

Software Crisis :->

Software Crisis can be defined as a set of problems raised during the software development periods.

The term software crisis was used to describes the impact of rapid increase in computer power and the complexity of the problems to be tackled. So the complexity, expectations and change hindered the software development process. This software crisis was growing rapidly.

Reasons responsible for software Crisis :->

- Problem of scale.
- Software are expensive.
- Software are late.
- Software are unreliable.
- inconsistent productivity.
- High optimistic estimates regard-

-ing Software development time
and Cost.

★ Software process :->

A process is a method of doing or producing something. Thus the software process is the way of developing or producing software.

The software process is :->

The set of Activities, methods and transformation that people use to develop and maintain software and associated products.
for example :-> product plans, design documents, code, test cases and user manuals.

Objectives of software process :->

A software process aims to achieve the following objectives:-

1. Effectiveness.
2. maintainability.
3. Quality.
4. predictability.
5. Improvement.
6. Follow up.

★ Key Characteristics of the software are :->

1. Software is not manufactured.
2. Software is developed or engineered.
3. Software does not wear out.
4. Most software continues to be custom built.
5. Software is flexible.

1. Software is not manufactured :->

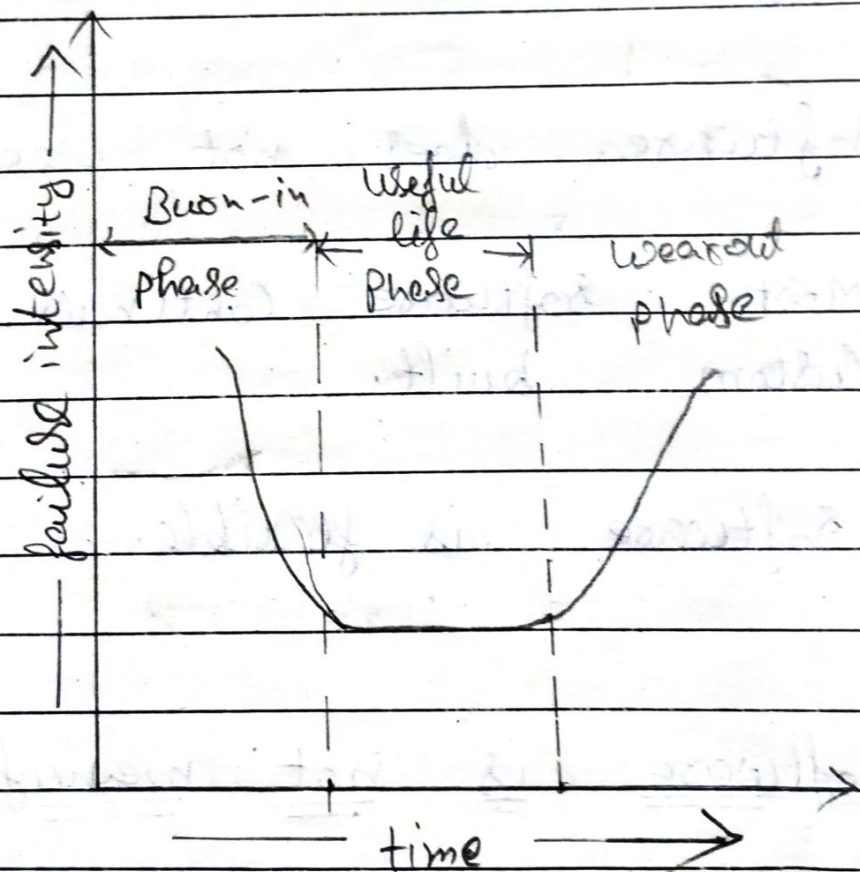
It is not manufactured

like physical products in a manufacturing plants.

2. Software is developed or engineered $\Rightarrow$

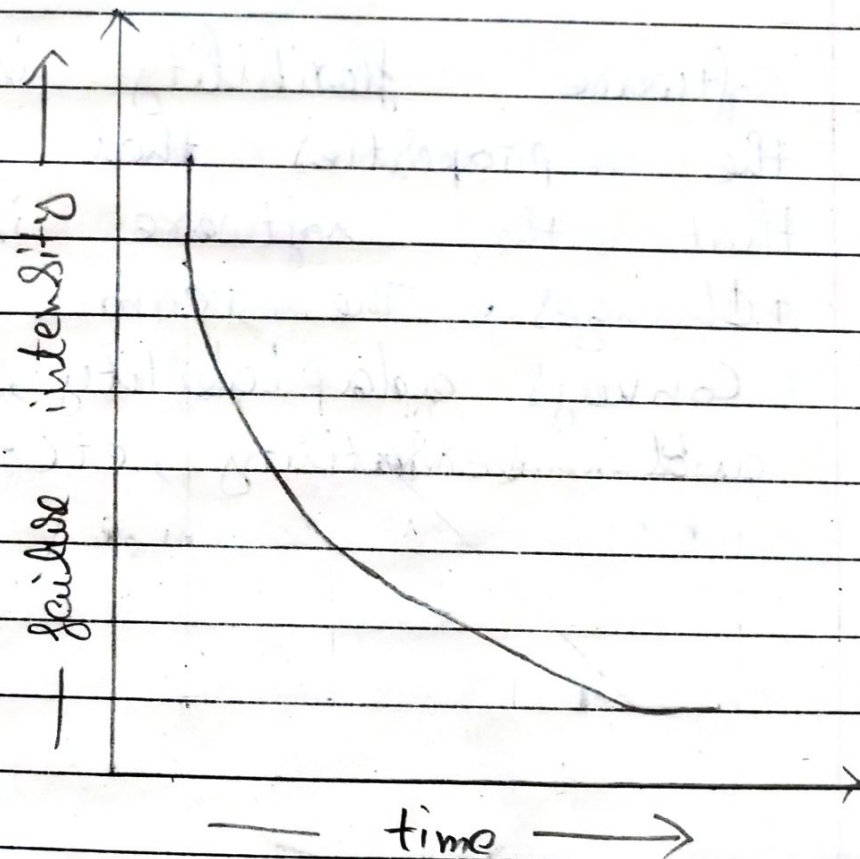
The software development begins from concept exploration. It is one time development efforts and continuous maintenance efforts to keep it operational. Software costs are concentrated in engineering.

3. Software does not wear out $\Rightarrow$



Software is not susceptible to the environmental malaises that cause hardware to wear out.

Software becomes reliable over time instead of wearing out. However software may retire if the environment, for which it was developed, changes. Software becomes obsolete due to new requirements, new expectations etc.



4. Most software continues to be custom built :->

Software components are built using a programming language that has a limited vocabulary. To understand the whole component for making some changes to it is very complex task because of complexity of code. So instead of this, custom component is created.

5. Software is flexible :->

Software flexibility is one of the properties that indicate that the software is easy to change. The term "flexibility" conveys adaptability, elasticity and versatility, etc.

* Software development life cycle →

The software development life cycle is that which is used to facilitate the development of a large software product in a systematic, well-defined and cost effective way.

problem description and
system study.

Feasibility study

Software requirement
analysis & specification

Software design and spe-
cification.

Coding and module testing

integration and system testing

System implementation

system maintenance.

Page
 Date

1. Problem description and system study :->

System development life cycle starts with problems description phase. Many important issues are clarified at this stage. These are

- Complete understanding of the problems.
- Areas to be automated.
- To identify the goals to be achieved.
- To estimate the various type of benefits in the new system over the current system.
- The scope of the project i.e. identifying what other areas are affected after automation.

2. Feasibility study :->

Depending upon

the results of the initial investigation a more detailed feasibility study is carried out.

various types of feasibility studies are $\Rightarrow$

(i) Technical feasibility $\Rightarrow$

In technical feasibility it is checked that

- Does the necessary technology exists?
- Can the development of the proposed system be done with current equipment?
- With the proposed system provide adequate responses to the enquiries?
- Can the system be expanded, if developed?

(ii) Economic feasibility $\Rightarrow$

It is to determine expected benefits and saving from a proposed system

and compare them with costs.

(iii) Social feasibility →

It is to determine whether a proposed project will be acceptable to the people or not.

(iv) Legal feasibility →

It is to determine whether a proposed project is under rules, acts, etc.

(v) Time feasibility →

It is to determine whether a proposed project can be implemented fully within a fixed time frame.

3. Software Requirement Analysis and specification →

Analysis is a detailed study

of the

- Various operations performed by the system.
- Relationship among the various sub-systems or functional unit and
- Finally the relationship outside the system.

It includes two distinct activities:

(i) Requirement Analysis:->

The goals of the requirement analysis is to collect & analyze all related data & information with a view to understand the customer requirements

(ii) Software Requirement Specification:->

The software requirement specification document contain:

- Properly organized & documented requirements of the users.
- Nature of its interface to users.
- Hardware requirements.

Software requirement specification document is used:

- To develop a solution by the software engineers.
- To analyse and confirm by the customer in order to verify whether it captures all of the customer's expectations.
- During system testing, the system is expected to be tested against its requirements.

4. Software design and specifications:->

This phase transform the requirements specification into a structure which is implemented into some programming language. There are two approaches:

(i) Traditional design approach:->

In this approach two different activities are performed :-

First is :

- Structured analysis of the requirements specification is carried out.
- Structured analysis is transformed into software design.
- Structured analysis involves preparing a detailed analysis of different functions to be carried out by the system

1 identification of the data flow among the different functions.

- Functional requirement specified in the SRS document are decomposed into subfunctions & data flow between these subfunctions is analysed.

Second is:

The architectural design defines:

- which components are required?
- The general structure of the software.
- The functions provided by each module.
- Interfaces between modules.
- Data structures and output format of the system.

The detailed design defines:-

- How each component can be implemented.
- Internal logic of each module with detailed data structure.

(ii) object - oriented Design:->

In object - oriented design various object that occur in the problem domain and different kinds of relationship that exist between those object are defined.

5. Coding and module testing

After the design phase comes the coding. Coding is translating the design structure into a programming language.

6. Integration and system testing:-

In this phase, all tested modules are combined according to the architectural design and testing as a whole system. It is also required to check whether the inter-connection between the modules is correct or not.

7. System implementation :->

implementation of system means installing the system on user's site. Various kinds of implementation are:

• Direct Conversion:-

new system directly implemented.

• Phased Conversion:-

modules are implemented separately.

• Parallel Conversion:-

new system is run in parallel with the old system.

8. System maintenance:- System maintenance is required after the system is delivered to the customer's site, installed and is operational. There are three kinds of maintenance:

i) Corrective maintenance:- correcting errors that were not discovered during the product development phase.

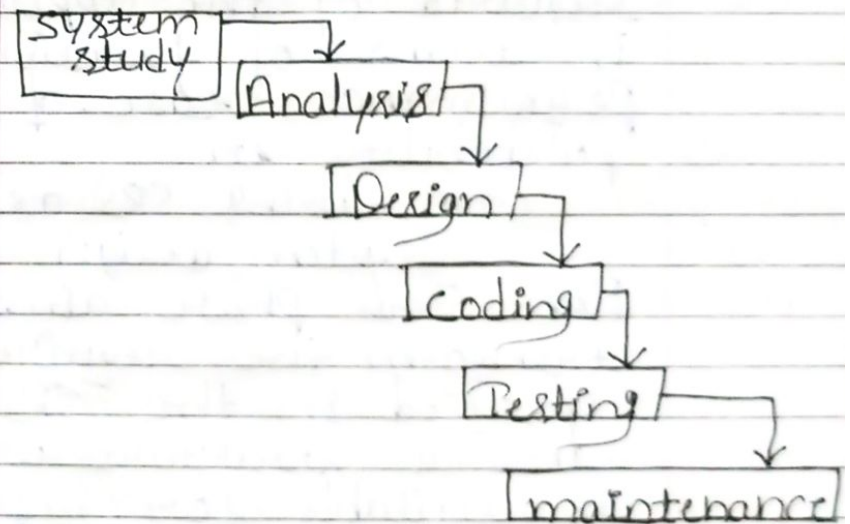
ii) Perfective maintenance:- improving the implementation of the system and increasing the functionalities of the system according to the customer's requirements.

iii) Adaptive maintenance:- porting the software to a new environment eg. to a new computer or to

to a new operating system. the main purpose of this phase is to preserve the software for a long time.

⇒ Waterfall MODEL:- Waterfall model is the simplest software process model. It is a traditional or the oldest model.

It is a systematic and sequential approach towards software development. It is also called sequential model.



i) System study:- first of all, it is necessary to know the problem. In this first step of waterfall

model; there is a meeting with the customer, to understand the requirements.

ii) Analysis:- This phase exactly tells the requirements and need of the project. The purpose of requirement analysis is to identify the quality required of the application in terms of functionality, performance, ease of use, portability etc.

iii) Design:- using SRS as input system design is done. This phase aims to transform the requirement specified in the SRS into a structure which is suitable for implementation in some programming language. During this phase the software architecture is defined and the high level and

Detailed Design work is performed.

iv) Coding:- After the design phase comes the coding. Coding is translating the design structure into a programming language.

v) Testing:- The testing phase is where the software is checked for any error or discrepancies.

vi) Maintenance:- Software maintenance include error correction, enhancement of capabilities, deletion of obsolete capabilities and optimization. maintenance is done to deliver these changes in the customer environment.

Advantage of waterfall model:-

⇒ Advantage of waterfall model are:-

- i) Every stage and activities are well defined.
- ii) Helps to plan and schedule the project.
- iii) Testing is inherent to every phase.
- iv) minimizes the cost of rectifying errors.
- v) It is simple and easy to explain to the user.

v) There is very little interaction b/w the user and software engineers.

Disadvantage of waterfall model

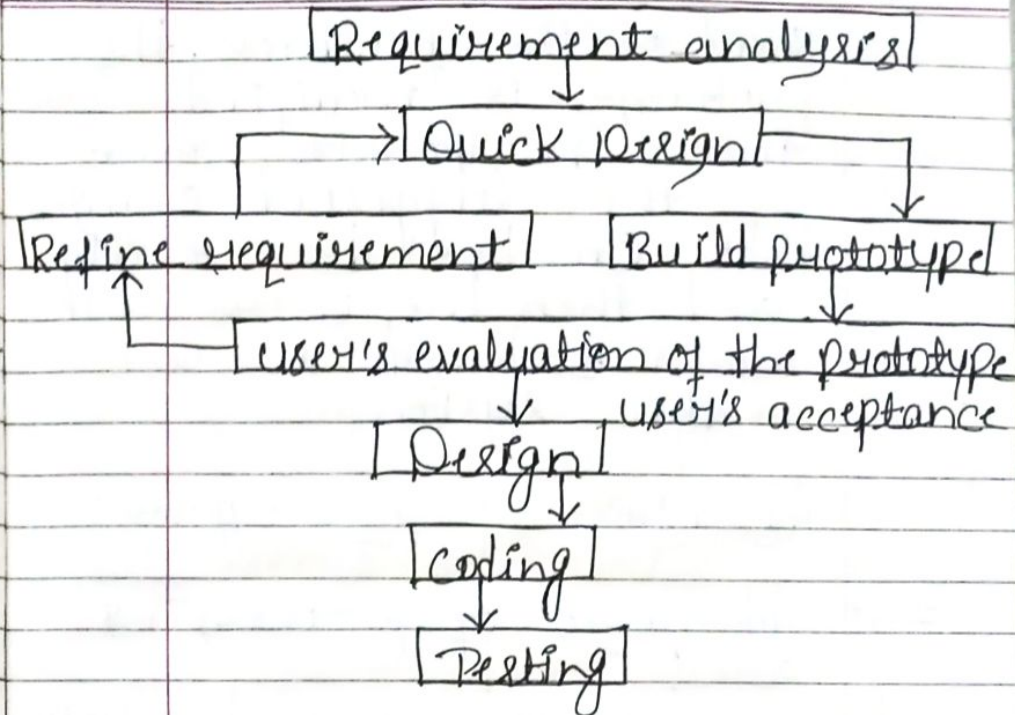
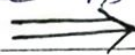
⇒ Disadvantage of waterfall model are:

- i) This model expects requirements early in the process, which is unrealistic.
- ii) it does not cover any kind of risk assessment.
- iii) user training is does not given enough importance.
- iv) Problems are not discovered until system testing.

⇒ PROTOTYPING:-

A prototype is a working system that is developed to test ideas about the new system. a prototyping is a process of building a model of the system to be developed.

The basic idea of prototyping is thus instead of fixing requirements before design and coding, a prototype is built to understand the requirements. on the basis of known requirements, the prototype is built.



After developing the prototype, the end user and given an opportunity to use the prototype. based on their experience, they provide feedback to the developers regarding the prototype like:

- * what is correct?
- * what needs to be modified?
- * what is missing?
- * what is not needed? etc.

Based on the feedback, the prototype is modified into incorporate some of the suggested changes that can be done easily and then the users are again followed to use the system.

Advantage of prototyping:-

⇒ Advantage of prototyping are:

- i) well suited where the user needs are not clearly definable / identifiable in advance
- ii) error can be detected much earlier.
- iii) modification of prototype is faster and cheaper than the modification of final software.
- iv) Reduces the risk associated with the project.
- v) Helps to determine the

flexibility of the system.

Disadvantage of prototyping:-

⇒ Disadvantage of prototyping are:

- i) Developer may ignore the quality of the system in order to get the prototype working quickly.
- ii) users on seeing a working prototype, often start expecting the actual system to be ready soon after.

Difference b/w waterfall model and prototype model:-

⇒ waterfall model	prototype model
i) In waterfall model a sort of linear & conventional approach is used	In prototype model is evolutionary approach is used.
ii) It requires specification of all the requirement in the beginning	It does not require the specification of all the requirement in the beginning

iii) Developers are not allowed to refine or add modify the requirement or specification of the previous phase until the next iteration

iv) It is implemented in the project where software has to be delivered to the client it is usually used in online project where client is not sure about his requirement.

v) Project cost almost the same as estimated more than what was estimated

vi) The complexity of an error increases because of the nature of the model each phase is sequential of the other the prototype enables the developer to detect any deficiency early at the process

vii) client can only preview the system only after the final version of the software is developed client can have preview of the system from the 'quick design' at an early phase of process

⇒ SPIRAL MODEL:-

This is a recent model proposed by Boehm in 1986. It combines the features of prototyping as well as waterfall model. "A model of the software development process in which the constituent activities, typical requirement analysis, preliminary and detailed design, coding, integration and testing are performed iteratively until the software is complete." is a spiral model.

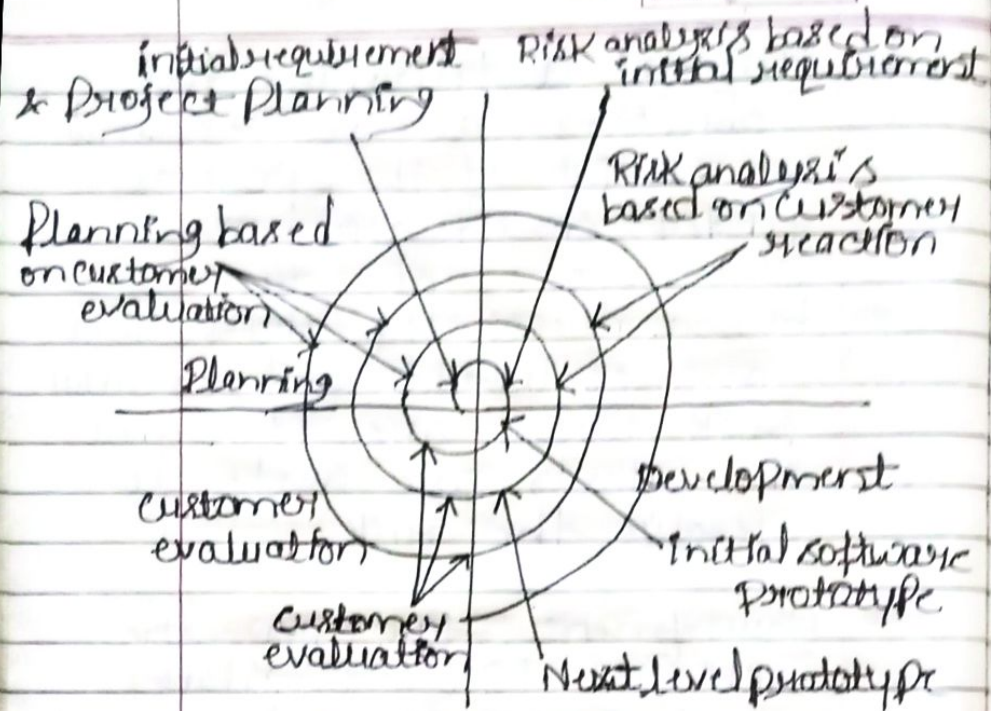
many software projects have failed due to ignoring the project risk. Boehm's this model deals with uncertainty risk factors which are inherent in software project risk are connected with the future. the spiral model is useful for identifying

★ Planning:- it determine the objectives, alternatives and constraint of the project under consideration.

★ Risk analysis:- The alternatives are evaluated and potential risk areas are identified the efforts are made to resolve the risk.

⇒ Development:- By using the activities like simulation and prototyping, the development and verification of the next level product is done.

→ Customer evaluation:- The customer evaluates the engineering work and makes suggestions and modification.



Advantage of spiral model:-

- ⇒ The various advantages of spiral model are:-
- i) most realistic model, which works for both development type as well as enhancement projects.
 - ii) it provide the potential for rapid development of incremental version of

- the software.
- iii) Less documentation is needed.

Disadvantage of spiral model:-

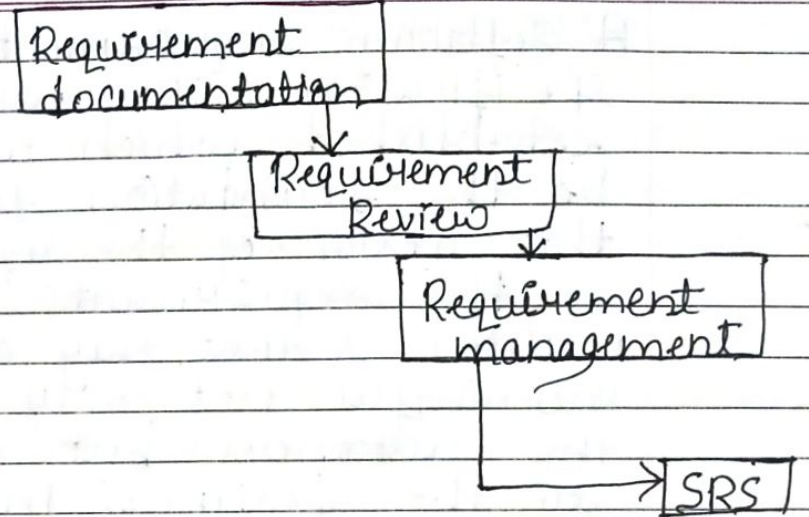
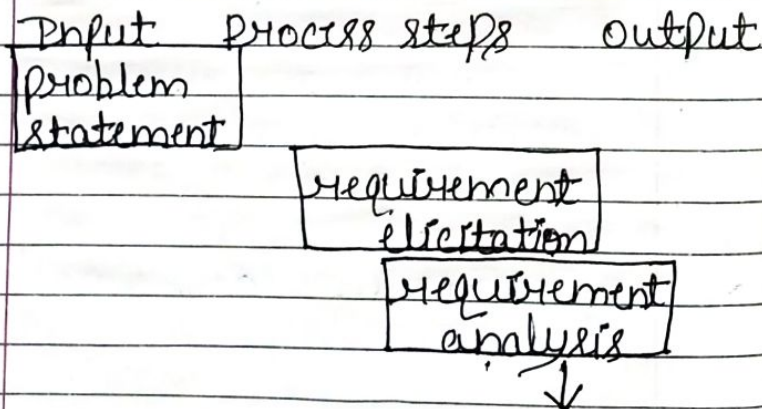
- ⇒ The various disadvantages of spiral model are:-
- i) Relies on risk assessment expertise.
 - ii) Require much more effort to be implemented

⇒ REQUIREMENT ENGINEERING :-

Requirement engineering (RE) is the science and discipline concerned with analyzing and documenting requirements.

Requirement engineering steps/ tasks :-

- ⇒ Requirement engineering consists of the following processes:
- i) requirement elicitation
 - ii) requirement analysis
 - iii) requirement documentation
 - iv) requirement Review.
 - v) requirements management



i) Requirements elicitation :-
it is to understand what a customer wants.

ii) Requirements analysis :-
After requirement elicitation, the requirements are analyzed in order to identify inconsistencies, defects, omissions etc.

iii) Requirements documentation :-

A software requirement specification (SRS) will be established which will be the foundation for the design of the software. The requirements are written so that they are meaningful not only to the customers but also to the designers in the development team.

iv) Requirement review:-

The review process is carried out to improve the quality of the SRS.

v) Requirement management:-

Requirement management is a systematic approach to eliciting, organizing and documenting the requirements of the system.

⇒ REQUIREMENT ELICITATION:-

→ Requirement elicitation is the practice of obtaining the requirements of a system from users and customers. Requirements elicitation is to find out what user really needs.

⇒ Requirements elicitation method/ techniques:-

How requirement can be properly captured from the customer into specification is done through the following methods:

i) Interviews:- The objective of conducting an interview is to understand the customer expectations from the software.

ii) Brainstorming session:-

Brainstorming is a group technique

that may be used during the requirements-gathering process to promote creative thinking.

Brainstorming for requirements is useful as it generates new ideas and provides platform to share view expectations and difficulties of implementation

iii) Facility application specification technique (fast):-

In this technique, a joint team of customers and developers work together to identify the problem, purpose and elements of the solution the team includes one expert from each field of the developer's side as well as expert from each department of the customers' side.

→ The basic of fast approach are:

- i) meeting is attended and attempted by both software engineers and customers
- ii) rules for preparation and participation are established
- iii) Rules for preparation of an agenda is suggested to cover all important points.
- iv) meeting is conducted at a neutral site.
- v) A facilitator is appointed to control the meeting a developer, a customer or an outside expert may be appointed as a facilitator.

⇒ fast contains the following activities or task:

- i) Each attendee prepare a list of objects, services and constraint etc.
- ii) list are displayed to all

- the participants.
- iii) Discussions are held on these objectives and services and then combined this for these are prepared.
 - iv) many such sessions are held and the same process is repeated eliminating redundant entries and adding new ideas till a consensus is finally reached.
 - v) as the consensus list is completed the team is divided into sub teams.
 - vi) a sub team may be asked to write the complete draft of specification and validations based on all input from the past meetings.
 - vii) This technique helps to understand the requirement and bridge the expectation gap b/w the developers and customers.

4) QFD (Quality function Deployment) :-

QFD is a Quality management technique that translates the need of the customer into technical requirement for software. QFD defines requirement in a way that maximizes the customer specification.

QFD is one of the requirement elicitation or gathering technique it is used to find out the requirement that enhance the quality of the software product. It emphasises on the requirement that are valuable to the customer. In QFD it is believed that quality in the software product is achieved by fulfilling customers' needs. Customer satisfaction is of prime concern.

QFD identifies three type of requirements.

- Normal requirements
- expected "
- exciting "

→ Normal requirements:-

Normal requirements are the requirement that must be present in the product for customer to be satisfied.

→ expected requirements:-

These requirements are implicit to the product or system these may be so fundamental that the customer does not implicitly state them but their absence may cause significant dissatisfaction.

→ exciting requirements:-

These prove to be satisfying when present

Qfd uses customer interviews and observation surveys and examination of historical data as main data for the requirement gathering activity those data are then translated into a table of requirement and this table is called customer voice table. Customer voice table contains summary of requirement.

⇒ SRS (SOFTWARE REQUIREMENT SPECIFICATION) :-

Software requirement specification is:

- i) A set of documents that contains the concise and clear specification of the functional, performance, design and interface requirement of the proposed system.

Characteristics of SRS :-

- i) Completeness :- SRS should be complete in every way because if it is incomplete there should be many problems which the system analyst has to face.
- ii) Accurate :- SRS should be accurate because if it is inaccurate then it cannot satisfy the

user needs.

- iii) Unambiguous :- SRS is unambiguous if every stated requirement has only one interpretation. SRS should not be confusing.

- iv) Consistent :-

SRS should be consistent.

- v) Modifiable :-

SRS should be modifiable

- vi) Traceable :-

An SRS is traceable if the origin of each requirement in the SRS must be clear.

- vii) Verifiable :-

A requirement is verifiable if and only

if there exist some cost effective process that can check together wheather the final product meets the requirements.

viii) Valid:-

SRS should be valid.

Components of the SRS:-

The basic components of an SRS doument are given below:

- 1) Interoduction
- 2) General Requirements.
- 3) Functional Requirements.
- 4) External Interface Require-
ments.
- 5) Performance Requirements
- 6) Design constaints.

1) Interoduction:-

Describe the scope of the

requirement definition effort.

2) General description:-

Describe the general factors affecting the product and their requirements.

Describe the relationship of the product with any other product.

Describe the general function of the product.

Provide the general charact-
eristics of the users.

3) Functional Requirements

clearly describe every function that the system needs to perform along with the corresponding input and output data set.

Specify the given input, validation conditions on inputs, operations to be performed on the input and corresponding output of operations. ∇ For example if there is any calculation then formula for that should be clearly specified.

External Interface Requirements:-

Describe all possible interface of the software with the user, hardware and with other software for input or output. These interface are shared memory, library routines etc.

5) Performance Requirements:-

There are two types of performance requirements.

These are static and dynamic.

~~Name~~ Number of terminals to be supported.

Number of simultaneous user to be supported etc.

Dynamic and requirements have:-

Execution time behavior of the system like response time, expected time for completion of the operations throughput etc.

Design constraints:-

Specify types of machine operating system and language supported.

Specify recovery requirement like what the system should do if some failure occurs.

Include security requirements like control to access data, user of passwords etc.

What do you mean by DFD?

Data flow diagrams are the diagrams which:

- Depict the flow of data through the system.
- Express system requirements in a graphical form.

The main goals of DFD are:

- To provide an overview of what data a system would process.
- To provide hierarchical breakdown of the system.
- To show what files are used and where the result flows.

- To document information flow within the system.
- To illustrate the essential components of a process/system and the way they interact.

DFD's are very useful in understanding a system and can be effectively used during analysis. The main reasons why the DFD technique is so popular are:

- It is very simple to understand and use.
- The graphical representation is a good communication tool between user & analyst.

Consideration involved in developing DFDs -

1. Symbols used for constructing DFDs

A DFD uses a limited no. of symbols to represent the function performed by a system and the data flow among these function. These symbols are:

External entity symbols :-

A rectangle represents an external entity. External entities are name with appropriate name.

Example :-

In a banking system, a customer opening an account is an external entity.

External entity CUSTOMER

An arrow or directed arc represents data flow. It represents the path over which data travels in the system.

The data flow can take place b/w

- Two process
 - File and process
 - An external entity and a process
- in the direction of the data flow arrow.

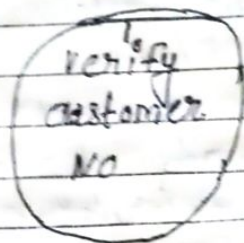
Example —

Book order
Data Flow →

(iii) Function / Process symbol

A function is represented graphically as a circle. Some system analysts use an oval or a rectangle with rounded edges to represent a function. This symbol is also called a bubble or a transformation.

Ex —



(iv) data store symbol

An open rectangle is data store

(2) Guideline for developing DFD

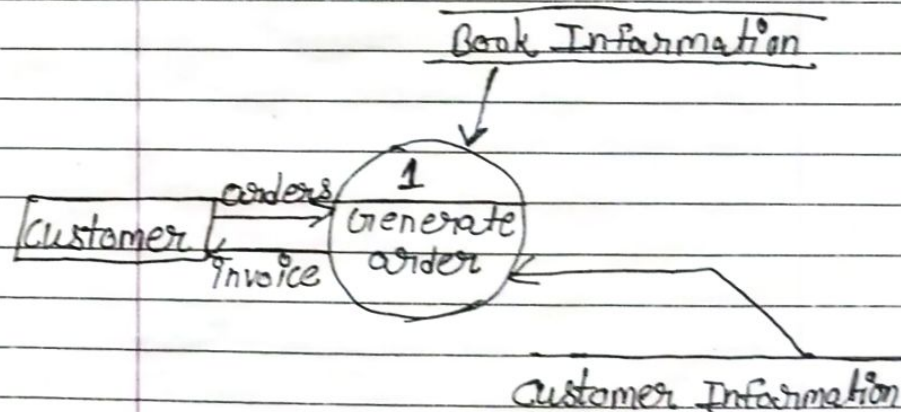
There is no standard method that is used to develop a DFD for a given problem. However the following steps are taken to develop a DFD.

- Draw a context-level diagram
- Decompose the context level diagram into a second level DFD.
- Decompose the second level

DFD to an elementary level DFD. This level of diagram explains in detail the actual processes.

(3) Context diagram

A diagram consisting of only one process which represents the entire system is called context diagram. It is a top-level data flow diagram.



(4) Detailed - level DFD

A detailed level DFD is that where the context level DFD is decomposed to elementary level using DFD symbols.

Data Dictionary

A data dictionary is a catalog - a repository - of elements in a system. A data dictionary stores.

- Details and description of dataflow
- Data stores
- Processes

Need for a data dictionary

- To manage the details in large systems.
- To document the features of the system.
- To facilitate analysis of the details.
- To search a data item.
- To evaluate characteristics of the system.
- To locate errors and omissions in the system.

Data Element

Data elements are also referred as data item elementary item or just field.

(i) Data Name

Unique name are given to data items. For each element is referred by its name in the entire system.

(ii) Data Description

It is the brief description of what the data element represents. Data description should be written assuming that the person does not know about the system.

(iii) Aliases

Aliases are synonyms or other name for the element.

For example Invoice may be called statement, Bill or Price.

(iv) Length —

Length identifies the number of spaces that are required for each data item. It is needed later during the system development process.

(v) Data values —

In some processes only specific data values are permissible. For example in many organization one-letter prefix to indicate the department like Refin A denotes the Accounting Department.

(vi) Data structure —

A data structure is a set of data items that are related to one another. Data structure are built on four relationships of

Components which may be data element or other data structure.

Relationship	Meaning	Symbol
Sequence relationship	and	+
selection relationship	either/or	[]
Iteration relationship	iteration of	{ }
optional relationship	optional	()

customer_order = customer_name +
customer_address +
{ total_items }
customer_name = First Name + (middle Name) +
Last Name
mode-of-payment = [Cash | Draft | money_order]

Advantages of Data Dictionary

- It is used to locate errors in the system descriptions.
- It helps both the user and the system analyst to

Understand all inputs, output, calculations and all components of the system.

- It is used to build the database and designing the software.

Disadvantages of data dictionary

- Data dictionary is a very ~~tidy~~ tedious and time consuming job.
- It is not acceptable to many non-technical users.

ER diagram

ER diagram is a graphical logical structure of a database. It represents

- All the entities in ^{the} logical structure.
- Constituents of the entities.
- Relationship b/w the entities.
- Types of mapping b/w relations.

Used to draw an ER diagram are:

- Entities are represented by labeled rectangular boxes. The label is the name of the entity.
- Weak entity types are represented by double rectangular boxes.
- Relationship types are represented by diamond shaped boxes.
- A line connecting two entities represents relationship. The name of the relationship is written above the line.

- Attributes are represented by ellipses.
- Key attributes have their name underlined.
- Multivalued attributes are represented by double ellipses.
- Derived attributes are represented by dotted ellipses.

Software Project management

The first layer of software engineering process is the software project management. it covers the whole development process from the beginning to the end.

1. Project Planning —

The basic goal of planning is to find the activities which are to be performed to complete a project. A good plan is that which can handle all the uncertain events which can occur during the developing of project. A good planning helps in good decision-making. The main process of the software project planning is to clarify goals and various requirements.

(2) Software cost Estimation

Software cost estimation is very important for a software organization. overestimates cost causes the management to disapprove the project whereas underestimation may lead to failure to the project by exceeding the budget.

(3) Project Scheduling

The main goal to the schedule estimation is to find out the total duration of the project and different phases.

(4) Personnel Planning

Personnel planning basically deals with team members working on the project. From the cost and overall duration of the project the number of

person for the project can be determined by dividing the total effort by the overall project duration.

(5) Team Structures

To develop a software project a proper team structure is needed. The best team structure for any particular project depends on the:

- Nature of the project and
- Characteristics of the individual team members.

(6) Software Configuration Management (SCM)

Software configuration management is the discipline for systematically controlling the changes that take place during developing.

7 Software Quality and Quality Assurance

software quality assurance is defined as a series of planned systematic sequential actions that ensure the quality of the software produced.

(8) Risk Management

Risk management is a software engineering practice with process, methods and tools to deal with all kinds of risks that might affect a project. It is a process of identifying and eliminating the risk problem before they can damage the project.

Challenging task for project management —

- 1 High level of innovation
- 2 Lack of proper skills
- 3 complexity
- 4 unclear requirements.

1 High level of innovation

Software professionals are required not only to deliver as per specifications but also use their intuition and capability to think new in order to deliver software design or software prototype or the software codes.

2 Immature software engineering

Lack of sound engineering practices makes software project management a difficult task.

3 Lack of proper skills

In the software industry, finding and retaining skilful and experienced resources is a big challenge.

4 Immature tools and techniques

Requirements and software design specifications in the software industry are still immature.

5 complexity

In case of a big size or a complex project, managing the project becomes a difficult task.

6 Unclear requirement

It is essential to identify major key requirements.

management spectrum

The four Ps of management spectrum have been described as follows —

People —

People of a project include from manager or developer from customer to end user etc. In fact project management is a people oriented activity.

Product —

Product is any software that has to be developed. The product encompasses any ~~soft~~ software that is to be built at the request of others.

Process —

In the order of priority process is positioned after people and product however, it also plays an important role for the success of the project.

Process is the way in which the software is product.

Project —

The project include all and everything of the total development process and to avoid project failure, it is important to understand what can go wrong and how to set it right.

The product or the process

A product can be defined as the fruit or output of a process is the time and can be any good or services. A process is the time and effort spent to achieve a product.

Of course a product is important but equally important is the process. A faulty processing can not yield excellent product.

The input of systematic efforts in the process of the product is thought the process that the quality of the product can be achieved. The quality of the product

also can be achieved. The quality of the product also speaks of the quality of the process.

Software Project Planning

Software Project Planning is defined as an ongoing process that sets the goals for an organization defines how these goals can be achieved and assigns some priorities to them. Project Planning involves deciding what tasks need to be done in what order to do the tasks and what resource are needed to accomplish the tasks.

Purpose of Project planning is to ensure that the end results of the project are delivered on the time within budget and are of high quality.

Goals of software project

- To Plan and document activities and commitments
- To make reasonable estimates of resources cost and schedule.

Activities involved in software

- Effort cost resource and project duration estimation.
- The plan for the software project is documented
- Software planning data are recorded.
- Another important task performed during the project planning stage is the selection of a suitable development process model.

Software size measure

In the physical world size is a measure of volume or mass. In the software world various expressions of size include measures such as line of code, number of features or functions, function points and their derivatives etc.

Two software sizing measures are widely used—

- 1 Lines of code (LOC or SLOC)
- 2 Function Points (FP)

Lines of code —

The physical LOC is a count of "non-blank, non-comment lines" in the text of the program's source code.

Logical Loc measures the number of specific definitions are tied to specific computer languages. Logical Loc is less sensitive to formatting and style conventions.

Advantages of Line of Code
LOC have the following advantages:

- Lines of code help in estimating programming time for a program.
- Easy to computer.

Disadvantages of LOC

- Difficult to relate to requirements early in the project
- Being language dependent the count may vary.
- Number of line of code differs from person to person.

= Function Pointer (FP)

Function point analysis is an objective and structured technique to measure software size. The basic idea behind to function point analysis is that the size of a software product is directly dependent on the number and type of different function it performs.

This technique is based on the principle that a system is decomposed into function units. The five major components or functional units are:

- External Inputs (E_Is)
- External output (E_Os)
- External Enquiries (E_Q)
- Internal Logical Files (ILF)
- External interface files (EIF)

External Inputs (EIs)

These are input transactions or $\pm$ information entering the system to update internal computer files.

External output (EOs)

These are the transactions where data output to the user.

External Enquiries (EE)

These are transactions initiated by the user to provide access to information.

Internal Logical Files (ILF)

It refers to a group of data or information held within the system.

External Interface Files (EIF)

These file types allow for output and input that might pass to and from other computer applications.

Advantages of FP

- Computing FP is independent of technology.
- FP are easily understood by the non-technical user also.

Disadvantages of FP

- Requires manual work on input, output, files etc.
- A great level of details is required to estimate the software size.

Plan Project Plan

A project plan specifies all of the actions necessary to successfully deliver a software product.

Project planning encompasses all the activities needed to develop and execute a project!

- Deciding what needs to be done.
- Estimating costs.
- Estimating the resources required
- Scheduling
- Defining responsibilities.

Various components of a good software project plan =

- Introduction
- Project organization
- Managerial process
- Technical process

1 Introduction

It provides an overview of the project and of the product to be developed.

2 Project organization

This section specifies how the product is to be developed both from the viewpoint of the software process and the organization structure of the developers.

Managerial process

This section describes how the project is to be managed: the philosophy, goals and priorities for management are described.

Technical process

methods tools and ~~ex~~th technical.
Software documentation.

Role of product size in cost estimation

Size of the product is an important criterion for estimating the cost of a software project. A large sized project consumes more resources than a smaller project hence it is more costly.

Methods to estimate the cost

- Algorithmic cost modeling
- Expert judgement
- Estimation by analogy
- Parkinson's Law
- Pricing to win

1. Algorithmic cost modeling

Algorithmic cost models are primarily used to make estimates of software development cost. Algorithmic cost modeling uses a mathematical formula to predict project

costs based on estimates of the project size; the type of software being developed; and other team process and product factors.

2. Expert judgement

Several expert on the proposed software development techniques and the application domain are asked to estimate the project cost.

3. Estimation by analogy

The cost of a new project is estimated by a completed project in the same application domain and then cost can be computed.

4. Parkinson's Law

Parkinson's Law states that the cost is determined by the available resource rather than by objective assessment.

5 Pricing to win
Each estimation technique has its own strengths and weaknesses.

Estimation in a project

Software cost estimation is very important for software organization. overestimated cost causes the management to disapprove the project whereas underestimation may lead to failure of the project by exceeding the budget.

Estimating project costs are important for a number of reasons.

- Along with project returns they facilitate decisions relating to project financing and funding.
- They provide the mechanism for managing cash flow during the course of the project.

- They give project manager a framework for allocating scarce resources as the project progresses.
- They provide the mechanism for revising project activity duration.

In software engineering to estimate the cost of a software is a difficult and error-prone task. In many cases estimation are made using past experience. But since the project are different and thus the past experience is not sufficient. It is difficult to make an accurate cost estimate during the planning phase of software development because of a large number of unknown factors at that time.

The cost of project is derived not only from the estimates of effort

and size but from other parameters such as hardware, travel expenses, costs training cost etc.

Various types of COCOMO model and advantages

COCOMO is one of the most widely used software estimation models in the world. COCOMO stands for Constructive cost model. It was introduced by BERRY BOEHM in 1981 to give an estimate of the number of man-months it will take to develop a software product based on size of the software. To classify a product into any of these classes the features of the product and the development team is considered.

CO Organic

In this relatively simple project, are developed by a small team of experienced programmers in a familiar environment. Such a team should have good application experience to less rigid requirements. Example of this type of project are simple business system, simple inventory management system.

(ii) Semidetached

In this class an intermediate project is developed by a team of experienced and inexperienced staff. This system falls b/w organic and embedded project. Example of semi detached system includes developing a new operating system, a database management system etc.

(iii) Embedded

In this class software project must be

developed within a set of tight hardware, software and operational constraints. Example of embedded system is flight control software for aircraft.

Basic cocomo model

it computer program development effort as a function of program size expressed in estimated lines of code. The project in a quick and rough manner.

The Basic cocomo equation

$$E = a (kloc)^b$$

$$D = c(E)^d$$

$$P = E/D$$

where

kloc is the estimated size of the software product expressed in kilo lines.

E is effort applied in person-months

D is the development time in months.

P is total number of persons required to accomplish the project.

(2) intermediate model

Intermediate model computer software developed effort function of program size and a set of "cost drivers" that include subject assessment of product, hardware, personnel and project attributes.

The Intermediate cocomo equations

$$E = a (kloc)^b \times EAF$$

$$D = c(E)^d$$

$$P = E/D$$

The value of a, b, c, d are give in the table

Project	a	b	c	d
organic	3.2	1.05	2.5	0.38
semidetached	3.0	1.12	2.5	0.35
Embedded	2.8	1.20	2.5	0.32

(i) Product attributes.

- * Required software reliability
- * size of database

(ii) Hardware attributes

- * main storage constraint
- * computer turnaround time.

(iii) Personnel attributes

- * Analyst capability
- * Application experience.

(iv) Project attributes.

- * use of software tools.
- * Required of development schedule.

Cocomo-II - Model

Developed by Dr. Boehm, Cocomo-II is the revised version of the original Cocomo. This newer version provides a quantitative analytic framework and set of tool and techniques for evaluating the effects of technology advancements on software cost and schedules.

* Software investment decisions.

For example, when to develop, reuse or purchase etc.

* Deciding how to implement a process improvement strategy.

1 End User Programming —

This category includes small systems developed by and users.

Infrastructure sector —

This category is applicable to the infrastructure development, software like operating system, database management system etc.

Intermediate sectors —

(i) Application generators and composition aids —

This subtype of project will create largely prepackaged capabilities for user programming.

(ii) Application composition sector—
Such application are complex, large diversified and require specialized developers.

(ii) system integration—
The sub-category deals with large scale, highly, embedded for system.

Application composition Estimation model—

This model is application for application composition type of project. It is suitable for project built with modern, graphic User Interface (GUI) builders.

Early Design Estimation model—
This model is used to get rough estimates of the project cost and duration before determining its entire architecture.

Post Architecture Estimation model—
This is the most detailed Cocomo-II model. It is the most detailed estimation model and is used after the project overall architecture has been developed.

Risk

A risk is any undesirable event that can occur during a project. It is a problem that can cause some loss or threaten the success of the project but which has not happened yet.

Risk factors—

There may be many reasons or factors that may lead to diverse results some risk factors that often threaten the project are.

1. Dependencies
2. Requirement related issues
3. Management related issues
4. Lack of knowledge
5. Other Risk Factors.

(1) Dependencies — Dependencies of project on some external agencies may lead to the risk factors like:

- * Non-availability of trained and experienced people
- * Inter component or inter group dependencies.
- * User based information

(2) Requirement related issues —

Requirement related issues can be:

- * Lack of clear project vision.
- * Lack of prioritization of requirements
- * Ever growing uncertain needs.
- * Rapidly changing expectations.

(3) management related issues —

These are:

- * Poor communication among staff member
- * staff's conflicts.

(4) Lack of knowledge —

The rapid rate of change of technologies and change of skilled staff may require knowledge or skill needed for a successful project.

5 other Risk factors —

- * unavailability of proper testing facilities.
- * unrealistic or unachievable performance requirement.

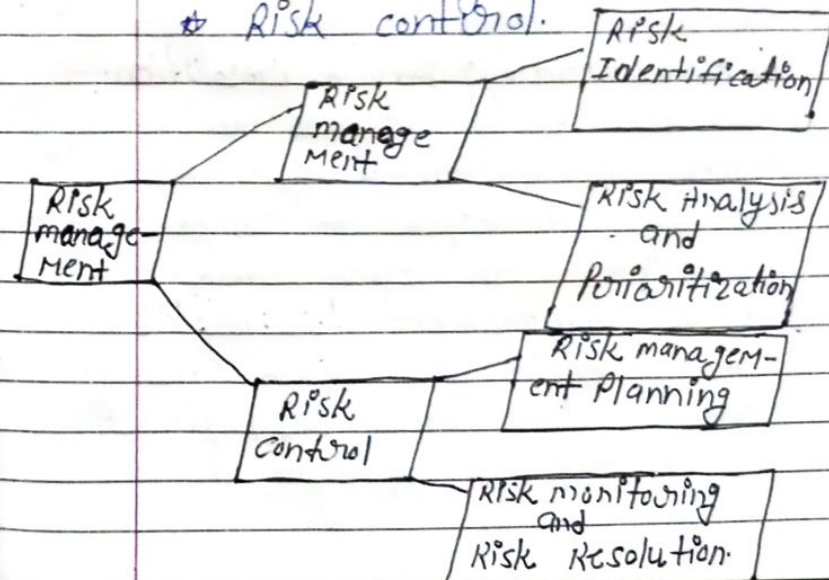
Risk management

it is a process of indentifying and eliminating the risk problem before they can damage the project.

Risk management Activities —

The process of risk management consist of a series of step that help a software team to understand and manage the uncertainty. The steps or the activities involved in the process of risk management are:

- * Risk management.
- * Risk control.



Risk Assessment —

(a) Risk Identification

(b) Risk Analysis & Prioritization

(a) Risk Identification —

it refers to examining a project and identifying areas of potential risk.

(b) Risk Analysis & Prioritization

Risk Analysis & Prioritization assesses the loss probability and loss magnitude for each identified risk.

(c) Risk Control —

it is the process of managing risks to achieve the desired output.

(d) Risk management planning —

Proper planning is done to avoid the occurrence of risks. Planning helps to turn risk information into decisions.

(e) Risk monitoring & risk resolution —

The status of the risks and actions taken is monitored. Risk resolution is the execution of the plans for dealing with each risk.

Risk management principles —

For effective risk management seven risk prevention principles need to be applied.

The seven principles that provide a framework for effective risk-

- 1 Global perspective
- 2 Forward-looking view.
- 3 open communication
- 4 Integrated management.
- 5 continuous process
- 6 shared product vision
- 7 Teamwork.

Coupling and cohesion

Coupling — Coupling is a property of a collection of modules.

Coupling is the strength of interconnections b/w modules and it is the measure of the degree of interconnections b/w modules.

The design with high coupling will have more errors.

Coupling increases as the number of calls b/w modules increases.

Types of coupling —

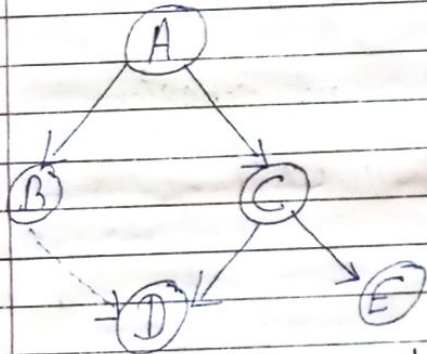
- 1 Content coupling
- 2 Common coupling
- 3 Control coupling
- 4 Stamp coupling
- 5 Data coupling

Worst
(tight coupling)

Best
(loose coupling)

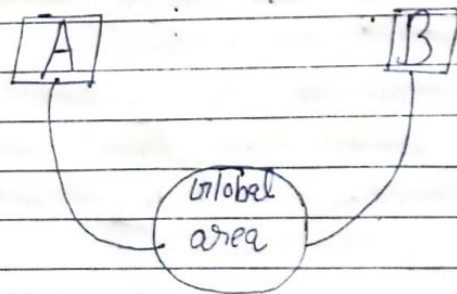
1. Content coupling -

This type of coupling occurs. # one module modifies local data value or instructions in another module. For ~~example~~ example module B branches into D event though D is supposed to be under the control of C.



(2) Common coupling -

Two modules are common coupled if # they share some global data area or data structure.



(3) Control coupling -

When module A transfers control to module B by procedure call, it is called control coupling.

(4) Stamp coupling -

Two modules stamp are coupled if ~~to~~ they communicate via a composite data item like a structure in C.

(5) Data coupling -

The an elementary data item which is passed is passed as a parameter b/w the two.

Cohesion -

Cohesion is a property or characteristic of an individual module. Cohesion is a measure of the degree to which the elements of a module are functionally related.

1. Coincidental cohesion - A module is said to have coincidental cohesion if it performs a set of tasks that are related to each other very loosely. It is the worst types of cohesion.

(2) Logical cohesion

A module is said to be logically cohesive if all the element of the module perform similar operations.

For example -

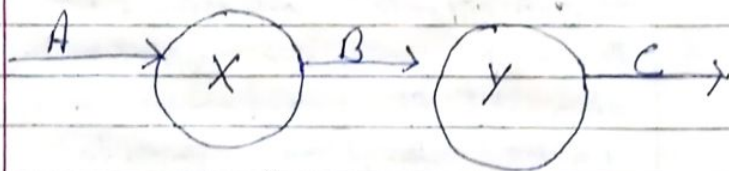
* Print appropriate messages for all

(3) Communication cohesion

A module is said to have communication cohesion if all the function of the module refer to or update the same data structure.

(4) Sequential cohesion

A module is said to possess sequential cohesion if it consists of those processing element which has the output of one as the input of the next.



(5) Functional cohesion

Functional cohesion is said to exist if ~~it~~ different

element of a module co-operate to achieve a single function.

Function oriented Design

Function oriented design is an approach to software design where a system is looked upon as something that performs a set of function. Where each unit has a clearly defined function. For example, a function create - new library member may consist of the following sub-functions: * assign - membership record * create - member record * Print - bill. Each of these subfunction may be further split into more detailed sub-functions and so on.

Object oriented Design

In the object oriented design approach the system is looked upon as a collection of interacting objects. Object oriented design focuses on the data that are to be

manipulated by the program. similar object i.e. object that share a common structure and a common behaviour, constitute a class. Inheritance allows to define a new class called derived class by extending or modifying an existing class base class.

Encapsulation is another term related to object oriented design. it is referred to as information hiding.

Design metrics—

Software metrics used during the development of design provide quantitative checks on software design. measuring various design attributes is useful not only in predicting faults but also in comparing two designs.

1. Architectural design metrics— Architectural or high level design metrics focus on characteristics of the program.



These emphasize the architectural structure and the effectiveness of modules or components.

- (2) component level design metrics— component level design metrics focus on the internal characteristics of the software component. The metrics include measures of 3 C's which are cohesion coupling and complexity.

Data structure metrics.

The set of metrics that measures the amount of data input to, processed in and output from software are called data structure metrics.

1. The Amount of data— one method for determining the amount of data is to count the number of entries in the cross-reference list. But the variables that are defined but never used are to be excluded.

(2) The data use usage of within a modules-
To computer intera-module data usage the metrics - ^{usage}
b Live variables and spans are

(3) Program weakness -
A program consists of modules
If weakness of a vice-versa
A weaker module will be made difficult to maintain.
Important of metrics in software

- * Improve quality and productivity.
- * measure the size of a software
- * know the cost to develop a software
- * cost and size estimation.
- * Determine relative complexity.
- * Analyze the defects
- * Make better decisions.
- * Provide quantitative checks on software design.

Software metrics

To control a process, humans need measure that process. This is the philosophy behind software metrics which are used to measure different characteristics of a software system or the software development process. Software metrics are quantifiable measures that could be used to measure different characteristics of a software system or the software development process. Categories of software metrics - There are three categories of software metrics. There are the follow-

1 Product metrics -

These describe the characteristics of the product such as size, complexity, design, reliability etc.

2 Process metrics -

- * Effort required in the process.
- * Time to produce the product.
- * maturity of the process.

(3) Project metrics -

- ★ Number of software developers.
- ★ cost and schedule
- ★ Productivity.

Need of software metrics -

- ★ Improve visibility, Planning and control.
- ★ Improve quality and productivity.
- ★ Measure the size of a software.
- ★ Know the cost to develop a software.
- ★ Discover more effective test technique.

Benefits of software metrics -

- ★ Monitor and control the progress of the development.
- ★ cost and size estimation
- ★ Determine relative complexity
- ★ Analyze the defects.
- ★ make better decisions.

Halstead theory of Software science -

Halstead Software science measurements were introduced in 1977 by M.H. Halstead. Halstead science is an estimation technique to find out size, time and effort of a software.

Halstead complexity is based upon count of operators and operands. It is based on interpreting the source code as a sequence of tokens and classifying each token to be an operator or an operand.

Program Length (N)

The program length (N) is the sum of the total number of operators and operands in the program.

$$\therefore N = N_1 + N_2$$

vocabulary size (n)

The vocabulary size (n) is the sum of the number of unique operators and operands.

$$\therefore n = n_1 + n_2$$

Program Volume (V)

The program volume (V) is the information contents of the program, measured in mathematical bits. It is calculated as the program length times the 2-base logarithm of the vocabulary size (n).

$$\therefore V = N * \log_2 n$$

Difficulty level (D)

The difficulty level or error proneness (D) of the program is proportional to the number of unique operators in the program.

$$\therefore D = (n_1/2) * (N_2/n_2)$$

or

$$D = (N_1 * N_2) / (2 * n_2)$$

Program level (L)

The program level (L) is the inverse of the error proneness of the program i.e. a low level program is more prone

to errors than a high level program.

$$\therefore L = 1/D$$

Effort to implement (E)

The effort to implement (E) or understand a program is proportional to the volume and to the difficulty level of the program.

$$\therefore E = V * D$$

Estimated program length

According to Halstead, The first hypothesis of software science is that the length of a well structured program is a function only of the number of unique operators and operands.

The estimated length is denoted by (N')

$$\therefore (N') = n_1 \log_2 n_1 + \log_2 n_2^2$$

Estimated program level/difficulty

Halstead offered an alternate formula that estimates the program level

$$\therefore \hat{L} = 2n_2 / n_1 N_2$$

$$\text{Hence } \hat{D} = \frac{1}{\hat{L}} = \frac{n_1 N_2}{2n_2}$$

Time to implement (T)

Since effort uses elementary discriminations as its unit of measure, the programming time would be:

$$\therefore T = E/B$$

Advantages of Halstead's theory of software science.

- (1) It predicts effort and time.
- (2) It predicts rate of errors.
- (3) It is simple to calculate.
- (4) It is useful in scheduling projects.
- (5) It measures overall quality of programs.

Software Implementation → It is the process of converting the manual or old software with the newly developed software and making it operational without disturbing the functioning of the organization.

Important activities involved in software implementation.

1. Training Personnel

The activity of training the users is a necessary part of the implementation of new software. Both software operators and users need training.

(i) Software operators training

The operators training must ensure to handle all possible operations both routine and extraordinary.

After training the software operators become familiar with the actions they need to take.

(ii) User Training

User training will enable the user to operate the software. It also instructs the user to identify the problem caused by the software.

(2) Conversion

Conversion is the process of changing from the old software. ~~to~~ the new software is fully tested and proved. Conversion involves several activities like;

- * User training to the member of the organization in which the new software is deployed.
- * converting forms. This may involve discarding of old forms, manual registers etc.
- * checking the result of the new software with the data etc.

3 Post-implementation Review

After software implementation and conversion, review of software is conducted by user and analysts.

- * How the software is working?
- * How it has been accepted?
- * Whether adjustments are needed?

Various types of Implementation

There are three types of implementation

1. Fresh Implementation
2. Replacement Implementation
3. Modified Implementation.

1 Fresh Implementation

Implementation of a totally new software by replace manual system is called fresh implementation.

(2) Replacement Implementation

Implementation of a software to replace an existing old software is called replacement implementation.

(3) Modified Implementation

Implementation of a modified application so easy to replace existing one in the same

software is called modified implementation.

issues to consider during implementation -

To ensure actual use and adoption of new technology requires to handle certain issues. There ^{issues} are related to:

1. Reuse -

it refers to reusing existing components to system. The reuse can be:

- * Application system reuse:
- * component reuse.
- * object and function reuse.

Reuse problems -

- * Increased maintenance costs.
- * Lack of tool support.
- * Not-invented-here syndrome.
- * Creating and maintaining a component library.
- * Finding understanding and adapting reusable components.

(2) Configuration management -

it refers to keeping track of different versions of software components being developed. It is a process of managing a changing software system.

Configuration management issues:

- * User roles.
- * Integration.
- * When to start using configuration management.
- * Control level.

3. Host-target development -

Host-target development issues -

- * Development platform usually has different installed software than execution platform.
- * These platforms have different architectures.

Coding style -

coding style refers to methods/rules of writing readable and understandable code to develop computer program.

- Technically correct.
- Easily understandable.
- Maintainable

Coding Efficiency -

coding efficiency is a broad term used to depict the reliability, speed and program methodology used in developing codes for an application. it is the key element in ensuring high performance. The goal of coding efficiency is to reduce resource consumption and completion time as much as possible with minimum risk to the business environment.

characteristics of a programming language -

Every programming language have many characteristics

that make it successful.

Following are the characteristics of a good programming language -

- 1 clarity, simplicity and unity
- 2 orthogonality of the language
- 3 naturalness for the application
- 4 reliability of the language
- 5 support for abstraction
- 6 portability of the program.

1 clarity, simplicity and unity
A programming language should provide a clear, simple and unified set concepts that can be used as primitives for developing and algorithm.

(2) orthogonality of the language
The ability of the programming language to combine language features freely that enables a programmer to make generalizations.

Naturalness for the application
The language should provide the appropriate data structures, operations and a natural syntax for the problem to be solved.

Reliability of the language -
The reliability of the program written in a language is always a central concern.

Support for Abstraction -
Language should support the abstraction. Abstraction means the act of representing essential features without including background details or explanation.

Portability of the program
One important criterion for any program is its transportability from the computer on which they are developed to other systems.

Coding -

Coding is the process of transforming the design of the system during the design phase into a computer language format i.e. coding is the activity where a design is translated to a runnable code.

Coding standards

There are the following standards for coding:

- * There should be no long function
- * Long function should be broken down into sub-functions.
- * Write comments before code.
- * No global variables.
- * No database access ^{script} ~~format~~

Need of coding standards are:

- The reasons for coding standard are:
- * To give a uniform appearance to the codes written by various programmers.
 - * To enhance code understanding.
 - * To make it easier to maintain the code.

Goals of coding

The main goal of the coding or programming phase is to convert the detailed design of the system produced during the phase into a programming language. The computation specified by the design.

- * Making a program more readable. it will increase code understanding.
- * To translate the design of the system into a computer language format.
- * To reduce the cost of later phases.
- * A well written code reduces the testing and maintenance effort.
- * During coding the focus should be on developing programs that are easy to write.

General rules for a good programming style

The for programming style there are some general rules and guidelines. These are:

- 1 clarity and simplicity of Expression.
- 2 Names
- 3 control constructs
- 4 Protos
- 5 information hiding
- 6 Module interface
- 7 Nesting
- 8 Module size
- 9 Program Layout
- 10 Robustness
- 11 internal Documentation.

1 clarity and simplicity of Expression

while designing the programs the clarity and simplicity of the expression must be kept in mind.

(2) Names —

While writing a program names should be meaningful. it should not be critical variable.

(3) Control constructs —

As much as possible single-entry single-exit constructs should be used. This means that the program starts from one defined point.

(4) Gotos —

User of gotos should be avoided as much as possible.

(5) Information hiding —

information hiding should be supported where possible.

(6) Module interface —

If a module has complex interface, it should be carefully examined.

7 Nesting —

Where possible try to avoid deep nesting in coding.

8 Module size —

The module size should be uniform. The size of module should not be too big or too small.

9 Program layout —

Program should be organized in such a way so that it has more readability.

10 Robustness —

A robust program handles situations invalid data processing situation.

11 Internal Documentation —

internal documentation helps to increase the readability and understanding.

Software testing

To make the testing process simple the testing activities are broken into smaller activities. The system is broken into subsystems.

In order to conduct a proper testing, the following order should be performed:

- unit testing.
- integration testing
- system testing

Role of software testing during software development life cycle-

The role of software testing in the SDLC is listed below:

Software Development Phase	Testing
Requirement Specification	• To identify the test strategy • To check the sufficiency of requirements

Design

- To check the consistency of design with the requirements.
- To check the sufficiency of design.

coding

- To check the consistency of implementation with the design
- To check the structural and implementation.

Testing

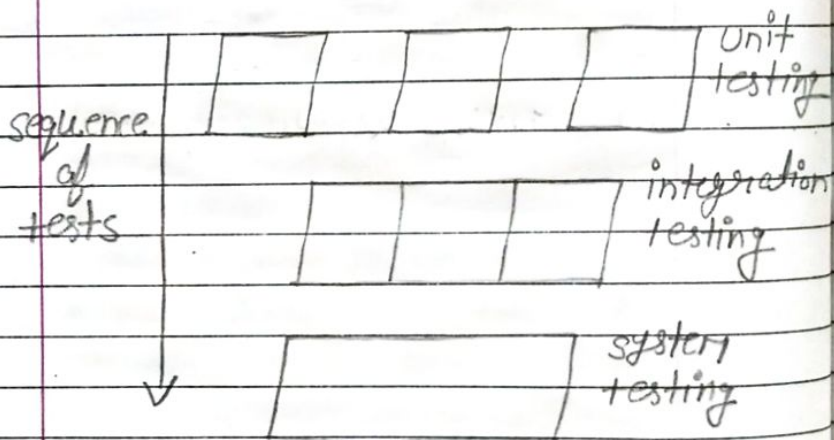
- To check the sufficiency of the test plan.
- To test the application programs.

Software Testing

Software testing is basically the process of executing a program with the intention to detect errors in the code.

in order to conduct a proper testing the following order should be performed:

- unit testing
- integration testing
- system testing



(1) Unit testing

Unit testing is the process of checking the functionality of the method. Unit testing begins after the source code is developed and verifiable for the correct syntax. The primary goal of unit testing is to take the smallest unit of testable software and to find out whether it works exactly as expected.

The following are the tests that are performed during unit testing.

(a) Module interface test

In Module Interface test it is checked whether the information is properly flowing into the module and properly coming out of it under the specified operation conditions.

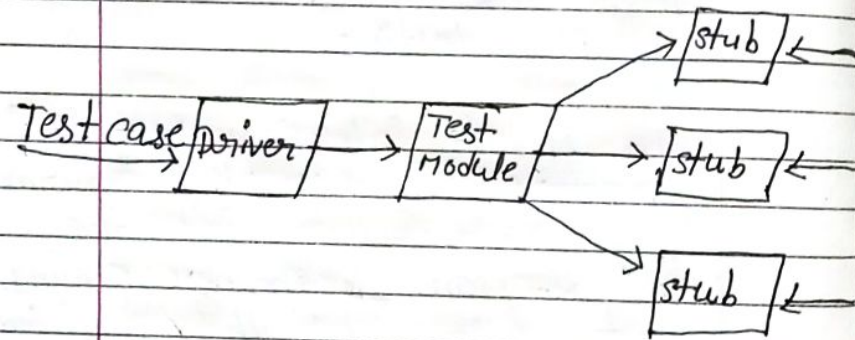
(b) Performance test

it is desing to verify res-
ponses time, throughput and
traffice

(c) Local data structure test -
It is testing to check if the local data within module is stored properly or not.

(d) Boundary conditions test -
Boundary conditions are tested to ensure that the program is working properly at the boundary conditions.

Unit Testing procedure -



(2) Integration Testing

Unit test does not guarantee that these modules will work fine if these are integrated together as a whole system.

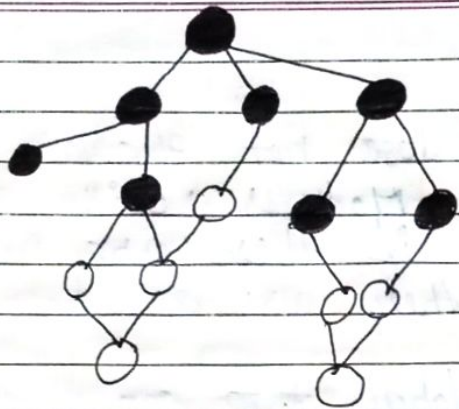
- * Global data can create problems
- * Data can be lost across an interface.
- * One module can have an adverse affect on another.

Two basic types of integration are usually used:

- (a) Top-down integration
- (b) ~~down~~ Bottom up integration

(a) Top-down Integration

Top-down integration is an incremental approach that requires the highest-level modules. Continue the process until all modules have been integrated and tested.



Advantages of top down Integration.

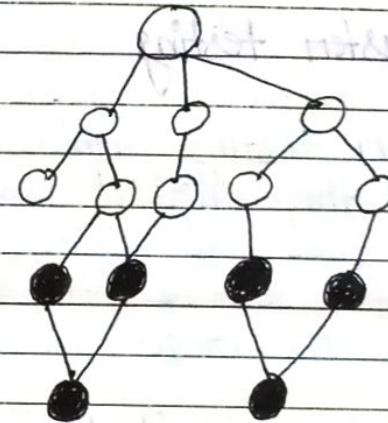
- * Top-down level interface are tested first.
- * one driver at the root is required

Disadvantages of top-down Integration.

- * Delays verification of low-level modules
- * Not enough data in the stubs to feed back to the calling module.

(b) Bottom up Integration
The bottom up integration

approach requires the lowest level module be tested and integrated first using a driver. Continue the process until all modules have been integrated and tested.



Advantages of Bottom up Integration

- * stubs are not required as developing and testing starts with actual modules.
- * Allows early verification of low-level modules.

Disadvantages of Bottom up Integration

- * Need for drivers complicates test Management
- * Delays Verification of high level behavior.

(3) system testing

system testing implies testing of the entire software.

The goal of system testing is to see if the software meets its requirements.

(a) Recovery testing

(b) security testing

(c) stress testing

(a) Recovery testing —

Recovery testing is designed to examine how easily and completely the system can recover as system failure. Recovery testing forces the software to fail in the variable.

(b) security testing —

- * Protection mechanism built into a system to determine.
- * other system cannot gain access to the system and information with it.

(c) stress testing —

stress testing
test the system performance in abnormal situations.

Examples:

input data rate may be increased in magnitude to determine how input function will respond.

Design of test cases

A test case is a set of instructions designed to discover a particular type of error or defect in the software system by inducing a failure.

The goal of test case selection is -

- ★ To select the test cases such that the maximum possible number of faults.
- ★ To ensure that there is no error in the program and if it is so then it should be immediately depicted.

simple	serial number of step
Step No.	
Step or Activity	Detailed operation or procedure.
Criteria for success	Expected result whether the code passed the test or not.
status	

Detailed

Test case Id	serial number to test case
Purpose	Brief idea about the case.
Test created by	Name of test creator
Test data	Input, variable and data
Actual result	What is actually done.

Designing/writing good test cases -

Good test cases save time and money and even the reputation of the organization.

1. simple and transparent.

Create test cases which are as simple as possible. They must be clear and concise as author of test case may not execute them.

2. Avoid test case repetition
Do not repeat test cases.

3. Avoid assuming

The preparing test case do not assume functionality.

and features of the software
 4 Impermanent testing techniques
 Testing techniques must be used for effective testing.
 5 Test cases must be identifiable
 - use exact and consistent name of forms, fields etc.

Debugging

The determination of error and removal of those error is called debugging. Debugging means identifying locating and correcting the bugs usually by running. Debugging can be viewed as a problem solving process. There are various debugging stages stages are-

Locate the error	Design the error repair	Repair the error found	Retest the modified program
------------------	-------------------------	------------------------	-----------------------------

Debugging Techniques

Some important debugging techniques

- 1 Brute force debugging
- 2 Backtracking debugging
- 3 Debugging by induction
- 4 Debugging by deduction
- 5 Debugging by testing.