an input does not result in a state change, we can indicate this in the diagram by showing an arrow originating and ending in the same state.

Example 3 :- Give the description of FSM

$T = \{S, A, B\}$

$\Sigma = \{0, 1\}$

S = Starting State

$\delta\{S, 0\} = A$

$\delta(S, 1) = S$

$\delta(A, 0) = A$

$\delta(A, 1) = S$

$\delta(B, 0) = S$

$\delta(B, 1) = B$

For the given data, loop concept is shown in

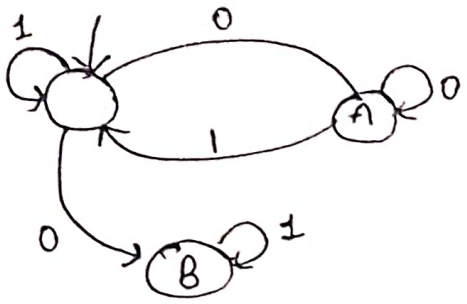


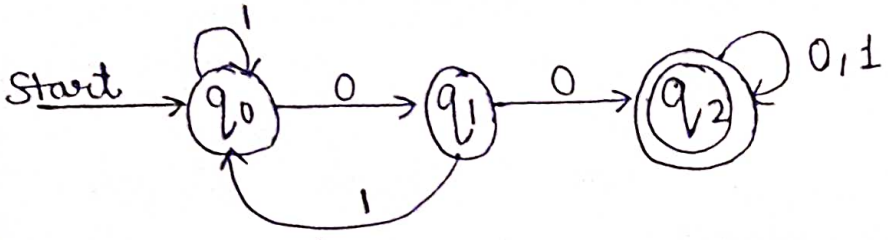
fig :- Loop concept

$Q = \{q_0, q_1, q_2\}$

$\Sigma = \{0, 1\}$

$q_0 = \{q_0\}$

$F = \{q_3\}$



* NDFA :-

→ NDFA refers to the Non Deterministic finite Automata. It is used to transit the any number of states for a particular input.

→ NDFA accepts the Null move that means it can change state without reaching the Symbols.

→ NDFA also has five states same as DFA, But NDFA has different transition function.

Transition Function of NDFA can be defined as :

$$\delta : Q \times \Sigma \rightarrow 2^Q$$

For

... Final result is shown in

Examples :-

(4)

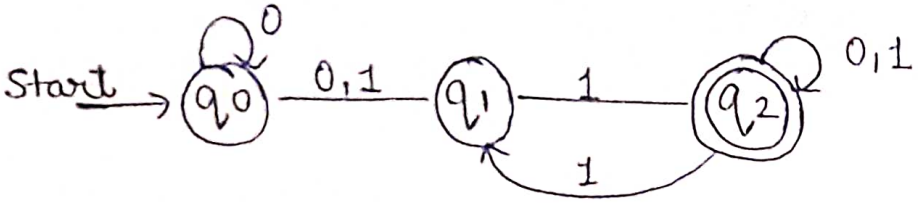
* See an example of non deterministics finite Automata:

$Q = \{q_0, q_1, q_2\}$

$\Sigma = \{0, 1\}$

$q_0 = \{q_0\}$

$F = \{q_3\}$



Examples :- A lamp can be either on or off and can go from on to off and as the consequence of an external action consisting of pushing the switch button, pushing the button again causes the opposite transition

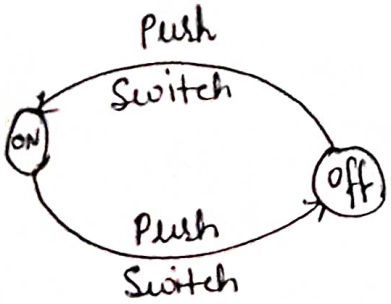


fig :- A Finite State Machine description of A Lamp Switche

UNIT-3

Structured Analysis

* Structured Analysis :-

→ A Software Engineering techniques that uses graphical diagrams to develop and portray System Specifications that are easily understood by users. These diagram describes the steps that need to occur and the data requirement to meet the design function of a particular Software.

* Systematic Approach :-

→ it is a Systematic Approach, which uses graphical tool that analyze and refine the objectives of an existing System and develop a new System Specification.

* Systmatic Attributes :-

→ it is logical rather than physical the elements of system do not depend on vendor or hardware.

→ It is an approach that works from high level overviews to lower level details.

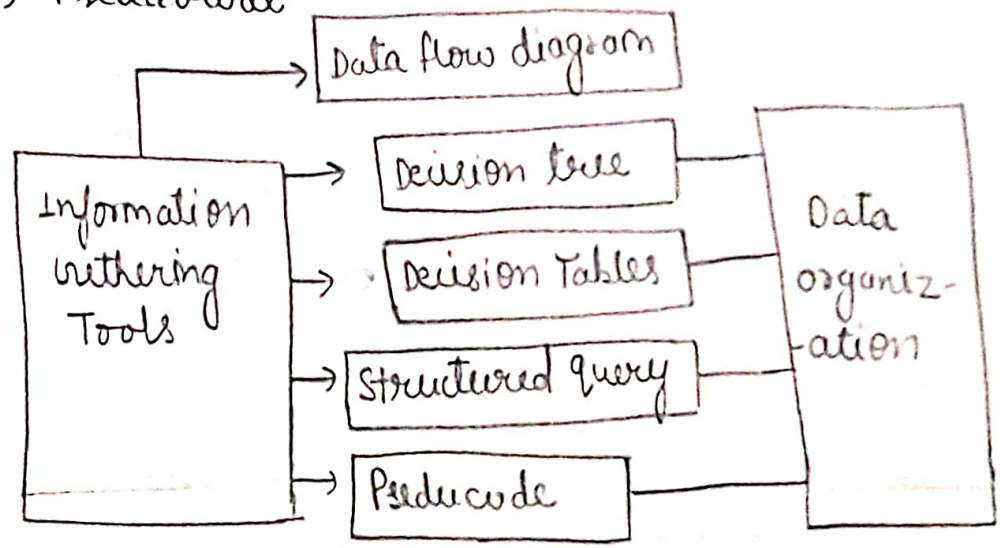
* Structured Analysis Tools :-

(2)

→ During Structured Analysis various tools and technology and Techniques are used for System development.

They are :-

1) Data Dictionary
2) Data flow diagrams
3) Decision Trees
4) Decision Tables
5) Structured English
6) Pseudocode



* Data Flow Diagram (DFD) :-

→ it is a Techniques developed by Larry constantine to express the requirements of System in a graphical form.

1) it shows the flow of data between various function of System and specifies how the current System Implemented.

Risk Analysis

Risk Analysis :-

→ A Software risk Analysis looks at code violation that pre sent a threat to the stability, security, or performance of the code.

→ Software risk is measured during testing by using code analyzers that can scan the code for both risks within the code itself and between units that must interact inside the application.

→ Risk is made up of two parts
1) the probability of something going wrong and
2) the negative consequence if it does.

*Project Risk :-

→ "Project Risk" is a problem that could some loss or threaten the Success Software project, but which has not happened.

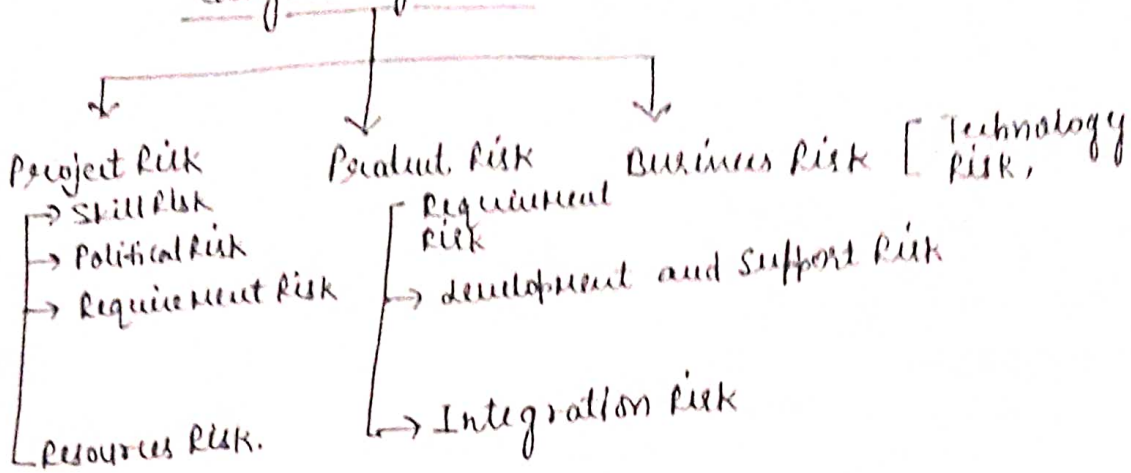
→ These problem can drastically impact on the schedule, cost or technical success of the Software project, the quality of Software product or project team morale.

* Categories of Risk :-

1) Project Risks
2) Product Risk
3) Business Risk

For

Categories of Risks



Categories of Risk :-

① **Project Risk :-** These are those risk that affect that affect the project schedule or resources.

Examples :- Loss of experience order.

② **Product Risks :-** This affect the quality or performance of the software being developed.

→

Example : Low performance.

③ **Business Risks :-** Risks that affect the organization. developing or loss of sw,

example :- Introduction of a new product by competitor.

Cohesion :-

→ Cohesion is the extension of the information hiding concept. Cohesion refers to the internal "glue" with which a Module is constructed. It is the concept that tries to captures the intra Module bond.

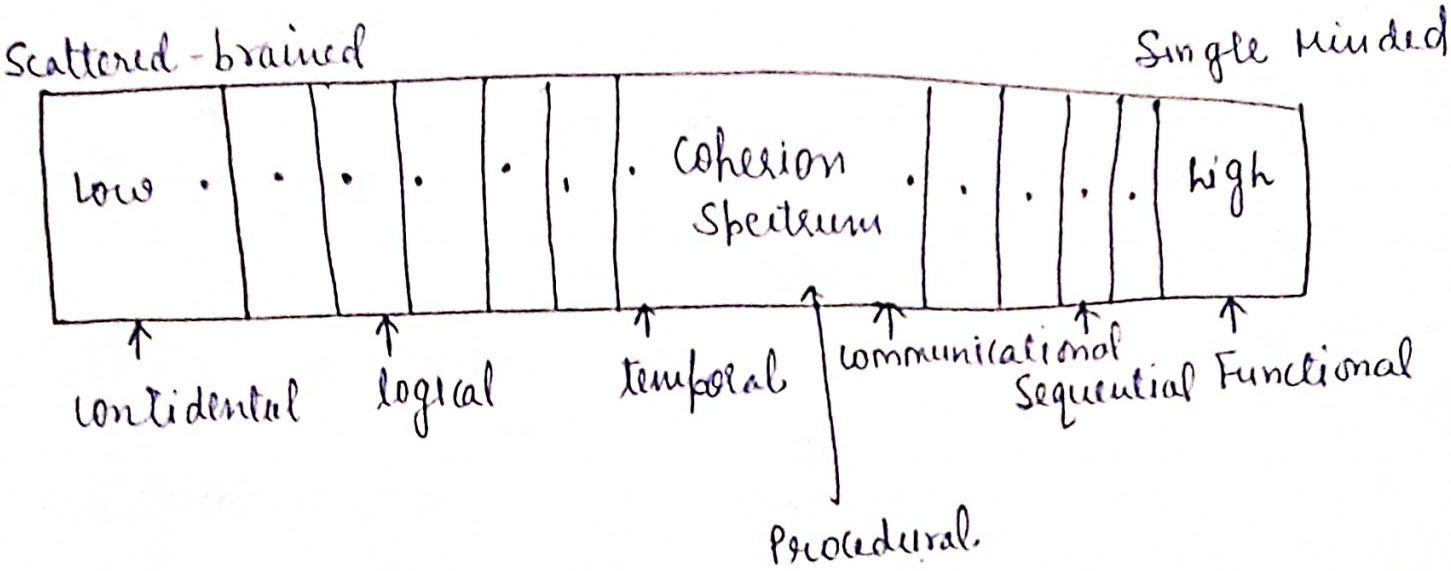


Fig:- Cohesion Scale : A Scale of the Relative Functional Strength of A Modelle.

It can be seen that there are total seven levels of Cohesion :-

1) Coincidental cohesion
2) Logical cohesion
3) Temporal Cohesion
4) Procedural cohesion
5) Communicational cohesion
6) Sequential Cohesion
7) Functional Cohesion

For

① **Coincidental cohesion :-**

→ The worst degree of cohesion is coincidental, which is placed at the lowest level.

→ it occurs when there is no meaningful relationship exist among the element of a module.

② **Logical cohesion :-** Logical cohesion is the next higher level of cohesion. where several logically related functions or data elements are placed in same component.

③ **Temporal cohesion :-** Temporal cohesion is same as logical cohesion, except that here the element are related with time and executed together. The time context is associated here.

④ **Procedural cohesion :-** When elements are grouped together in a module just to follow a sequential manner/order, the component is procedurally cohesive.

⑤ **Communicational cohesion :-** communicational cohesion has elements that are related by a reference to the same input or output data.

⑥ **Sequential cohesion :-** if the output from one part of a module has sequential cohesion. So, all the element are related in such a way in a module so that the output of one from the input to another.

⑦ **functional cohesion :-** Functional cohesion which is the strongest cohesion. All the elements of the module are related to perform a single wholle function.