1.2.1 Characteristics of Software

Q. State characteristics of software..

– Software means anything which is not hardware but which is used with hardware, such as window OS (is system software) in computer (is hardware).

You will get more precise idea about the characteristics of software and how it differs from hardware from the below table.

Table 1.2.1 : Characteristics of software and how it differs from hardware

Sr. No.	Software	Hardware
1.	It is developed or engineered.	It is manufactured.
2.	It doesn't wear out as it is not prone to environmental problems.	It wears out as the time passes due to the affects of dust, vibration, temperature extremes and many such environmental problems.
3.	There are no software spare parts which can be used to replace the software.	When hardware fails, it can be replaced by spare parts.
4.	Software is untouchable. It is the code or instructions that tell a computer/hardware how to operate. It has no substance.	Hardware is a physical device something that you're able to touch and see.
5.	Software is usually generic but it can also be custom built (developed according to the customer specification).	It is manufactured or assembled by using the existing components.
6.	Software is invaluable as it can be installed in any hardware.	Hardware has no value without software in it. If computers (h/w) had no operating system (s/w), then no customer will buy it.
7.	Examples: Microsoft, Windows any operating system that allows you to control your computer. Example 2: Internet browsers, video games, applications like Payroll etc.	Examples: disks, disk drives, display screens, keyboards, printers and chips.

2.2 Generic Process Model

Q. List the umbrella activities followed in generic process model.

This model defines a set of *umbrella activities* which are also a must for any software engineering process as shown in the Fig. 2.2.1.

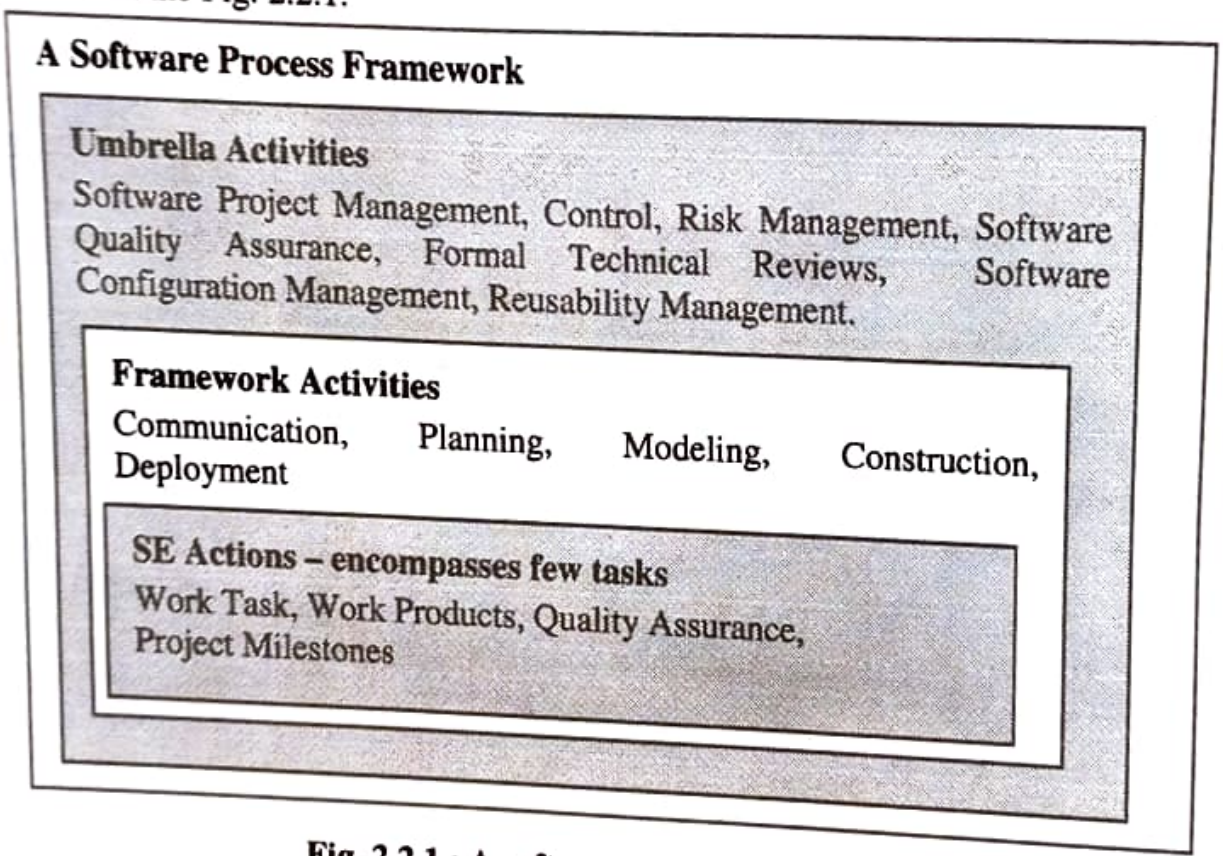


Fig. 2.2.1 : A software process framework

Again, each *framework activity* contains a set of *software engineering activities* which is a collection of tasks that develops a major software product.

2.2.1 Framework Activities

First, we take a look at the ***generic process framework activities*** *that are must for the development of any software project :*

☞ **Communication :**

The main cause of communication is requirement gathering. This activity establishes sufficient interaction or collaboration between the developer and the customer for gathering the requirements and knowing the expectations of the customer.

Example : We can explain the **work task** regarding the **communication activity** of a simple project as listed below :

- Making the list of the end-users, software engineers and support people for the project.
- Inviting all of them for an informal meeting.
- Ask each end-user to make a list of features and functions required.
- Discuss these requirements and prepare a final list.
- Arrange the requirements according to their priority.
- Note the areas of uncertainty.

☞ **Planning :**

This activity defines the software development process to be conducted. It describes all the needed technical tasks, possible risks, the resources that are required, the work product to be produced and the schedule to workout the whole process. This activity plans the work, identifies the resources, tasks and sets the schedule.

☞ **Modeling :**

This activity creates a model (blueprint) which clearly describes the software requirements and the design that will achieve these requirements. This is helpful to both customer and the developer respectively, to understand what he wants from the software and how he can develop it. Modeling is composed of two main activities-*analysis* (requirements gathering, elaboration, negotiation, specification and validation) and *design* (data design, interface design and each module level design).

☞ **Construction :**

This activity includes code generation either manually or using automated tools and then testing the code to correct the errors if any.

☞ **Deployment :**

The software (as a complete product or in a partial stage) is delivered to the customer who then checks the product and provides feedback on evaluation.

The framework activities are applied on every project but the degree of tasks depend on the :

- Type of the project
- Characteristics of the project
- Agreement of the project team on common views.

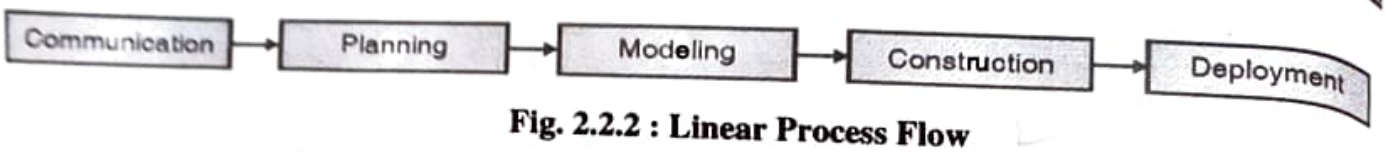


Fig. 2.2.2 : Linear Process Flow

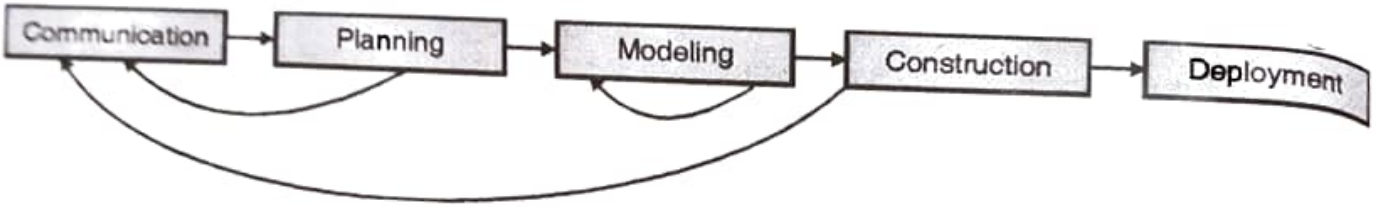


Fig. 2.2.3 : Iterative Process Flow

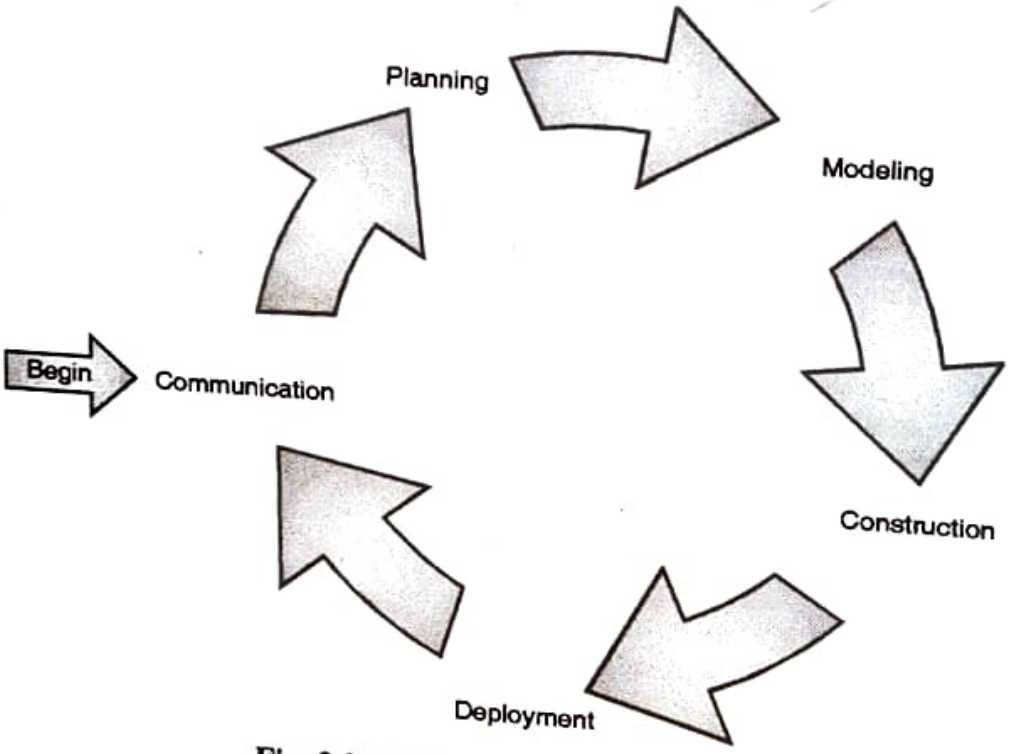


Fig. 2.2.4 : Evolutionary Process Flow

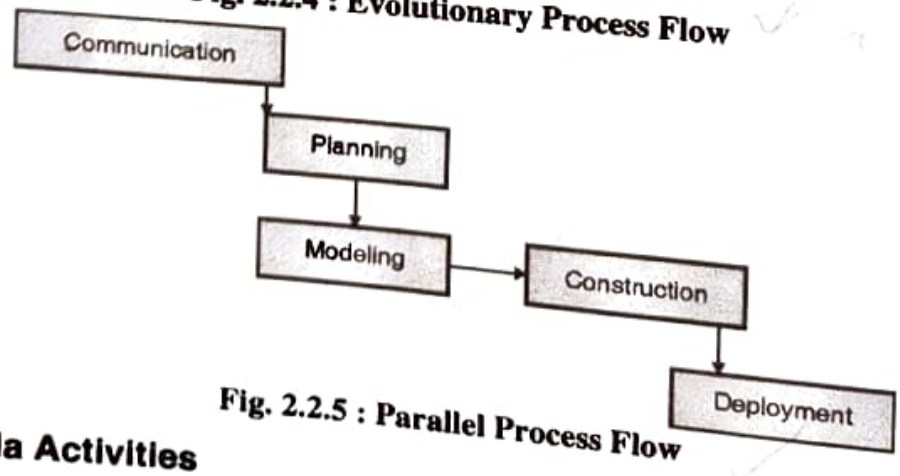


Fig. 2.2.5 : Parallel Process Flow

2.2.3 Umbrella Activities

Now, we look at the ***Umbrella activities*** *that are applicable throughout the software process :*

- **Software Project Tracking and Control :** Software team assesses the progress of the project plan time to time and takes necessary action to maintain the schedule. Thus, software team tracks and controls the project schedule.
- **Risk Management :** Software team assesses the risks that may affect the outcome of the project or say the quality of the product.
- **Software Quality Assurance :** Software team defines and conducts the activities needed to preserve the quality of the software product.
- **Formal Technical Reviews :** Software team assesses the technical efforts to find and remove the errors before they are forwarded to the next action.
- **Measurement :** Just the coincidence is the four P's of Software Engineering : Project (the task at hand), Process (the manner it is done), Product (the object produced) and People (by whom it is done). Software team collects all the project, process and product measures so that it can be used in combination with all other framework and umbrella activities.
- **Software Configuration Management :** It is about managing the changes and their version throughout the software process.
- **Reusability Management :** defining the criteria for work product reuse and establishes mechanisms to achieve reusable components.
- **Work Product Preparation and Production :** proper planning so as to create work products such as models, documents, logs, forms and lists.

Syllabus Topic : The Waterfall Model

2.3 Waterfall Model

Q. Explain waterfall model in brief.

This model was proposed by the Winston Royce. It is also called as a ***classic life cycle.*** It suggests systematic sequential approach for software development. It is oldest paradigm for software engineering.

It begins with software requirement and customer specification and progress through planning and testing.

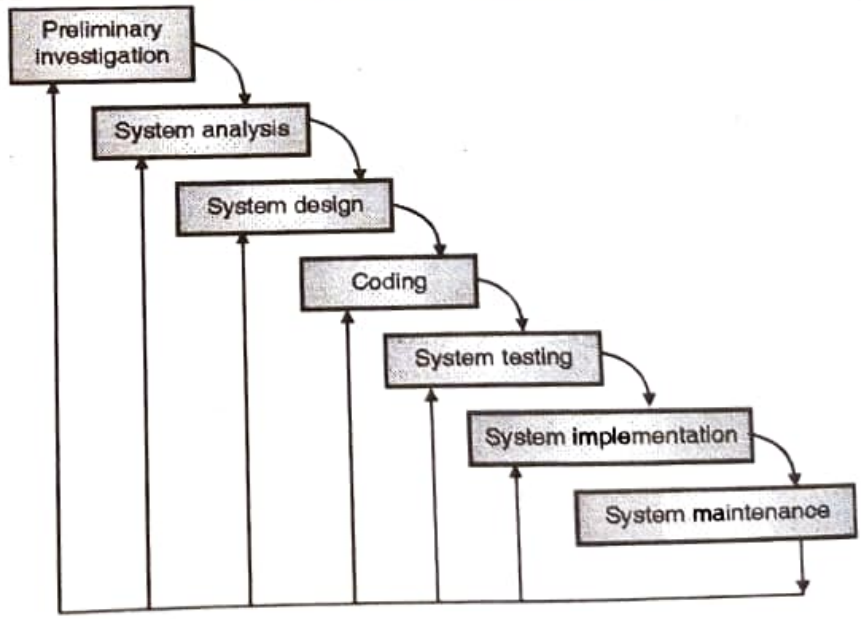


Fig. 2.3.1 : Waterfall model

Step 1 : Preliminary Investigation

Preliminary investigation means total inspection of the existing system i.e. clear understanding of the system.

Its basic task is to find out real problem of the system with causes and complexity of the problem. Its secondary but very important task is to find out all possible solutions to solve that problem and according to that which solution is feasible for the system in terms of technology, cost, operational. Its last task is to mention all benefits can be expected after problem is solved.

So this phase is divided into three main goals as follows:

1. Problem identification
2. Possible and feasible problem solution i.e. Feasibility study.
3. Expected benefits after the problems are solved.

Problem Identification

It requires to completely investigate the environment of the system. Generally it requires studying two environments - Internal environment and external environment, which are listed below :

Sr. No.	Internal Environment	External Environment
1.	Company Management	Customers
2.	Employees of all departments	Management consultant
3.	Internal auditors	External auditors
4.	Data Processing department	Government Policies
5.	Financial Reports	Competitions

There are normally seven types of problems encountered in the system :

1. **Problem of Reliability :** If system may not work properly or same procedures give different (i.e. unreliable) result.
2. **Problem of Validity :** Reports contain misleading information.
3. **Problem of Economy :** System is costly to maintain.
4. **Problem of Accuracy :** Reports have many errors.
5. **Problem of Timeliness :** Every work requires large time.
6. **Problem of Capacity :** Capacity of the system is inadequate.
7. **Problem of Throughput :** System does not produce expected results, or we can say system has more capacity but it accomplishes very less work as compared to capacity.

The main advantage of waterfall model is, it exactly pin points the problem. So it is very useful in setting all goals of the system as well as used to decide system boundaries.

Feasibility Study

Feasibility study is essential to evaluate cost and benefit of the proposed system. This is very important step because on the basis of this; system decision is taken on whether to proceed or to postpone the project or to cancel the project.

Need of Feasibility study

Q. What is the need of feasibility study? Explain its types.

1. It determines the potential of existing system.
2. It finds or defines all problems of existing system.
3. It determines all goals of the system.
4. It finds all possible solutions of the problems of existing system. We can call it as proposed system.
5. It finds technology required to solve these problems.

6. It determines most suitable solution
7. It determines the required hardware and software.
8. It does the cost estimation in terms of cost of hardware required, software required, designing new system, implementation and training, proposed maintenance cost.
9. It avoids costly repairs, crash implementation of new system.
10. It chooses such system which is easy for customer to understand and use so that no special training is required to train the customer. It may give some training to employees of the system.

Method-Steering committee

This committee conducts detailed study. This committee first studies existing system and identifies all problems and looks into three types of feasibility study. Those are given in Fig. C2.1:

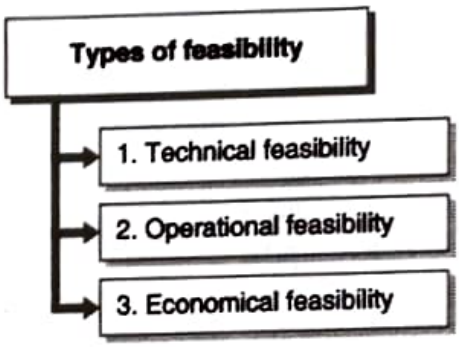


Fig. C2.1 : Types of feasibility

1. Technical feasibility

The committee first finds out *technical feasibility of the existing system.* It involves following steps :

1. It determines available hardware.
2. It determines available computer with configuration.
3. It determines available software.
4. It determines operating time of available system that is computer, hardware software.

After that it finds out *technical feasibility required for the proposed system.* It involves following steps :

1. It mentions new hardware requirements of proposed system.
2. It mentions computer with new configuration requirement of proposed system.
3. It mentions new software requirements of proposed system.
4. It mentions new operating time of available system that is computer, hardware, software.
5. It mentions old as well as new facilities which will be provided by the proposed system.
6. It mentions all benefits of the system.

2. Operational feasibility

It is also called as behavioural feasibility. It finds out whether the new technology or proposed system will be suitable using three type of aspects; that are human, organizational and political aspects.

It involves following steps :

1. It finds the ease of operation of proposed system compared to existing system.
2. It finds if the users of the system require any extra training ?
3. It finds out whether the user or customer of the system requires extra training or not ?
4. If it requires any retraining then it is accepted by user as well as customer or not ?
5. It finds if any job reconstruction is required or not ?
6. It finds if this reconstruction of the job is accepted in organization ?
7. It also finds if it is acceptable then what should be the skill sets of that job.
8. Watches the feelings of the customers as well as users.
9. It should provide right and accurate information to user or customers at right place as well as at right time.

3. Economical feasibility

Here, steering committee finds total cost and all benefits as well as expected savings of the proposed system.

There are two types of costs – one time cost and recurring costs.

One time cost involves :

1. Feasibility study cost.
2. Cost required to convert existing system into proposed system.
3. Cost of hardware's, OS, application software.
4. Technical experts consulting costs.
5. Cost of training.
6. Cost of Documentation

Recurring cost involves following :

1. Cost involves in purchase or rental of equipment.
2. Cost of phones and mobiles Communication equipment.
3. Cost of Personnel search, hiring, staffing.
4. Cost of Salaries of employee's.
5. Cost of supplier's.
6. Cost of maintenance of equipment.

Step 2 : System Analysis

This phase of the water fall model is nothing but complete understanding of all important facts of the business using preliminary investigation. It involves following steps :

1. It is nothing but study of all components of the system as well as inter relation between all components of the system and relation between components and environment.
2. It determines what is to be done in the organization.
3. It finds the **procedures** of how to do that.
4. Which is input data ?
5. What is the procedure through which inputs are to be converted into output ?
6. When transaction should occur on the data.
7. When problem arises, determine the solutions to solve it and what are the reasons of those problems.

Objectives of System Analysis: The main roles of the system analysis are :

1. Define the system.
2. Divide the system into smaller parts.
3. Finds all nature, function and inter-relationship of various parts of the systems.
4. If the system is not analyzed properly then there may be problem in the Preliminary investigation phase.

Step 3 : System Design

The main objective of this phase of waterfall model is to design proposed system using all information collected from preliminary investigation and directed by the system analyst. This is very challenging phase. It includes following steps :

1. Design of all types of inputs of proposed system.
2. Design of all types of outputs of proposed system.

3. Design of the procedures which convert input to output.
4. Design of the flow of information.
5. Design of the information which is required to store within a files and data bases Volumes.
6. Design of collection of inputs using forms (Manual forms).
7. Design in terms of program specification i.e. logical design.
8. It determines the hardware cost, hardware capability.
9. It determines the speed of software.
10. It determines error rates, and other performance characteristics are also specified.
11. It also considers the changes to be made in the organizational structure of the firm in design.
12. This phase also designs standards for testing, documentation.

Generally traditional tools are used for the designing of the procedures that are as follows :

- Flowcharts
- Algorithms
- IPO (i.e. Input Processing and output) and HIPO (Hierarchy of IPO) charts
- Decision tables
- Data Flow diagrams

1. If system design phase is facing problem during the design then first go back to the system analysis phase and redesign the system but if problem is not solved then go for preliminary investigation.
2. If System design phase produces all expected results then it goes to next phase.

Step 4 : Coding

This phase is just implementing the design in to programming language that means it actually develops the proposed system. It involves the following steps :

1. It first of all, finds out the best suitable programming language that is suitable for the design as well as also suitable in the organization.
2. It accepts design and break system modules into smaller programs.
3. It develops or writes program for each part in selected programming language.
4. Prepares documentation that means add necessary comment lines wherever necessary within a program.
5. Now it combines all small programs together and builds one big program.
6. If any problem occurs during the coding phase then waterfall model tries to solve it by repeating system design phase:
   a) If that problem does not get solved then waterfall model repeats system analysis phase and system design phase.
   b) If that problem does not get solved then waterfall model repeats from first phase preliminary phase through system analysis phase and system design phase.
7. If coding phase produces all expected result then it goes to next phase.

Step 5 : System Testing

This phase includes the testing of the code or programs developed by the coding phase. This includes following steps :

1. First of all, it finds out all possible expected results (i.e. output data) for the set of input data.
2. It also checks the validity of the input data as well as checks expected output data.
3. It finds out all wrong results and immediately tries to correct it by repeating coding phase.
4. It finds the speed of functions using special codes.
5. It determines whether each program can perform the intended tasks or not?

6. It checks result by test data.
7. It checks the logic of the individual programs.
8. It checks interfaces between various programs.
9. It checks quality of code in terms of speed and space.
10. It checks whether system has produced correct and desired results which lead to designated goals.
11. If testing does not produce expected result then waterfall model tries to solve it by repeating system design phase and coding.
    a) If that problem does not get solved then waterfall model repeats system analysis phase through system design phase and coding.
    b) If that problem does not get solved then waterfall model repeats from first phase i.e. preliminary phase through system analysis phase, system design phase and coding.
12. If testing phase produces all expected results then it goes to next phase.

Step 6 : System Implementation

System Implementation is not creative process but it is some what difficult task. This phase has two parts - *implementation* and *evaluation* of the system.

☛ Implementation

There are two ways of implementation. Those are as follows :

1. Implement proposed system with existing old system and find out performance of the both systems and slowly replace new system with older one.
2. Totally replace old system with proposed new system.

Risk factor of second type of implementation is more as compare to first one. Second step needs strict evaluation.

Both types of implementation consist of following steps :

1. It prepares site for new proposed system.
2. It installs required hardware within a system.
3. It installs required software in a system.
4. It installs a developed code i.e. programs in a system.
5. It prepares training program for user of the proposed system as well as customers of the system.
6. It prepares user manual which includes all the steps which give guidance to user.
7. It gives training to all types of user of proposed system.
8. Observe the system when users of system are using it.
9. If users are facing any problems regarding the new system, it tries to find out exact phase from where root cause of the problem starts, and accordingly starts waterfall model.

☛ Evaluation

Evaluation is nothing but feed back for the system. It is very essential check point of the system which is the process of verifying the capability of a system. It continuously evaluates and checks whether proposed system is meeting the objectives or not. It includes :

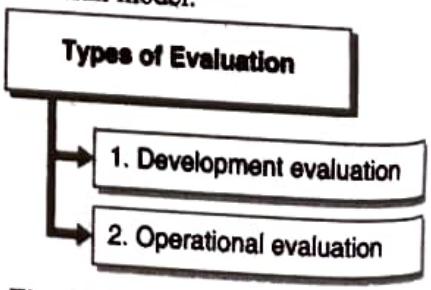


Fig. C2.2 : Types of Evaluation

➜ 1. Development evaluation

- It checks whether the system is developed within time.
- It checks whether the system is developed within the budget.

- System is assed by development methods and tools.

➜ **2. Operational evaluation**

- It checks response time of proposed system.
- It checks whether it is really easy to use or not?
- It checks accuracy of computations (It is seen in testing also).
- It checks storage capacity.
- It checks reliability.
- It checks functioning of the existing system.
- Collects necessary feedback from users.
- It finds all benefits of the proposed system.
- Collects information of attitude of different persons regarding proposed system.
- It evaluates cost, time and effort taken for the overall project.

Step 7 : System Maintenance

Maintenance is the process, in which it finds out essential changes (i.e. new trends) of the market or business or to correct some errors and tries to implement it in the existing system. There are usually three types of maintenance, that are :

1. **Correction :** Some times, proposed system has few types of errors; and it is the duty of software engineer to correct it as soon as it is encountered by the user. Generally there are four types of errors, that are as follows :
   - Minor changes in the processing logic.
   - Errors detected during the processing.
   - Revisions of the formats for data inputs.
   - Revisions of the formats of the reports.

   These errors can be corrected by repeating waterfall model from coding phase through testing, implementation and maintenance.
2. **Adaptation :** Some times, our proposed system is executable on Windows environment, but somebody wants to run it in LINUX environment, or some other operating system. Then we are required to design our proposed system from third phase that is from System Design phase.

   This is actually error free system. It is good so other people also want same system in
3. **Enhancement :** Because of new technology and business competition, organization needs to imply or to add new functions or additional capabilities to the proposed system.

   After some time, people think that some techniques may be used in the system so some additional features can also be added into it. Sometimes, new hardware is required to add some extra features.

   For enhancement it may repeat whole system or may repeat it from design phase or some times from coding phase.

☞ Advantages of waterfall model

1. It defines very first software development process.
2. The product of waterfall model always defines all constraints of the organization.
3. It always produces a good quality product in terms of space and time.

☞ Disadvantages of waterfall model

There are some disadvantages of the waterfall models that are as follows :

1. Real products rarely follow this sequential flow.

2. Because of iteration, changes can cause confusion as the project team proceeds.
3. It is very difficult for customer to state all the requirements in one time.
4. Many projects face this uncertainty at beginning only , so it is very difficult to design next phases.
5. Time span required for each phase could not be specified.
6. Naturally project requires more time.
7. Project becomes lengthy also.
8. Customer should have patience.

To overcome these drawbacks of waterfall model, a new model was designed known as ***'Incremental process model'***. Incremental process model has two process models 1) Incremental Model and 2) Rapid Application Development model.

Syllabus Topic : Incremental Process Models

2.4 Incremental Process Models

Method

- As soon as customer comes to software engineer and gives requisition of new project to software engineer; he starts to collect information from user to start preliminary investigation.
- He immediately increments towards second phase and third phase and starts to analyze and design the system.
- He starts to convert design into coding i.e. as time or calendar grows, some functions of software are also progressed towards completion.
- When testing of first requirement goes towards completion stage, customer comes with second requirement. Immediately he starts to collect all essential information for second increment i.e. preliminary investigation of the second requirement in parallel.
- He starts implementing first requirement i.e. module 1 as well as starts designing the second requirement i.e. module 2.
- Maintenance phase of first module is continuing then simultaneously coding of second module is starting as well as customer may ask for third requirement and he increments for the third incremental model.

This is explained in following graph ;

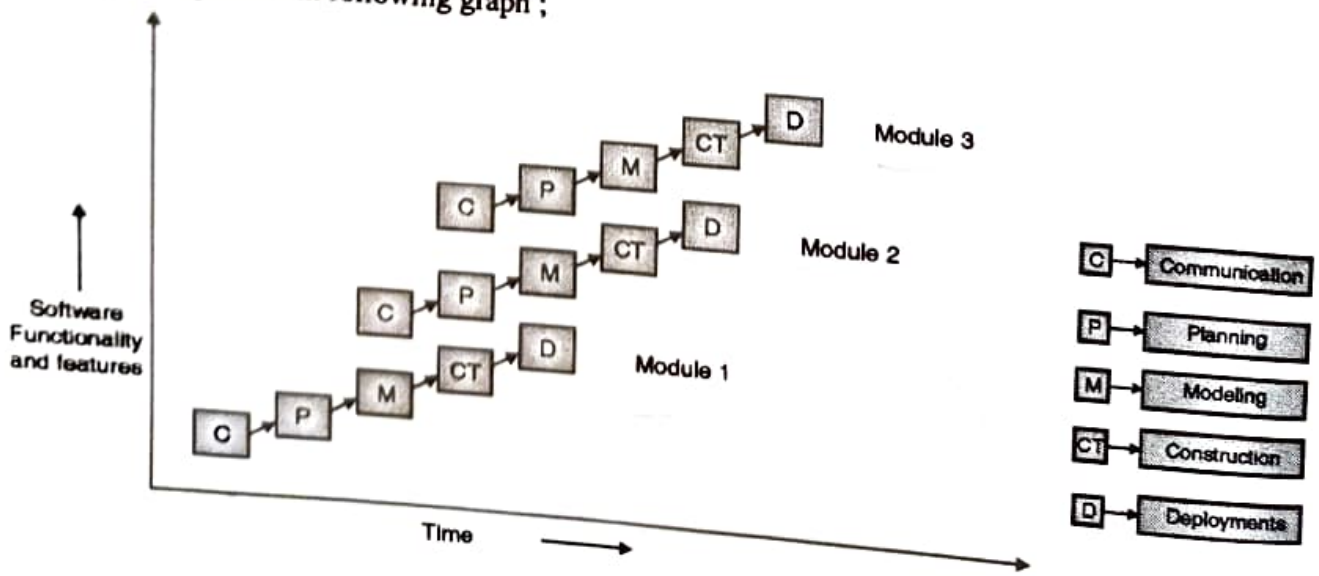


Fig. 2.4.1 : Incremental model

Example of Incremental Model

The software word processor is designed using incremental model. It is designed as follows :

1. Customer comes to software engineer and gives requisition of new project of a **word processor** to software engineer.
2. He starts to collect information from customer. Customer tells him to design file management system with all editing functions and document production functions.
3. Immediately software project team starts preliminary investigation to decide the plan.
4. Software engineer immediately increments towards second phase and third phase and starts to analyze and design the system. When third phase of first increment leads towards completion state then customer comes with second requirement i.e. about more sophisticated editing capabilities like copy, paste, and find capabilities. He starts preliminary investigation of the second increment in parallel.

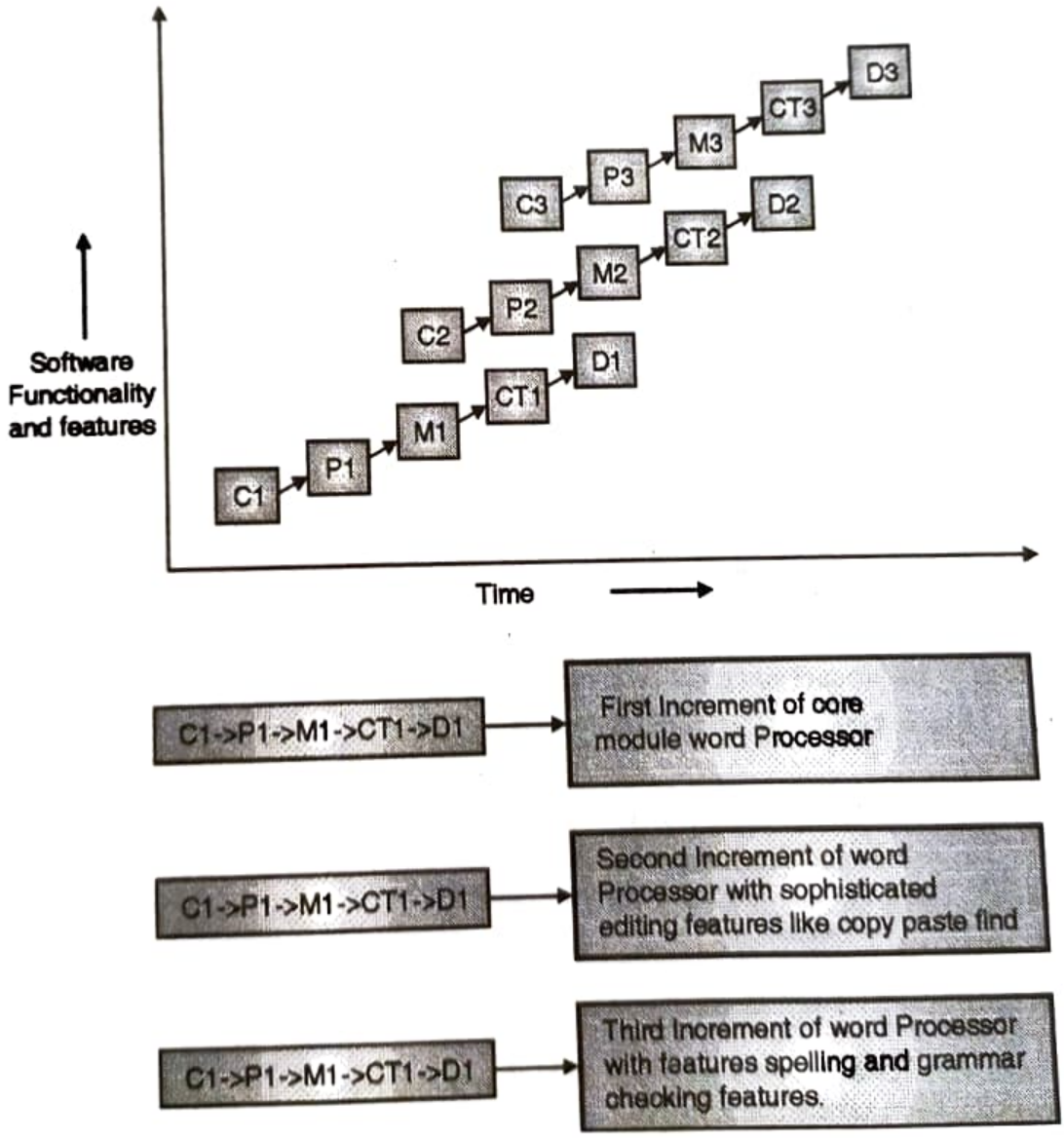


Fig. 2.4.2 : Example of incremental model

5. He starts to convert design into coding of first increments as well as he designs plan and objectives of the second increment for these sophisticated editing capabilities.
6. He starts implementing first requirement i.e. core module as well as it starts design of second requirement i.e. second module.
7. Maintenance phase of first module is continuing then simultaneously coding of second module is starting as well as customer may ask for third requirement that is spelling and grammar checking feature and it starts for the third incremental model.

This is explained in Fig. 2.4.2.

Method used in Incremental model

1. Generally in incremental model, the first increment is a core product which includes maximum basic requirements.
2. Second increments and third increments include all supplementary features (known and unknown features) which increase additional features with more functionality in a product.
3. This process is repeated till the completion of the final product.

Advantages of Incremental model

Q. List the advantages of Incremental model.

1. Preliminary investigation time is very small or reduced.
2. It requires very less time as compare to waterfall model
3. It requires less number of staff to develop the system.
4. Customer is satisfied because of quick development of the new system.
5. If system has many increments its functionality also increases; thus, the product has many features.

Disadvantages of the Incremental model

Q. List the drawbacks of Incremental model.

1. All the tasks are not decided in first phase so there may be some problem with designing phase. That means, overall design of the system is not so good.
2. It may produce software, which requires large space in memory.
3. Some times, speed of software is also slow.
4. Delivery date of the product could not be decided.
5. New levels of increments may require new hardware because old hardware is involved with old increments.
6. Some times, it produces same code in different module because of partial functionality.
7. Some times, problem is not identified properly.
8. Quality of software or product is poor.

2.5 RAD Model

RAD is Rapid Application Development. It is high speed adaptation of waterfall model. It is the example of incremental software process model.

The main disadvantage of incremental model is - the quality of product is very poor as well as it also requires more time to develop. So this model is only designed to produce a good quality product in very short duration of time.

Method

1. It also adapts generic framework activities like waterfall model that means five phases.
2. This allots sufficient time for study of the existing system.
3. It accepts sufficient number of staff for development of the system.
4. Staff is distributed into various teams (i.e. senior programmer junior programmer ,team leader, project leader etc).
5. Communication phase is used to understand business problem as well as it tries to identify all type of problems and gather all type of information.
6. Planning is essential because it divides work into manageable different parts and distributed among the various teams.
7. Modeling will be accomplished by different teams simultaneously. It includes three different phases. Business modeling, Data modeling and process modeling.
   - Business modeling collects all essential information about the product from market view or from the business point of view.
   - Data modeling shows the flow of data using some techniques like DFD.
   - Process modeling performs each process which handles a flow of data. It decides activities performed in each process.

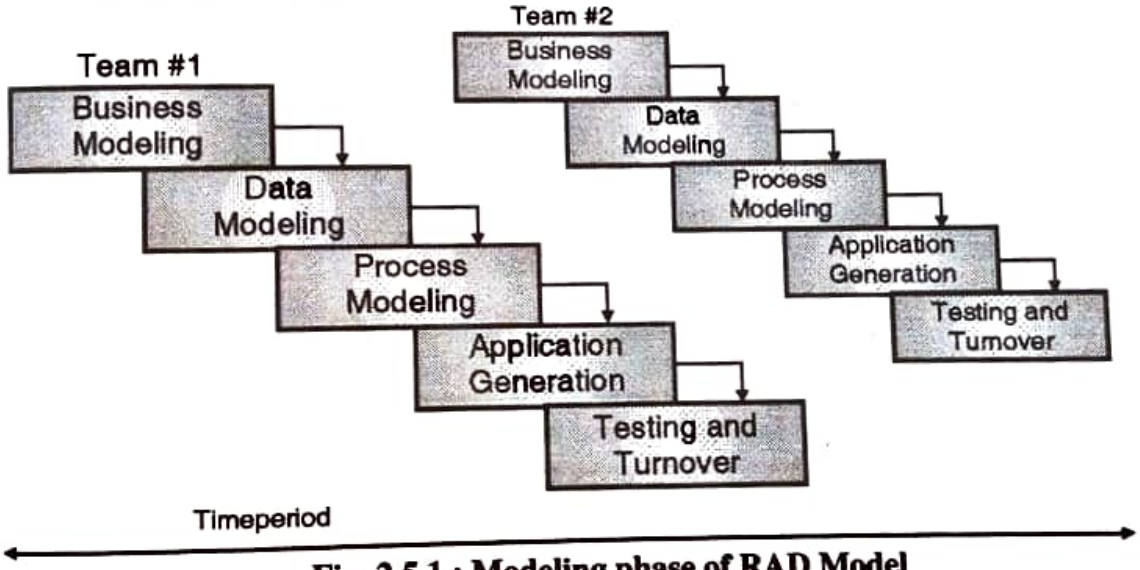


Fig. 2.5.1 : Modeling phase of RAD Model

   - Application Generation is about using a set of automated tools to facilitate the construction of the software say for example, the 4th GL techniques.
   - Testing and Turn over : Many of the components that will be used in the development of the proposed system might have already been tested since RAD focuses on reuse which reduces overall testing time. It is needed to concentrate on testing the new components.

   Once modeling activity of each team is over, it immediately starts construction phase.
8. In construction phase, each team develops the code of our product as well as performs testing. It reuses old software components and develops new functions which are called as automatic code generation.
9. Deployment collects each code from different team and clubs them into single project implement it and if required then perform iteration among the previous phases.

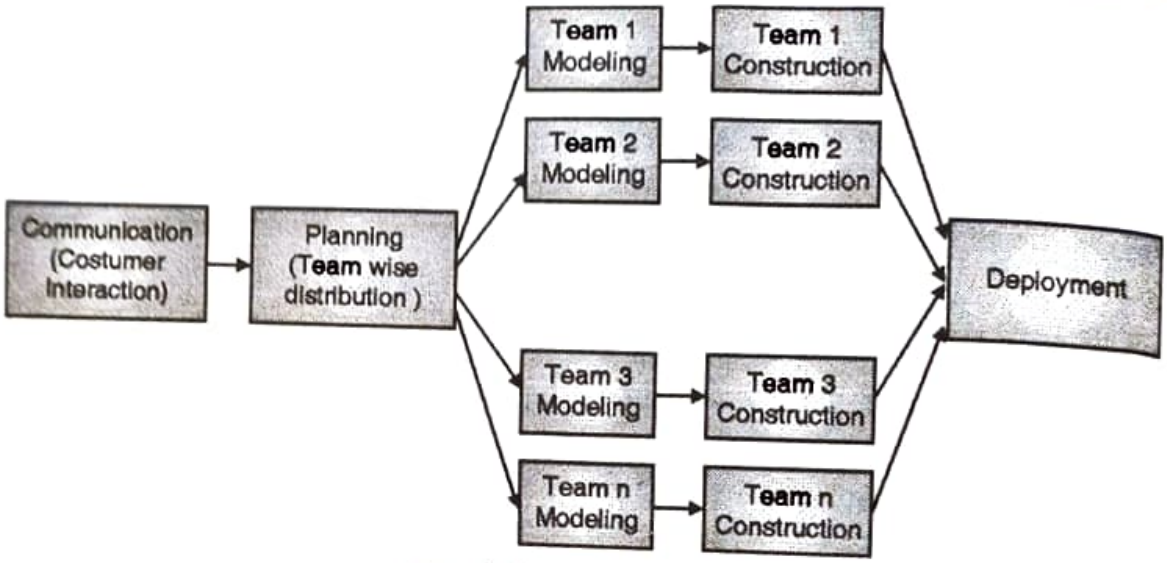


Fig. 2.5.2 : RAD model

Advantages of RAD model

1. It requires very less time to develop the product.
2. Planning and Design is performed before construction so it is not lengthy.
3. Product may be produced before its delivery date.

Disadvantages of RAD model

1. Large projects require sufficient or more human resources.
2. If developers and customers do not interact with each other then whole project may elapse.
3. If system is to be modularized properly then RAD is problematic.
4. If high performance is the Issue then RAD does not work.
5. It also doesn't work in high technical risk.

Syllabus Topic : Evolutionary Process Models

2.6 Evolutionary Process Models

The main drawbacks of Incremental process model are :

1. Overall design of the system is not so good.
2. It occupies large space in memory.
3. Some time speed of software is also slow.
4. Delivery date of the product is not decide.
5. Some times, problem is not identified properly.
6. Quality of software or product is poor.
7. Lack of interaction between developers and customers.
8. Poor modularization.
9. Doesn't produce high performance.
10. It is also not work in high technical risk.
11. Not suitable for complex product

These drawbacks are overcome in Evolutionary process model.

Main objectives of Evolutionary Process model are :

1. It should be suitable for complex system.
2. As time grows business also changes and product requirement may change during its development or construction phase.
3. It makes straight line path between customer requirement, business requirement and unrealistic product.

4. Evolutionary products should be iterative.

Though Evolutionary process model provides these advantages, it also has some issues.

Problems faced while using Evolutionary process models

1. Business competition makes product unrealistic
2. Tight market deadlines.
3. During development, product extension may be possible.
4. Software engineers must be flexible to requirements of customer and business changes.

2.6.1 Prototyping

This is the method or model of evolutionary process model. When customer tells his requirements, he is not aware about the development process, detailed input processing and also developer may be unsure of efficiency of the algorithm and adaptability of operating system, then prototype is the first step which is a best approach. This is explained in Fig. 2.6.1.

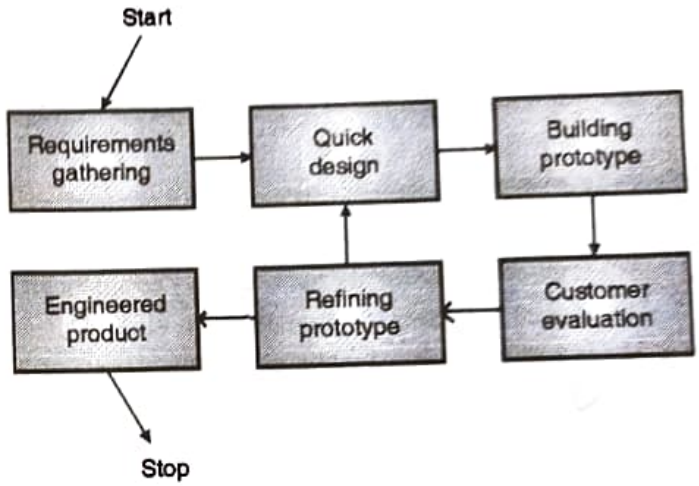


Fig. 2.6.1 : Prototyping model

Method

1. It begins with ***requirement gathering*** phase. Here customer and software engineer meet and define the overall objective of the product, It includes
   - Problem identification
   - Requirement analysis
   - Outline areas where further definition is mandatory

   According to overall objective of the product, software engineer makes planning of the design quickly.
2. ***Quick design*** focuses on the representation of human interface i.e. those aspects that are visible to the end user and customer. It uses different tools for design. Then, the analysts estimate a prototyping cost and inform about it to the management.
3. ***Building Prototype*** : Depending upon the quick design, construction of prototype is done that includes the software programs having the following capabilities :
   - accept input,
   - validate data,
   - perform calculations,
   - interact with files
   - produce outputs
   - Application generators

   Such a prototype can be built using several tools such as :

Screen generators :

- Input data screens with data validation are prepared.
- Meaningful prompt messages with each data entry
- Output screens are prepared.

- Table should have column headings and row heading.
- Labels, Messages, Colours, Fonts etc are decided.

Report generators :

It shows output according to users requirement where reports are made from records extracte from data base.

4. ***Customer Evaluation*** : The built prototype is then evaluated by the customers or end-users to fin if any changes are required. If so, then the requirements of the proposed software are refined.
5. ***Refining Prototype*** : The prototype is again refined based upon the feedback of the end-users abou what they need and what they don't expect. This process is repeated till both the end users and the developers feel that all the requirements are fulfilled in the software and thus there is no need o refinement.
6. ***Engineered Prototype*** : This completed product is then given to the customer as per the specification.

Advantages

- It is simple and an iterative process.
- It is revised to satisfy the needs of the customer.
- It does not require more cost to build.
- It can be prepared using pencil and paper, or computer software like screen generators, repor generators and application generators.
- It saves time of development.
- It develops the product through ongoing communication with the user.
- So easily finds user friendly services and non user friendly services.
- Readymade tools are used, to development of code.
- Product is to be delivered within proper time.
- It provides training to user.
- Prototypes are also used for testing.

Disadvantages

- It does not contain all features or perform all the necessary functions of the final system.
- Customer/user evaluates the module/prototype and they suggest addition and modifications so it may become lengthy.
- So may produce poor level quality product.
- Developer may compromise with operating system.
- It is not linear.
- It is not finding the risk of the project.

Q. State the difference between :Classic life cycle model and Prototyping model.

Table 2.6.1 : Comparison between Classic life cycle and Prototyping model

Sr. No.	Classic Life Cycle Model	Prototyping Model
1.	It is not an iterative process.	It is an evolutionary i.e. iterative process model
2.	During development, mostly product extension or addition of new requirements is not encouraged	During development, product extension may be possible.

Sr. No.	Classic Life Cycle Model	Prototyping Model
3.	Testing is not conducted from the initial phases of the SDLC – therefore when errors are detected after the coding phase it costs a lot for refining the product.	Building and refining of pro involved before the actual enginee process – therefore, much cost is not involved
4.	Included Manual coding	Uses readymade tools such as CASE tools for coding
5.	Communication with the users is done at the start i.e. while requirement gathering and then while testing phase. No ongoing communication with the users is encouraged	develops the product through ongoing communication with the user

2.6.2 The Spiral Model

Q. What does the radius imply in Spiral model ?

The Spiral model is also an evolutionary software process model that couples the iterative nature of prototyping. It is proposed by Boehm. The main objectives of spiral model are ;

- It provides controlled and systematic aspects of the linear sequential model.
- It uses potential of rapid development of incremental versions of the software.
- It finds all risks in the project.
- It finds future risks also.
- It is explained in following diagram.

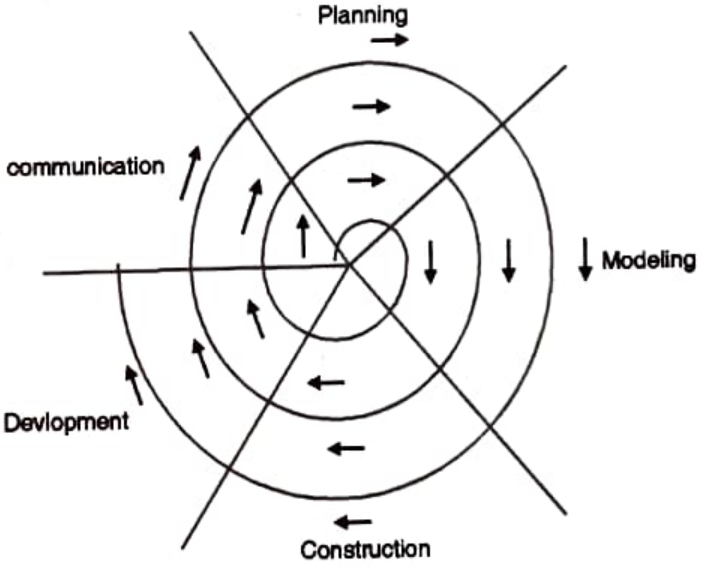


Fig. 2.6.2 : Basic spiral model

Method

Requirement Gathering includes;

- Interaction with customer and user.
- Problem identification regarding existing system.
- Deciding product specification.
- Deciding objectives of the system.

Planning decides the project plan. It includes

- Cost measurement of the system.
- Deciding schedule of the system.
- Collecting feedback from the user.
- Adjusted and planned number of iterations required to complete software.

Risk Analysis task is involved in each of the iterations of the project. It includes;

- Identification of risk areas
- Identification of technical as well as managerial risks
- Measuring cost related risk
- Efforts are taken to resolve the risk

Engineering phase immediately designs the prototype of the proposed system depending upon the objectives decided in the planning phase. It also converts design into coding and does the needful testing. It then shifts to *deployment phase* which includes :

- Implementing the developed system
- Do the Verification
- Gives training to users
- Collect feedback

Customer Evaluation : The product is then evaluated by the customers or end-users to find if any changes are required. If so, then the above phases are repeated again in loops.

Q. State the difference between : Waterfall Model and Spiral Model.

Table 2.6.2 : Comparison between Waterfall and Spiral Model

Sr. No.	Waterfall Model	Spiral Model
1.	Process flows from top to bottom like a flow of water from a hill to ground. Flow of water can't be reversed - similarly any new changes cannot be incorporated in the middle of the project development.	Best suitable for projects associated with risks.
2.	Process goes to the next phase only after the completion of the previous phase. Here, end user feedback is not taken into consideration for any change in SRS. This will result in restarting the work from beginning.	Each and every step goes through testing which makes it easy to recover any error and fix it then and there itself. In this model we don't have to start work from beginning.
3.	Purely a pre planned /strategic in nature.	Used to build a product which doesn't have adequate requirement gathering.
4.	Stresses more on requirement gathering.	focuses more on risk analysis which is not much considered in waterfall model.
5.	Customer feedback is not considered at every step of project development.	Customer feedback is considered at every step of project development.

Advantages

- It develops large scale systems and software.
- Better understanding of developers and customers.
- It performs risk analysis from the point of view of technical aspects as well as managerial aspects.
- It follows systematic and stepwise approach of SDLC.
- Wherever needed, it uses prototyping approach during development.

Disadvantages

- It is not suitable for fixed budget development.
- It is not panacea.

- It demands for expertise for risk assessment.
- If major risk is not covered then problem is never solved.

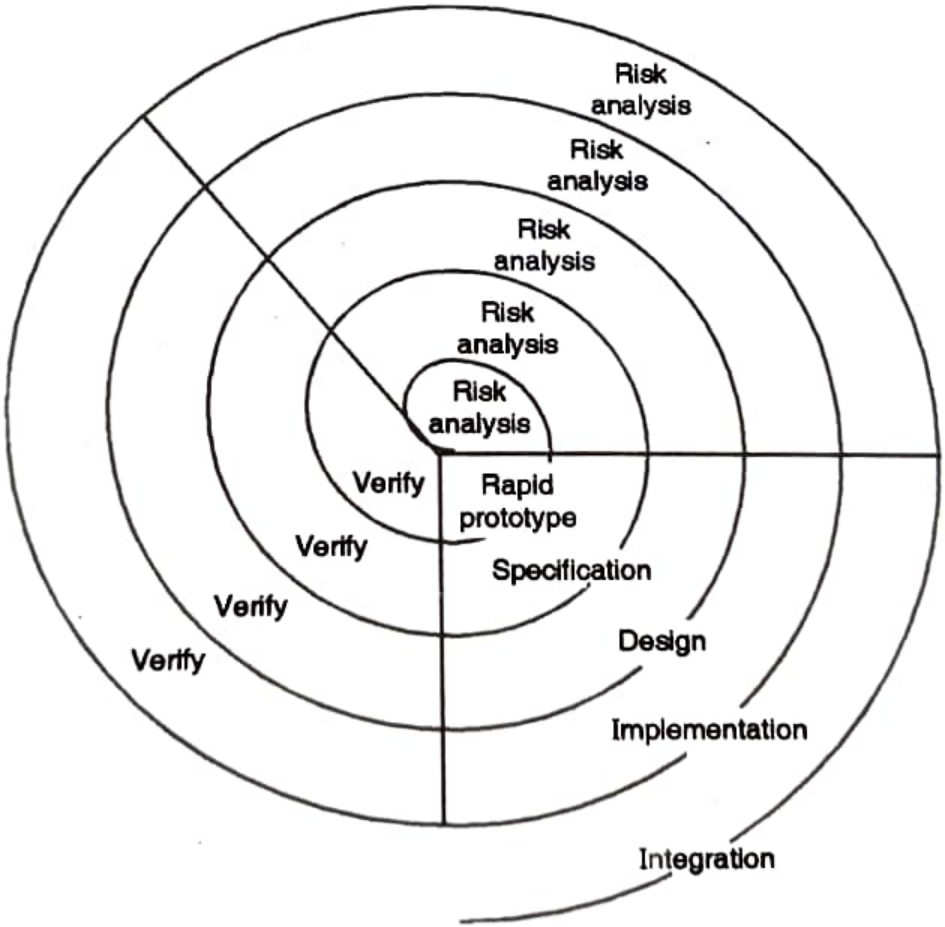


Fig. 2.6.3 : Spiral model used in industries

Now days, spiral view is to do risk analysis and prototyping that means to follow traditional framework then go for verification. If necessary goes for second iteration. This is well explained in Fig. 2.6.3.

2.10 Agile Development

2.10.1 Agility

Q. What is agility ?

Agility is the unending process, which always accepts specific requirement of the product from the customers or end user then it checks whether the requirements are growth oriented or not ? If it is growth oriented then it aggressively changes the existing system.

The Agile alliance [AG103] defines 12 principles to achieve agility

1. Customer satisfaction by continuous delivery of the software.
2. Always accept changes in requirement though it is late in development.
3. Deliver working software frequently in shorter time span.
4. Business people and developers must work together during complete development of the project.
5. Develop the project from the software project team. It includes;
   - Motivate each individual of the project.
   - Provide then necessary resources and environment to build the project.
   - Trust them to get the job done.
6. Do the **face to face conversation,** if essential or urgent.
7. Primary measure of the progress is the working software.

8. Agile process promotes sustainable development. (That means somebody from agile team doing good task to helpful for development should get promotions).
9. Keeps Continues track to technical excellence and best design increases agility.
10. It collects information about amount of work not done and tries to find out reasons of that.
11. Agile team collects requirement, and fulfil them using best architectures.
12. Continuous thinking about how the performance of team can be increased.

Agility can be applied to any software process.

Syllabus Topic : Agile Process

2.10.2 Agile Process

An Agile software process is characterized in a manner that addresses 3 key assumptions of software project.

1. It is difficult to predict which requirement is changing and which is not changing?
2. Some type of software, design and construction phase are related with each other, so again it is difficult to predict exactly how much time required to design process.
3. Time required for analysis, design and construction are not predictable.

This unpredictability can be managed by;

- Agile software development adapts incrementally.
- It accepts feedback from the customer.
- If requires then use prototype approach.

Agile Process models

- The main objective of agile process model is to provide best quality software in allotted time span.
- It follows conventional approach.
- It follows philosophy and guidelines suggested in manifesto for agile software development.
- There are many Agile Process models that are :
  - XP
  - ASD
  - DSDM
  - Scrum
  - Crystal
  - FDD
  - AM.

Syllabus Topic : Extreme Programming

2.10.3 Extreme Programming

Q. Explain XP in detail.

Extreme programming is most widely used Agile development model; it is also called as XP. It was popular in late 1980's. Idea of this extended programming is developed by the Kent Beck, Jeffries, Beck and Fowler. Extended programming is associated with ideas and methods.

- Extended programming (XP) uses all techniques of object oriented paradigm during the development process.
- XP follows an extreme approach to iterative development.
- New versions of software may be built several times per day and increments are delivered to customers roughly every two weeks.

- Incremental development is supported through small frequent releases of the system. Where requirements of these increments depend on customer stories and process planning.
- New build of the software is accepted only if all tests execute successfully.
- Change in a system is supported through development, through test first development, as well as through continuous integration.
- XP follows basic framework activities which consist of planning, designing, coding and testing.
- Each activity consists of set of rules and regulation.
- XP process is illustrated in Fig. 2.10.1.

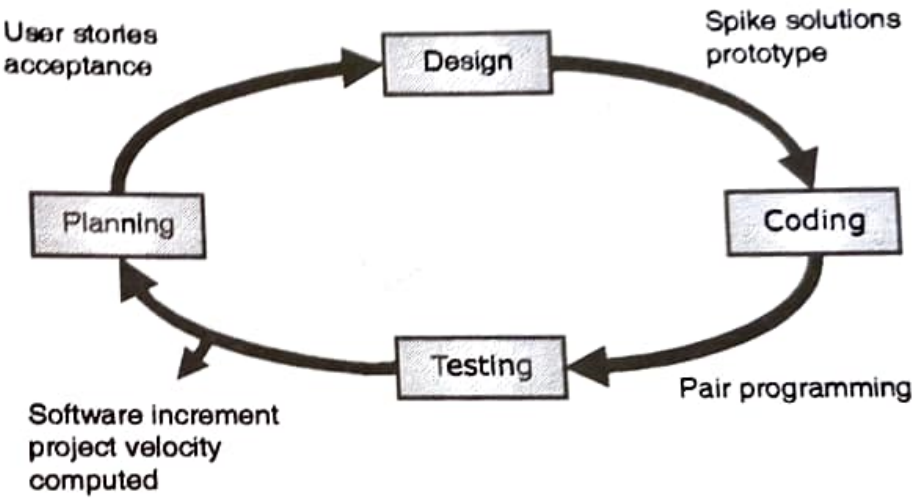


Fig. 2.10.1 : Basic XP process model

- Customer or representative of customer (i.e. may be end user) is involved in through out development process.
  - Customer or representative of customer and end users are the part of agile team so they should be active during the development process.
  - They should be active in defining all requirements of the project.
  - They should be active during the planning also.
  - They may passive during designing and coding.
  - But they are again active during testing if they are accepting the test results then project is to be implemented in actual office site for implementation and maintenance purpose.
- Coding supports pair programming which includes;
  - Collective ownership of the programs
  - Sustainable development of the program
  - It does not involve excessively long working hours.
- Maintaining Simplicity through
  - Simple designing that do not anticipate future enhancement.
  - Constant refactoring to improved quality of code.

Table 2.10.1 : Experts view regarding extreme programming

Experts View

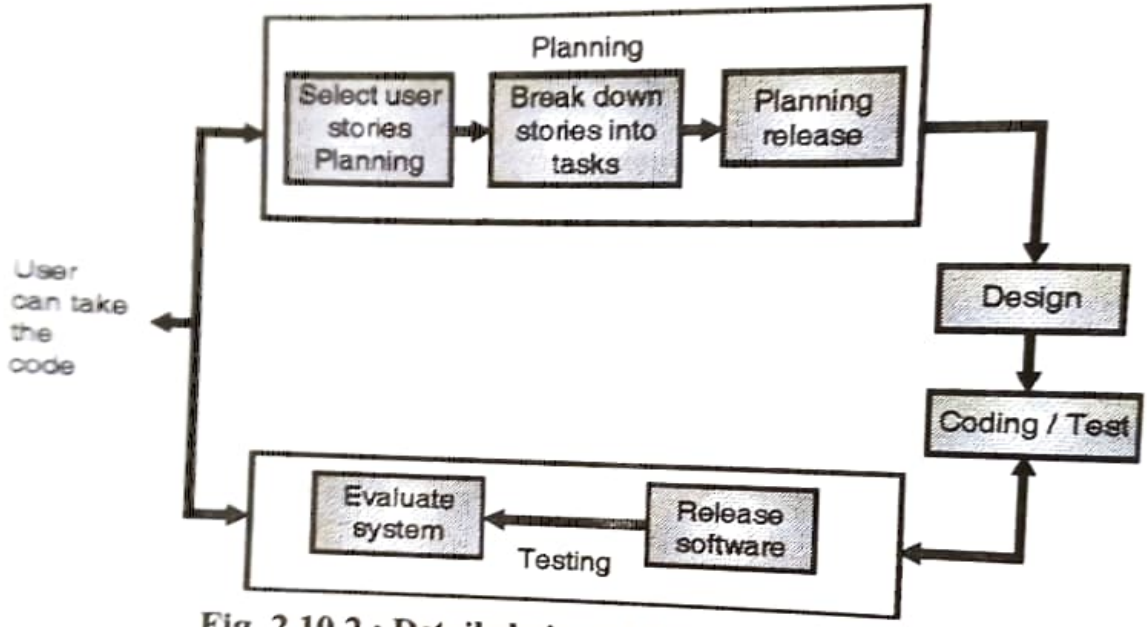


Fig. 2.10.2 : Detailed view of XP process model

I. Planning

Planning activity means gathering the information from customer and plans the project. It consists of :

- Planning activity is begins with creation of set of stories, which consist of features and functionality of the software to be built.
- Each story is written by the customer, which is placed on an index card or story card. Where each index card having;
  - A value assigned by customer which is nothing but priority.
  - Stories (i.e. features and functions) based on business.
  - The value of story may also depend on presence of another story.
  - Example of story card is given below :

Table 2.10.2 : Story card or index card

- This time span is measured in hours, days, or weeks.

- If story requires more than three development weeks then customer is asked to split story in to smaller stories and again priorities are assigned and costs are evaluated.
- New stories can be written at any time.
- Customers and XP team work together and to group stories to next increments of the software developed by the XP team.
- Once basic commitment or agreement (It includes delivery date with another project matter) is made up for a release, the XP team orders the stories that will be developed using three ways;
  1. All stories will be developed and implemented quickly using first in first out method.
  2. The story with highest value will be moved up in the schedule and implemented first.
  3. The riskiest stories will be moved up in the schedule and implemented first.
- First increment is called as a first project implementation. Similarly second iteration development is called as a second increment or second project implementation.
- After the delivery of first increment of the project XP team computes project velocity. Where project velocity can be find out as number of stories implemented during the first or one increment in some time span i.e. some few weeks.
- Project velocity is used for
  - Computing the delivery dates.
  - It finds whether over commitment is made or not? And if over commitment is occurs then immediately it again compute the exact delivery dates.
  - As development work proceeds, customer can add new stories.
  - Change the priorities of existing stories.
  - Split stories.
  - Delete stories.
- XP teams decides schedule of development according to project velocity and project priorities.

II. Design

XP design rigorously follows the KIS principles.

- KIS stands for Keep It Simple.
- Design gives guidelines for story writing i.e. nothing less, nothing more..
- The design of extra functioning is discouraged.
- These design guidelines should be followed in every software engineering method although there are times when sophisticated design notation and terminology may get in the way of simplicity.
- XP encourages the CRC (Class Representative Collaboration) cards.
- CRC identifies and organizes the object oriented classes that are relevant to the current software increment. That means it uses inheritance property of object oriented programming language.
- CRC card is nothing but to design work product produced.
- XP design phase converts each story card into CRC card.
- During the design of some story card it founds some difficulties. It follows operational prototyping approach to design complex part of that story card. This is called as a **spike solution**.
  - Spike solution is used to design complex part of the story card.
  - It uses operational prototype to design.
  - It is immediately implemented and evaluated from the end user.
  - So it reduces risk when true implementation is started.
  - It easily validates the original estimate.

- XP encourages the **refactoring**.
  - Refactoring means constructing such a technique which is also used to design new techniques.
  - It improves internal structure of the code.
  - It is disciplined way to clean up code.
  - This keeps code simple and maintainable.
  - Easy way to develop new code.
  - It minimizes bugs of the design as well as chance of introducing new bugs in the design.
  - Time required to create new design is very less.
  - Refactoring can be presented before, as well as after coding, that means continuous programming.

Table 2.10.3

Experts View
"Refactoring is the process of changing a software system in such a way that it does not alter the external behavior of the code yet improves the internal structure. It is disciplined way to clean up code [and modify/simplicity the internal design] that minimizes the chances of introducing bugs. In essence, when you refactor you are improving the design of the code after it has written." Fowler

- Some times, work product is other than CRC card or spike solution that means they can be developed using procedure oriented programming language.
- A central notion of XP is that - design occurs both before and after coding.
- Construction activity provides XP teams guidance on how to improve the design.

III. Coding

Once stories are converted into CRC cards, then XP team moves towards code development. It develops the code using following steps.

- The team exercises on each story and creates a series of unit test.
  - This approach is similar to knowing the previous exam questions before we begin the study.
  - It makes studying easier by focusing attention only on asked questions. Developer is better able to focus on what must be implemented to pass the unit test.
  - In this way, developer can construct a code, which is ready for unit test.
- Once code is complete, it can be unit tested immediately;
  - Test is taken by developer itself;
    1. It checks validity of the all input data.
    2. Whether code is producing expected result or not?
    3. Calls customer or representative of customer or end user to test the output of constructed code.
  - Feedback is collected.
- The key concept of coding activity is pair programming.

develop good software.

IV. Testing

Till now we know that there are two development strategy, Plan based development and Iterative development. Each development uses different testing techniques. Our Agile view process supports iterative development. XP spares more emphasis on than other Agile methods on the testing process.

- Testing procedure is central to XP where approach has been developed that reduces the same functions development in to the existing software.
- The XP Team assesses each story and breaks it into many tasks.

- Each task represents discrete features of the system.
- Unit test for each task are designed.
- Each task can generate one or more unit tests.
- Each unit test describes the test cases.
- The role of customer in the testing process is to help in developing acceptance tests for the stories that are to be implemented in the next release of the system.
- Acceptance testing is the process where the system is tested using customer data to check that it meets the customer's real needs.
- XP supports incremental testing.

☞ **Acceptance test of each story involves**

- Making several document selections.
- Paying for them in different ways.
- Printing them on different printers.
- Data validity test.
- Expect output tests.
- Series of different tests rather than single tests.
- Writing code for testing.

☞ **The Key advantages of testing in XP are:**

1. Test Driven Development (TDD)
2. Incremental test development from scenarios.
3. User involvement in test development and validation.
4. The use of automated tests.

System requirements fall into two categories

1. Functional 2. Nonfunctional

Table 3.1.1 : Difference between functional and nonfunctional requirements

Sr. No.	Functional Requirements	Nonfunctional Requirements
1.	Describes Product behavior	Describes Product's quality attributes.
2.	Describes what the product should do i.e. the actions/work/services of the software.	Describes capabilities of the product i.e. how the software should behave to meet the user needs.
3.	Describe the services that a system should provide	Describes the design and implementation constraints on the services and the external interface which a product must have.
4.	Describe 'what' the system should do in a particular condition.	Describe 'how' the system should work so as to be user friendly and secured.
5	Example : A bank software has functions – open acc., close acc, withdraw, loan claim etc.	Example : Performance, reliability, portability, security – system must run on windows server 2003, system must be secured against Trojan attacks.
6	The functions are represented using use case diagram & use case specifications.	The performance, usability, reliability, and security quality attributes are documented in narrative descriptions.

Syllabus Topic : Requirements Engineering

3.2 Requirements Engineering

Requirement Engineering is a bridge to design and construction of a quality software product.

- Requirement Engineering establishes an interaction between the developers, customers and users.
- The *input* to requirement engineering is : wishes, problems, unclear requirements etc. and the *output* of requirement engineering is the *Requirement Specification* and *Analysis model* that includes complete coverage of the problem and complete-exact definition of each requirement.
- Requirement Engineering is collecting the information (requirements) about what the system should do (not how to do it).

Requirement engineering includes following tasks so as to improve Software quality :

Task 1 : Inception : Defines the scope and nature of the problem to be solved.

Task 2 : Elicitation : Helps the customer to define what is required.

Task 3 : Elaboration : Customer's basic requirements are refined and modified.

Task 4 : Negotiation : As the customer defines the problems, negotiation occurs by highlighting the priorities, what is essential and what isn't, when it is required and many such.

Task 5 : Specification (SRS): Finally, the proposed requirements are specified. Every project needs requirements specification document because it is the formal agreement between the client/end-users, the business owner/stakeholder and the project manager. It states exactly what should and should not be included in a project and what the end-user expects from the proposed project.

Task 6 : Validation : Ensures that both customer's and engineer's understanding of the problem coincide.

Task 7 : Management : Helps the software engineers to control and track the changes in requirements at any time as the project proceeds.

- The above Requirements engineering tasks are important in software development projects as it influences the development cost, time, effort and quality.
- SRS includes the demands of stakeholders (customers, managers or end users). They specify the desired functions, quality attributes and other properties of the proposed software that is to be built or assembled.
- SRS helps software engineers to better understand the problem they will work to solve. SRS involves analyzing and accurately representing the client's requirements in a manner that can be effectively implemented in a system that will confirm to the client's specifications.
- But, in many situations, enough care is not taken in establishing correct requirements. This causes problems, later, in the development life cycle, and more time and money is spent in fixing these problems. Thus, it is necessary that requirements are established in a systematic way to ensure their accuracy and completeness. Requirement Analysts require both communication and technical skills.

Example : A Library Management System is to be designed so that information on books, CDs, DVDs, Journals, etc. can be stored and retrieved.

Requirements demanded by the client i.e. 'what system should do' are:

1. *Functional requirement*: Searching by Title, Author and ISDN should be possible.
2. *Implementation requirement*: User Interface should be web based i.e. accessible via web browser.

elicitation.

3.5 Software Requirement Specification

- Specification is concerned with documenting the requirements and this document is maintained over the life of the project. This is the most fundamental condition for successful implementation of the project.
- Specification is not limited to just text document, it can include graphical notations, mathematical specifications, user scenarios or prototypes.
- Specification includes a set of use cases that describe how the user interacts with the software. Use cases are also known as functional requirements. In addition to use cases, specification also includes nonfunctional requirements such as performance requirements and quality standards. (Example of requirements is given in Section 3.2).
- Specification is the complete description of the behavior of the system to be developed. It is the final work product produced by the requirements engineer which serves as the foundation for subsequent software engineering activities.

☞ **The basic issues that the SRS shall address are the following :**

(i) **Functionality :** behaviour or services provided by the software

(ii) **External interfaces :** external interactions of the software such as with end users, system's hardware or external software

(iii) **Performance :** the throughput, the availability, the response time, the recovery time of the software.

(iv) **Quality attributes :** portability, correctness, maintainability, security, etc.

(v) **Constraints :** implementation language, policies for database integrity, resource limits, operating environments etc.

Example : The ATM system is dependent on the network in the village areas. Due to flood and consequent network jams there will be a delay in the transactions

3.5.1 Benefits (Need) of SRS

- **SRS forms the basis for agreement between customers and development organization on what the software product is expected to do.** Complete descriptions of the functions that are

expected from the software are specified in the SRS. This will help the end users to verify whether the software meets the specified needs or not and if not, then how the software must be manipulated so as to meet their requirements.

- **SRS reduces the development effort**. The work of preparing SRS forces various stakeholders i.e. various concerned groups in the customer's organization to think thoroughly of all their requirements before the project design begins, thus reducing the efforts needed for redesigning, recoding, and retesting. Careful inspection of the requirements specified in SRS can reveal the left-outs, misinterpretations and inconsistencies early in the development cycle; hence it becomes easier to correct these problems.
- **SRS forms a base for cost estimation and project scheduling**. As complete description of the product to be developed is specified in the SRS, it helps in estimating the project costs and can be used to obtain approval from the customer for the decided price estimates.
- **SRS provides a baseline for verification and validation**. Software development team can develop their verification and validation plans or say, test plans much more effectively from a well-prepared SRS document. As SRS provides a baseline against which requirement confirmation can be measured.
- **SRS facilitates transfer of new software to new environments**. SRS makes it easier to transfer the software product to new clients or new hardware machines. Therefore, developers feel easier to transfer the software to new clients and customers feel easier to transfer the software on other systems of their organization.
- **SRS serves as a basis for software improvement**. SRS describes the product features and not the process of project development; therefore, it serves as a basis for enhancements by allowing the developers to do any later modification if required on the finished product. But then, SRS needs to be altered in that case i.e. SRS forms a base for continuous product evaluation.

Syllabus Topic : Characteristics of SRS

3.5.2 Characteristics of SRS

Great SRS leads to Great Product: We have to keep in mind that the goal is to create great products and great software. A great product can be created only from a great specification. Systems and software these days are so complex that to get on with the design before knowing what you are going to build is foolish and risky.

A great SRS has following quality factors :

- **Correct :** A set of requirements is correct if and only if every requirement stated therein represents something required of the system to be built." Davis (1993)
- **Unambiguous :** The reader of a requirement statement should be able to draw only one interpretation of it.
- **Incorrect Example** – "The system should not accept password longer than 8 characters"

The above stated requirement can have multiple interpretation as given below :

- The system should not allow user to enter more than 8 characters in the password field
- The system should truncate the entered data to 8 characters
- The system should display an error message if user enters more than 8 characters in the password field

Correct Example – "The system should not accept passwords longer than 8 characters in the password field. If the user enters more than 8 characters while entering the password, an error message should prompt the user to correct the same."

Complete - No requirements should be missing. A complete requirement leaves no room for guessing. Include all significant requirements, whether related to functionality, performance, design constraints, attributes, or external interfaces

Consistent - A consistent requirement does not conflict with other requirements in the requirement specification.

Incorrect Example :

REQ1 – "The birth date should be entered in mm/dd/yyyy format"

REQ2 - "The birth date should be entered in dd/mm/yyyy format"

Correct Example

REQ1 – "For the US users the birth date should be entered in mm/dd/yyyy format"

REQ2 – "For the French users the birth date should be entered in mm/dd/yyyy format"

- ***Ranked for Importance*** – Requirements should be achievable. But sometimes, few requirements are not attainable.
- ***Verifiable*** - Don't put in requirements like – "It should provide the user a fast response" or "The system should never crash". Instead, provide a quantitative requirement like: "Every key stroke should provide a user response within 100 milliseconds."
- ***Traceable*** - Each requirement should be expressed only once and should not overlap with another requirement. Every requirement has a unique identification number so that requirements can be easily traced through out the specification.
- **Adaptability :** Can the requirement be changed without a large impact on other requirements ?

Syllabus Topic : Components of SRS

3.5.3 Components of SRS

Following is the IEEE standards SRS format which depicts the components of a SRS document.

1. **INTRODUCTION**
   - 1.1 PRODUCT OVERVIEW
     (Product description).
   - 1.2 PURPOSE
     (Product description)
   - 1.3 SCOPE
     Write the benefits, objectives, and goals.
   - 1.4 AUDIENCE
     (lists the users of SRS such as developers, project managers, marketing staff, users, testers, and documentation writers.).
   - 1.5 DEFINITIONS, ACRONYMS AND ABBREVIATIONS
     (defines all the terms necessary to properly interpret the SRS).

1.6 CONVENTIONS

(describes the standard conventions followed while writing this SRS such as fonts or special significant highlights) .

1.7 REFERENCES

(lists the reference documents or web page address used as reference)

2. OVERALL DESCRIPTION

2.1 PRODUCT PERSPECTIVE

(lists the reference documents or web page address used as reference)

2.2 PRODUCT FUNCTIONALITY

(designs such as DFDs that describe major functions of the system).

2.3 USER CHARACTERISTICS

Identify various users who will be using this product. Differentiate them based on frequency of use, subset of product functions used, technical expertise, security or privilege levels, educational level, or experience.

2.4 OPERATING ENVIRONMENT

(Designs that describe that shows the major components of the overall system, subsystem interconnections, and external interface.).

2.5 DESIGN AND IMPLEMENTATION CONSTRAINTS

Describe the issues of the developers such as - hardware limitations, interfaces to other applications, specific technologies, tools, and databases to be used, language requirements, communications protocols, security considerations, design conventions.

2.6 USER DOCUMENTATION

List out the user manuals, on-line help, and tutorials that will be delivered along with the software.

2.7 ASSUMPTIONS AND DEPENDENCIES

List any assumed factors that could affect the requirements stated in the SRS. These include third-party components or issues around the development or operating environment. The project could be affected if these assumptions are incorrect, are not shared, or change.

Also list out the dependencies such as software components that you plan to reuse from another project.

3. SPECIFIC REQUIREMENTS

3.1 EXTERNAL INTERFACE REQUIREMENTS

3.1.1 USER INTERFACES

List out the different screens that will be available to the user.

3.1.2 HARDWARE INTERFACES

List out any special libraries that are used to communicate with your software mention them.

3.1.3 SOFTWARE INTERFACES

(Describes databases, operating systems and tools, services and the data that will be shared across software components)

3.1.4 COMMUNICATIONS REQUIREMENTS

(Describes the communication related requirements such as e-mail, web browser, network server communications protocols, electronic forms and the communication standards such as FTP or HTTP.)

3.2 FUNCTIONAL REQUIREMENTS

(describes the functional requirements in detail with proper explanations regarding each and every function.)

3.3 BEHAVIOUR REQUIREMENTS

3.3.1 USE CASE VIEW

Draw a use case diagram that will illustrate the entire system and all possible actors.

4. OTHER NON FUNCTIONAL REQUIREMENTS

4.1 RELIABILITY

4.2 AVAILABILITY

4.3 SAFETY and SECURITY

(describes the functional requirements in detail with proper explanations regarding each and every function.)

4.4 MAINTAINABILITY

4.5 PORTABILITY

4.6 PERFORMANCE

Provide at least 5 different performance requirement such as -"Any transaction will not take more than 10 seconds".

5. SUPPORTING INFORMATION

This section is Optional. Define any other information/requirements not stated elsewhere in the SRS. This might include internationalization requirements, legal requirements, reuse objectives for the project, and so on.

UNIT I

CHAPTER 4

Object-oriented Design using UML

Syllabus

Class diagram, Object diagram, Use case diagram, Sequence diagram, Collaboration diagram, State chart diagram, Activity diagram, Component diagram, Deployment diagram

4.1 Introduction

- Unified Modeling Language (UML) begins by constructing a model which is a simplification of reality.
- Model is an abstraction of some underlying problem.
- Model is a complete description of a system from any particular perspective constructed to understand the system before building the new or before modifying the existing system.

Static and Dynamic Models

- A *static model* is the snapshot of system's parameters at rest or at a specific point of time. For example, a customer could have more than one account at a time. Static models have stability and there is no change of data in future. The UML *class diagram* is a type of a static model.
- A *dynamic model* is a collection of behaviours of the system. It represents how objects interact with each other in order to perform any task. The UML *interactive* (*sequence and collaborative*) and *activity diagrams* are types of a dynamic model.

Benefits of a Model

1. Clarity

Models make it easier to. We can very easily express complex ideas using models as it allows visual examination of the whole system possible.

2. Complexity

Models reduce complexity by separating unimportant aspects from those that are important.

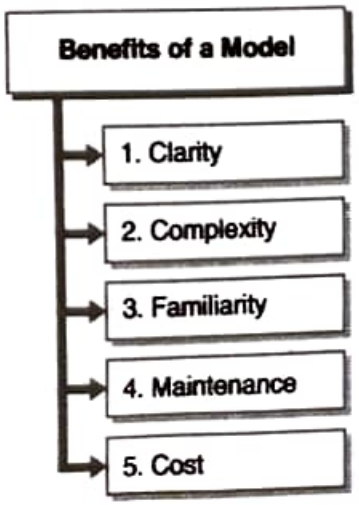


Fig. C4.1 : Benefits of a Model

➜ **3. Familiarity**

Models are the representation of how the information is actually represented so that you may understand the system and feel more comfortable to work.

➜ **4. Maintenance**

Visual identification and confirmation about the locations to be changed will reduce errors. Also, manipulation (changing) of a model is more easier than manipulating a real system.

➜ **5. Cost :**

The cost of building and modifying a model is much lower as compared to the same task performed on a real system.

4.2 UML

- UML is a set of notations and conventions used to model an application to describe system's parameters and behaviours..
- The UML is an object-oriented methodology and technique because it is generally applicable to object-oriented problem solving.
- UML is not defined as 'data modelling' technique, but as an "object modelling" technique. Instead of entities, it models "object classes".
- Because of confluence in ideas, techniques, personalities, and politics; UML became a standard notation for representing the structure of data in the object-oriented community. It was developed when the three great personalities of the object-oriented community - James Rumbaugh, Grady Booch, and Ivar Jacobson agreed to adopt a standard notation for UML.
- UML was originally developed at Rational Software but is now administered by Object Management Group (OMG) i.e. responsible for creating and maintaining the UML language specifications.

OMG defines UML as, "a graphical language" for :

- **Visualizing :** There are some aspects of software system that you can not understand unless you build models.
- **Specifying :** models are precise, unambiguous, and complete.
- **Constructing :** allows direct connection to a variety of programming languages.
- **Documenting :** artifacts of a software system.

4.2.1 Features of UML

➜ **1. Syntax**

UML is just a language that tells what model elements and diagrams are available and the rules associated with them. It doesn't tell you what diagrams to create.

➜ **2. Comprehensive**

It can be used to model anything that may fill the user's requirement.

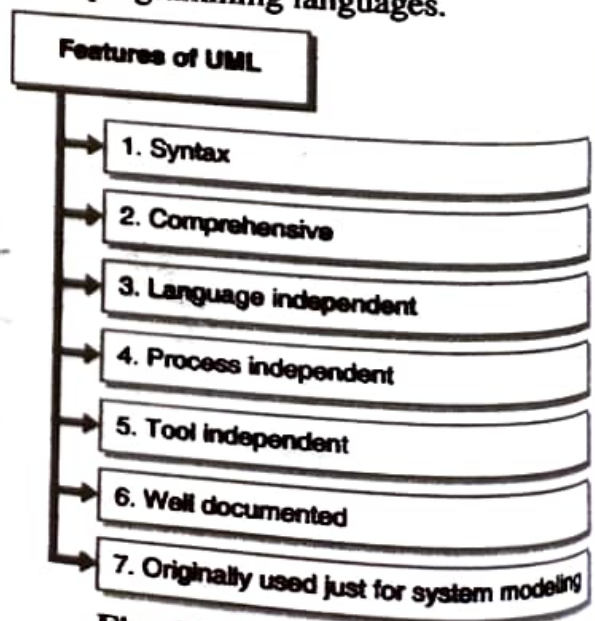


Fig. C4.2 : Benefits of a Model

➜ **3. Language independent**

It does not matter what high-level language is to be used in the code. Mapping into code is the responsibility of the case tool that you use.

➜ **4. Process independent**

The process by which the models are created is separate from the definition of the language.

➜ **5. Tool independent**

UML gives freedom for tool vendors to be creative and add value to visual modeling with UML.

➜ **6. Well documented**

The UML notation guide is available as a reference to all the syntax available in UML language.

➜ **7. Originally used just for system modeling**

Some user defined extensions are becoming more widely used now, for example for business modeling and web-based modeling.

4.2.2 Need of UML

- **Software construction needs a plan :** Software development is a very complex job and once a particular structure is in place, it is very difficult job to make any changes. UML helps in constructing a visual model of the real system.

  Example :A building architect creates a drawing of the building from more than one direction, ensuring that the structure and dimensions of the building are appropriate. Only when this visual model of the proposed work is approved, then the real construction of the building i.e. laying of bricks is begun.

- **Visualize in multiple dimensions and levels of detail :** As in the above example, UML, allows software to be visualized from multiple dimensions so that a computer system can be completely understood before construction begins.
- UML is also used to produce several models at increasing levels of detail. These increasing levels of detail can be added to each part of the software as it is constructed until final stage.
- **Appropriate for both new and legacy systems :** UML can be applicable for both new system developments and improvements in existing systems where only those parts that need modifications are modeled.
- **Documents software assets :** Undocumented or poorly documented software has a very low value as a company asset. Documentation saves the company from losses occurred by depending on key personnel who may leave at any time and thus improves understanding of the company's critical business processes.
- **Unified and universal :** OMG has made UML as a de-facto (actual) standard modeling language in the software industry. UML is called as a universal language because it can be applied in many areas of software development.
- **Inherent traceability :** The Use Case driven approach of UML modeling ensures that all levels of model trace back to elements of original functional requirements.
- **Incremental development and re-development :** UML models can be applicable in incremental development environment. It is not only possible to develop only those parts of the model that are required to satisfy the new requirements, but also possible to demonstrate that the code needed to fulfil these requirements is in place.
- **Parallel development of large systems :** Large complex UML models can be decomposed into simple models that can be developed independently by different people or groups simultaneously at the same time.

4.2.3 UML Advantages

- UML modelling is effective for large, complex software systems and also for modeling middleware.
- It is easy to learn and provides advanced features for expert analysts, designers and architects.
- It can specify the system in an implementation-independent manner.
- It allows re-usability i.e. 10-20% of the constructs are used 80-90% of the times.
- Structural UML modeling represents a skeleton that can be refined and extended with additional structure and behavior.
- Use case modeling specifies the functional requirements of the system in an object-oriented manner.
- Allows modelling of any type of application m running on any type of combination of hardware, operating system, programming language, and network.
- Increases efficiency and thus reduces cost and time-to-market.
- UML Profiles (i.e. subsets of UML designed for specific purposes) help to model Transactional, Real-time, and Fault-Tolerant systems in a natural way.

Syllabus Topic : Use-case Diagrams

4.3 Use-case Diagrams

- Use case diagrams describe what a system does from an external observer's viewpoint. The focus is on **what** a system does rather than **how**.
- Use case diagrams are based on scenarios. A ***scenario*** is an example of ***what*** happens when someone interacts with the system i.e. it describes the course of events that may take place in the system.
- A Use Case Diagram shows relationships between ***actors, use cases*** and their ***associations*** (also called as interactions or communications) in a ***system***.
- Actors are represented using *stick figures*, Use cases by *ovals* and Communications by *lines* that link actors to use cases.
- A use case diagram is a visual representation of different *scenarios* showing the *association* between an *actor* (primary elements/business roles) and a *use case* (business processes that forms the system).

Example : A use case scenario for a medical clinic:

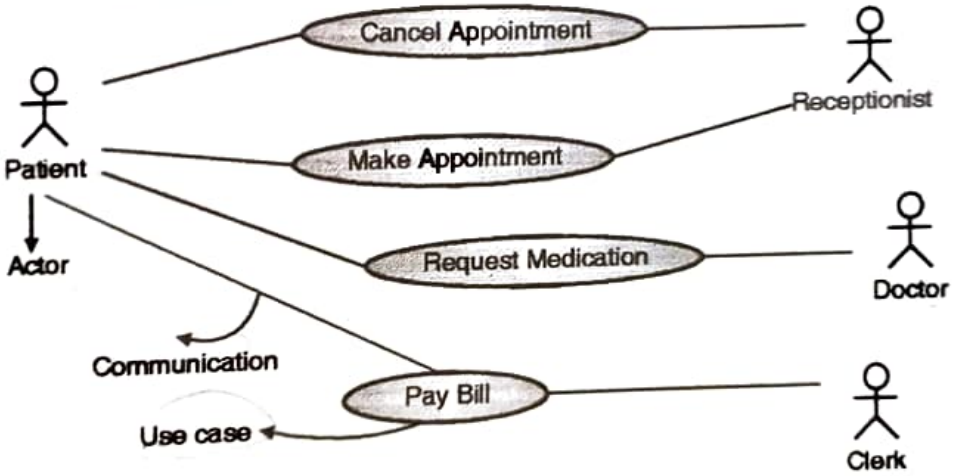


Fig. 4.3.1 : Use case diagram for a medical clinic

A patient calls-up in a clinic to get an appointment for monthly check-up. The receptionist finds out the nearest empty time slot in the appointment book and schedules the appointment for that time slot.

4.3.1 Need of Use Case Diagrams

1. **Determines requirements :** New use cases often generate new requirements as the system is analyzed and so on, the design takes shape. It is helpful in project planning, documentation and development of system's requirements
2. **Communicates with clients :** Notations in use case diagrams help developers to easily communicate and discuss with clients.
3. **Generates test cases :** Collection of scenarios in a use case leads to set of test cases for those particular scenarios.
4. **Represents complete functionalities :** Use cases represent complete functionality of the system or say, business process rules.
5. Discovers classes and relationships among subsystems of the system.

4.3.2 Elements of a Use Case Diagram

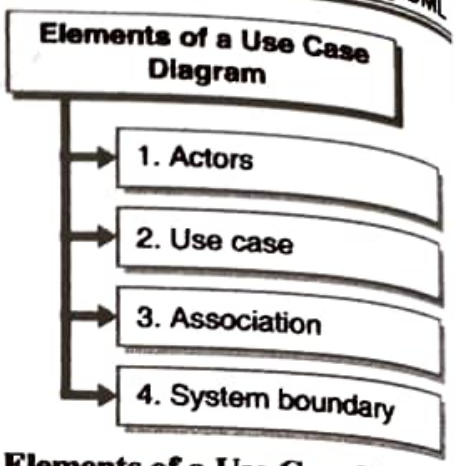


Fig. C4.3 : Elements of a Use Case Diagram

Use case diagram constitutes of following elements :

➜ **1. Actors**

- An actor in a use case diagram is any entity that performs certain roles in a given system. These may be the business roles of users who perform certain functionalities in a given system. If some entity does not make any affect on a certain piece of functionality then, it makes no sense to represent it as an actor.
- An ***actor*** is who or what initiates the events involved in that task.
- Identify an actor in the proposed system by searching for business terms that represent certain roles.
- An actor is shown as a stick figure in a use case diagram portrayed outside the system boundary.

Example : In the statement "students visit the library to borrow books. Librarian updates the catalogue" - *student* and *librarian* are the roles that are identified as actors in the system.

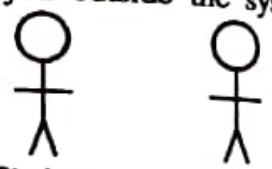


Fig. 4.3.2 : Actors in a Library Use-case diagram

➜ **2. Use case**

- A use case in a use case diagram is a visual representation of distinct business functionality in a given system.
- Identify the use cases by listing the unique business functions in your problem statement. Each of these business functions can be classified as a potential use case.
- A use case is shown as an ellipse in a use case diagram.

Example : In the same above statement about library, there are two functions - "Borrow book" and "Update catalog" that are identified as use-cases in the system.

Fig. 4.3.3 : Use cases in a Library Use case diagram

➜ **3. Association**

- An association is an interaction between an actor and a use case.
- Associations between actors and use cases are indicated in use case diagrams by solid lines.
- Associations are shown by lines connecting use cases and actors to one another, with an arrowhead (arrowheads are very rarely used) on one end of the line. The arrowhead is used only when you need to indicate the direction of initial invocation or to indicate the primary actor within the use case.

Example : In the same above statement about library, there is an association between actor 'student' and use-case 'borrow book'.

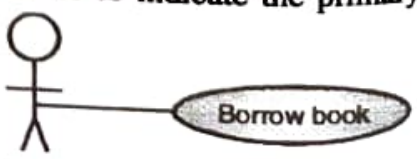


Fig. 4.3.4 : Example of an Association in a Library use case diagram

➜ **4. System boundary**

- A system boundary of a use case diagram defines the limits of the system.

- A system boundary defines the scope of what a system will be. A system cannot have infinite use cases.
- The system boundary is shown as a rectangle spanning all the use cases in the system.
- System boundary of a system encloses the use cases of the system inside a rectangle and actors of the system outside the rectangle.

Example : In the same above statement about library, there is an association between actor 'student' and use-case 'borrow book'.

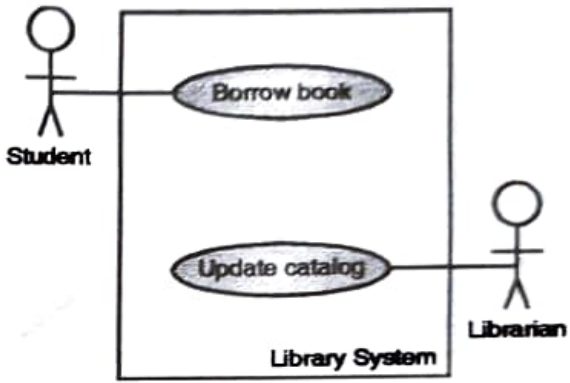


Fig. 4.8.9 : State diagram of phone line showing

Syllabus Topic : Activity Diagram

4.9 Activity Diagram

- Activity diagram displays the sequence of activities.
- Activity diagrams show the workflow from a start point to the finish point detailing the number of decision paths that exist in the progression of events contained in the activity.
- They also describe situations, where parallel processing may occur in the execution of some activities.

☛ Use of Activity diagram

- Describes Business rules
- Describes Complex series of multiple use cases
- Describes the processes related to decision points and alternate flows
- Describes operations that take place in parallel
- Describes software flows and logic control structures

☛ Designing an Activity diagram

➜ 1. Initial node

The filled in circle is the starting point of the diagram. This indicates the beginning of the sequence of activities. An initial node makes it easier to read the diagram. ●

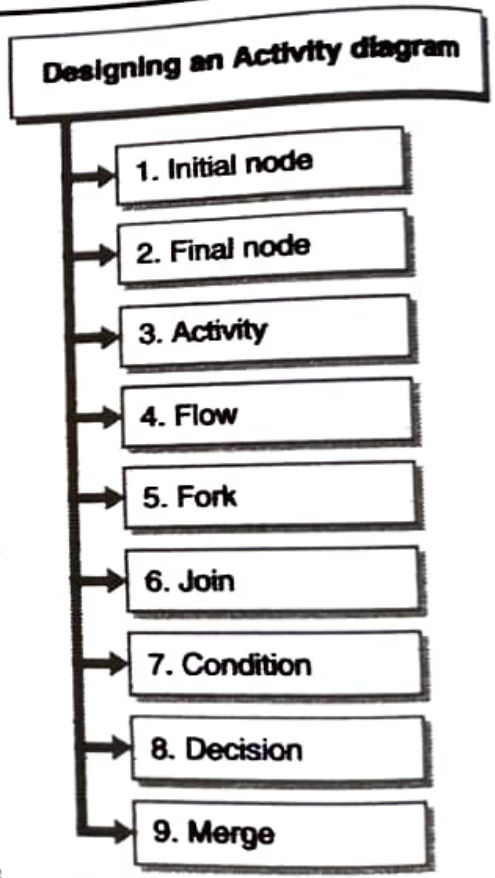


Fig. C4.7 : Designing an Activity diagram

2. Final node

The filled circle with a border is the ending point. An activity diagram can have zero or more activity final nodes. The final state ends the sequence of activities.

3. Activity

The rounded rectangles represent activities that occur. An activity may be physical, such as *Inspect Forms*, or electronic, such as *Display Student Screen*. It describes the action state of the system.

4. Flow

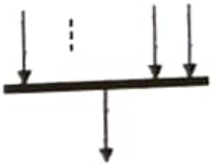
The solid arrows in the diagram represent the flow of activity. The outgoing arrow attached to an activity symbol indicates the transition i.e. triggered by the completion of the activity.

5. Fork

A thick black bar with one flow entering into it and several leaving it. This denotes the *beginning of parallel activity*. Forks have *One Entry Transition*.

6. Join

A thick black bar with several flows entering it and one leaving it. All flows going into the join must reach it before processing may continue. This denotes the *end of parallel activity*. Joins have *One Exit Transition*. Is exactly opposite to 'Fork'.

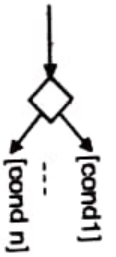


7. Condition

Text such as *[Incorrectpassword]* on a flow, defining a guard which must evaluate to true in order to traverse the node.

8. Decision

A *diamond with one flow entering and several leaving*. The flows leaving include conditions although some modelers will not indicate the conditions if it is obvious.



9 Merge

A *diamond with several flows entering and one leaving*. The implication is that one or more incoming flows must reach this point until processing continues, based on any guards on the outgoing flow. Is exactly opposite to 'Decision'.

Example : Showing Decision, Condition and Merge.

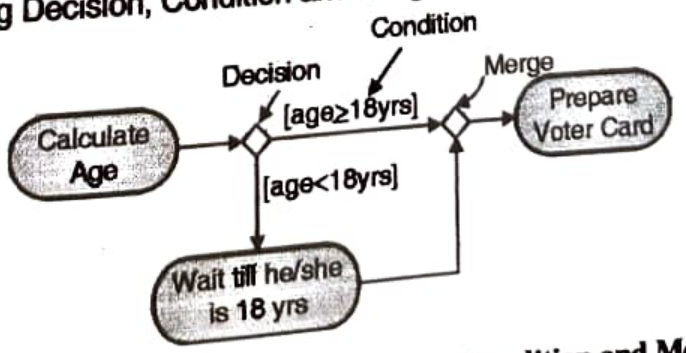


Fig. 4.9.1 : Example showing Decision, Condition and Merge

Partition (swimlanes) : Objects are generally organized into partitions, also called swimlanes, indicating who/what is performing the activities.

5.1 Introduction

This chapter focuses on the design phase of SDLC. After requirement gathering and analysis phase, the next step is to ***design the analysis model***.

☞ System

System is an interrelated set of components, with identifiable boundaries working together for achieving some goal.

☞ A system has nine characteristics

1. **Components:** These are the indecomposable or irreducible parts that make up a system which are also called as subsystems.
2. **Inter-related Components:** It is about the dependency between the subsystems i.e. dependency of one subsystem on one or more subsystems.
3. **A boundary:** It is the line that marks the inside and outside of a system and that sets off the system from its environment.
4. **A purpose:** It is the overall goal or function of a system.
5. **An environment:** It encompasses all the external entities of a system that interact with the system.
6. **Interfaces:** Interaction between the system and its environment or interaction between various subsystems.
7. **Input :** Whatever a system takes from its environment in order to fulfil its purpose.
8. **Output:** Whatever a system returns to its environment in order to fulfil its purpose.
9. **Constraints:** The limit to which a system can accomplish or the restrictions on developing the system.

Syllabus Topic : System/Software Design

5.2 System Design

Systems analysis : Process of studying an existing system to determine how it works and how it meets user needs.

Systems design : Process of designing a plan for an improved system based upon the results of the existing system analysis.

Need of System Design

- It is an Effective method to solve problems of existing system as it designs the new plan based upon the study of existing system analysis.
- It differentiates logical view and physical view of the software to be built up.
- Designing also partitions the requirements such that it becomes easy to build up a model and develop its documentation.
- It keeps track of all interfaces and designs them properly.
- Whenever possible, graphics are used to design the analysis model.

Main Objectives of Design

- Practicality – Efficiency – Cost – Flexibility – Security

Analysis & Design Model

- Analysis modelling uses diagrammatic form and text to describe requirements of :
  - Data o Functions o Behaviour of o The software to be built
- It should have the following features :
  - Easy to understand. o Straightforward to review. o Correctness.
  - Completeness. o Consistency.

Who does it ?

- Software engineers or – System analysts or – Project manager or – Modeller

Importance

- Analysis model is multidimensional.
- Design phase completely depends on analysis model.
- If some deficiency remains in the analysis models then errors will be found in product to be built.

Ensures

- It must be review for correctness, completeness and consistency.
- It should ensure that it accomplishes all needs of stakeholders.
- It establishes a foundation from which design can be conducted.

5.2.1 Thumb Rule

Arlow and Neustadt produce rules of thumbs for the analysis model. These rules are given below:

Syllabus Topic : System/Software Design

5.2 System Design

Systems analysis : Process of studying an existing system to determine how it works and how it meets user needs.

Systems design : Process of designing a plan for an improved system based upon the results of the existing system analysis.

Need of System Design

- It is an Effective method to solve problems of existing system as it designs the new plan based upon the study of existing system analysis.
- It differentiates logical view and physical view of the software to be built up.
- Designing also partitions the requirements such that it becomes easy to build up a model and develop its documentation.
- It keeps track of all interfaces and designs them properly.
- Whenever possible, graphics are used to design the analysis model.

Main Objectives of Design

- Practicality – Efficiency – Cost – Flexibility – Security

Analysis & Design Model

- Analysis modelling uses diagrammatic form and text to describe requirements of :
  - Data
  - Functions
  - Behaviour of
  - The software to be built
- It should have the following features :
  - Easy to understand.
  - Straightforward to review.
  - Correctness.
  - Completeness.
  - Consistency.

Who does it ?

- Software engineers or – System analysts or – Project manager or – Modeller

Importance

- Analysis model is multidimensional.
- Design phase completely depends on analysis model.
- If some deficiency remains in the analysis models then errors will be found in product to be built.

Ensures

- It must be review for correctness, completeness and consistency.
- It should ensure that it accomplishes all needs of stakeholders.
- It establishes a foundation from which design can be conducted.

5.2.1 Thumb Rule

Arlow and Neustadt produce rules of thumbs for the analysis model. These rules are given below:

1. Model should focus on requirement visible in business domain.
2. Each element of the analysis model should add to all requirements and provides insight into information domain function and behaviour of the system.
3. Some functional and non functional models are required to design the software.
4. Minimize coupling throughout the system. It is important to represent relationship between classes and functions. That means functions are interconnected to each other very tightly then efforts are put to reduce this interconnectedness.
5. It gives assurance that analysis model provides value to all stack holders.
6. It keeps model as simple as it can be.

5.2.2 Design Modeling Principles

Design model provides a variety of different views of the system just like architecture's plan for a house. Different methods like data driven, pattern driven or object oriented methods are used for constructing the design model. And all these methods use a set of design principles for designing a model.

1. **Design must be traceable to the analysis model** Analysis model represents the information, functions and behaviour of the system. Design model translates all these things into architecture: a set of sub systems that implement major functions and a set of component level designs that are the realization of analysis classes. This implies that design model must be traceable to the analysis model.
2. **Always consider architecture of the system to be built** Software architecture is the skeleton of the system to be built. It affects interfaces, data structures, program control flow and behaviour, the manner in which testing is conducted, maintainability of the resultant system and much more.
3. **Focus on the design of data :** Data design encompasses the manner in which the data objects are realized within the design. It helps to simplify the program flow, makes the design and implementation of software components easier and makes overall processing more efficient.
4. **Interfaces (both user and internal) must be designed :** The data flow between the components decides the processing efficiency, error flow and design simplicity. A well designed interface makes integration easier and tester can validate the component functions more easily.
5. **User interface should consider the user first :** The user interface is the main thing of any software. No matter how good its internal functions are or how well designed its architecture is but if the user interface is poor and end users don't feel ease to handle the software then it leads to the opinion that the software is 'bad'.
6. **Component level design should exhibit functional independence :** It means that the functions delivered by a component should be cohesive i.e. it should focus on one and only one function or sub function.
7. **Components should be loosely coupled :** Coupling (Union or Combination) of different components into one is done in many ways – via a component interface, by messaging or through global data. As the level of coupling increases, error propagation also increases and overall maintainability of the software decreases. Thus, component coupling should be kept as low as possible.
8. **Design representations should be easily understood :** The design model helps engineers to generate code, testers to test software, and maintain the software easily in the future. If the design is difficult to understand, it will not serve as an effective communication medium.
9. **The design model should be developed iteratively :** Designs are done in iterations to make designing as simple as possible.

Table 5.6.4 : Function Oriented v/s Object Oriented Design

Sr. No.	Function Oriented Design	Object Oriented Design
1.	Focuses on functions which operate on data.	Focuses on objects which are conceptual entities having attributes and methods.
2.	It is a set of functions each of which performs a task.	It is a set of objects that can be modified and that can perform tasks.
3.	Relies on identifying functions which transform their inputs to create outputs.	Relies on encapsulation of objects and messaging between objects.

Sr. No.	Function Oriented Design	Object Oriented Design
4	Divides a bigger problem set to small functional units and then organizes these functional units to design the solution.	Identifies the objects involved in the system and designs the solution based on their relationships and interactions.
5	Used mainly for computation of sensitive applications.	Used mainly for evolving systems that mimic a business process.

- Testing and debugging can be performed more effectively.
- Long-term maintenance can be performed by no serious side effects.

Syllabus Topic : Coupling and Cohesion

5.11 Coupling and Cohesion

Coupling

- Coupling within a software system is the degree to which each module depends on other modules. It is the degree of dependency among the components.

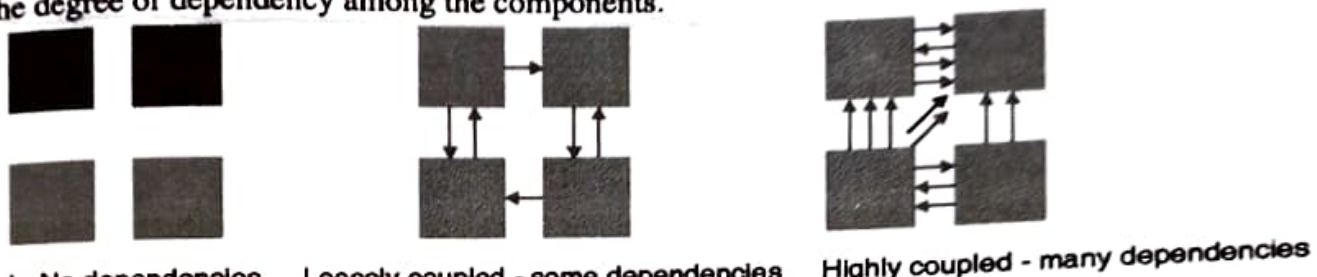


Fig. 5.11.1 : Various Types of Coupling

- Coupling is generally considered a bad thing where software components are dependent for some specific details of other components.
- In high coupling, the components are very closely and tightly tied to each other which make it difficult to modify the components of the system because modifying a component affects all other components to which the modifying component is connected.
- ***Therefore, Low-Coupling is good.***
- Object-oriented design has low coupling i.e. the objects (components) are independent of each other. Therefore, modifying one object doesn't have any effect on another object.

Example : Functions used in C++ & Java programming represent coupling i.e. dependency between software components (functions)

Cohesion

- Cohesion is the measure of internal interdependence with in a component i.e. the degree to which all elements of a component are directed towards a single task is called cohesion.
- ***Therefore, High-Cohesion is good.***
- Cohesion is considered as good thing because it is about creating a software component that combines related functionalities into a single unit. This is the core principle of OOD where it creates a component that encapsulates the objects of similar properties and functions that strive to perform same single task.

Example : Classes and objects in C++ & Java programming represent cohesion i.e. the degree to which the elements of a single component form a meaningful unit (object) and perform similar task.

- Cohesion represents how tightly the internal elements of a module are bound to one another - Coupling is degree of independence between different modules.
- High cohesion and low coupling is main criteria for good s/w design.

5.14 Verification for Designs

Design verification ensures that the product i.e. designed is the same as the product i.e. intended.

☞ Design Verification Methods

1. **Demonstration :** It is done in actual or simulated environments. The cost of demonstration varies as per the complexity of the demonstration.
2. **Inspection** : It is done to verify the requirements related to physical characteristics.

3. **Analysis :** It is done to verify the design and is the most preferred method incase if testing is not feasible or incase when there is minimal risk.
4. **Testing :** It is the most expensive verification methods due to the project complexity as well as equipment and facility requirements. The most common method for validating this requirement is transportation testing.

Verification Activities

Fig. C5.5 : Verification Activities

1. Identification and preparation

- While developing a specification, the test engineer starts writing detailed test plan and procedures.
- Identifying the best approach of verification, identifying the measurement methods and required resources, tools and facilities.
- Once the verification plan is ready, it is reviewed by the design team to identify issues before finalizing the plan.

2. Planning

- Planning for verification is a simultaneous activity with core development activity which continues throughout the project life cycle. The verification plan is updated as and when any changes are made to design inputs.
- Planning includes the scope of the project, tools, test environment, development strategy.

3. Developing

- The test case development identifies a variety of test methods.
- Developing of design inputs which are unambiguous and verifiable.
- Using the output of one test as an input for subsequent tests.
- Traceability links are created to ensure that all the requirements are tested and the design output meets the design inputs.

4. Execution

- The test procedures created in development phase are executed as per the test plan.
- If actual doesn't match with the expected results, such issues are identified and logged as defect at this stage.
- Traceability matrix is created to ensure that all the requirements are tested and the design output meets the design inputs.

5. Reports

- The design verification report gives the detailed summary of verification results which includes the configuration management and test results and defects found during the verification activity.
- Design verification traceability report ensures that all the requirements have been tested and provided with appropriate results.

.3 Types of Testing Metrics

List various types of metrics.

Metrics can be categorized into four types :

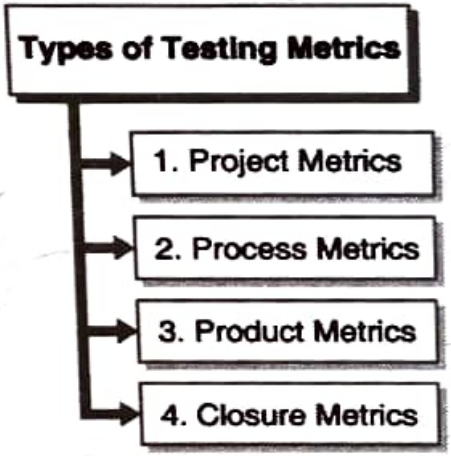


Fig. C6.2 : Types of Testing Metrics

6.2 Project Metrics

Project metrics are used by a project manager and software team to adapt project work flow and technical activities.

These metrics are useful in :

- minimizing the development schedule by making the necessary adjustment to avoid delays and mitigate problems.

- Assessing the product quality on an ongoing basis

1. Test coverage = (# of Test executed / # of Tests estimated) * 100
2. Defect Density = (Total number of defects found in all the phases + Post delivery defects)/Size

Defect arrival rate - This metric indicates the quality of the application/product under test.

Defect arrival rate = # Of Defects * 100 / # of Test Cases planned for Execution

6.3 Process Metrics

Process metric focuses on the quality achieved as a consequence of a repeatable or managed process. It is usually strategic and planned for long term.

1. Cost of Quality

% Cost of Quality = (green money + blue money + red money)*100/(Total efforts spent on project)

where,

- Prevention Cost: (Green Money) : Cost of time spent by the team in implementing the preventive actions identified from project start date to till date.
- Appraisal Cost: (Blue Money): Cost of time spent on review and testing activities from the project start date to till date
- Failure Cost: (Red Money) : Cost of time taken to fix the pre and post-delivery defects. Customer does not pay for this

1. Delivered Defect Rate
2. Defect Injection Rate
3. Rejection Index - measures the Quality of the defects raised
4. Rejection Index = # of Defects rejected / # of Defects raised
5. Review Effectiveness = 100 * Total no. of defects found in review / Total no. of defects
6. Defect Removal Efficiency (DRE) = (No. of pre-delivery defects / Total No. of Defects) * 100

6.4 Product Metrics

Product metrics focus on the quality of deliverables. Product metrics are combined across several projects to produce process metrics

1. Test Case Design Productivity = # Of Test cases (scripts) designed/ Total Test case design effort in hours
2. Test Case Execution Productivity = # Of Test cases executed/ Total Test case executed effort in hours

6.5 Closure Metrics

1. Test Design Review Effort = (Effort spent on Test Case design reviews / Total effort spent on Test Case design) * 100

Syllabus Topic : **Metrics for Object-Oriented Design**

6.7 Metrics for Object-Oriented Design

- OO methodology supports continuous integration of subsystems, classes, interfaces thus, increasing the chances of early detection of risks and incremental corrections without hampering the stability of the development process.

OO methodology allows 'Architecture first approach' in which integration is an early and continuous life-cycle activity that brings stability in development, allows development and configuration of components in parallel.

OO architecture creates a clear separation between the unrelated elements of a system, and includes firewalls that prevent a change in one part of the system that may be caused due to the errors in other part of the system thus, rending the structure of the entire architecture.

OO metrics are used to evaluate the quality of the software. The software engineering metrics are usually associated with coupling, cohesion, reliability, maintainability and fault-proneness. But the OO metrics are specially concerned with;

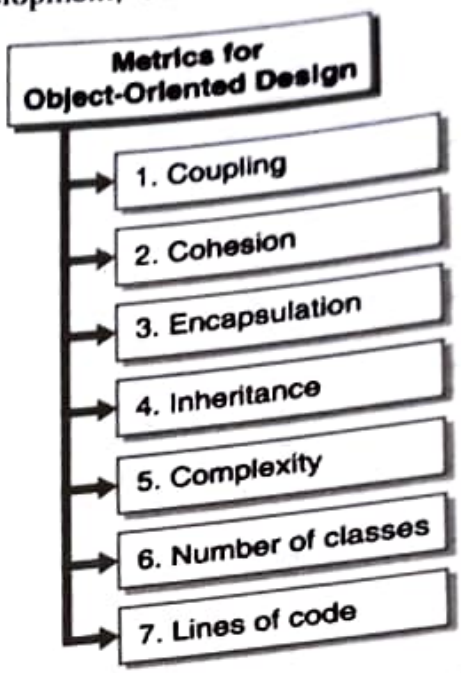


Fig. C6.3 : Metrics for Object-Oriented Design

1. Coupling

It refers to the association between modules. Coupling is a measure of interdependence of two objects.

Example : Objects X and Y are said to be coupled if a method of X calls or accesses a method or variable of Y.

$$\text{Coupling Factor (CF)} = \frac{\text{number of non-inheritance couplings}}{\text{maximum number of inheritance and non-inheritance couplings in a system}}$$

where, , Inheritance couplings arise from derived classes, inherited methods and attributes form its base class.

Desirable value of CF is lower.

2. Cohesion

It refers to how closely the operations in a class are related to each other. There are two ways of measuring cohesion :

i. Calculate the percentage of methods in a class that used the data field - Average the percentages then subtract from 100%. Lower percentages mean greater cohesion of data and methods in the class.

ii. Count the number of disjoint sets produced from the intersection of the sets of attributes used by the methods.

Desirable value of Cohesion is higher - High cohesion increases complexity, thereby increasing the likelihood of errors

3. Encapsulation

It refers to hiding of information means all that is seen of an object is its interface. There are two types of encapsulation measures.

i. **Attribute Hiding Factor (AHF) :** It measures the percentage of invisibilities of attributes in classes i.e. An attribute is said to be visible if it can be accessed by another class or an object. Attributes should be "hidden" within a class by declaring them as private. Desirable value of AHF is higher.

$$AHF = \frac{\text{sum of the invisibilities of all attributes defined in all classes}}{\text{total number of attributes defined in the project}}$$

ii. **Method Hiding Factor (MHF) :** It measures the percentage of invisibilities of methods in classes i.e. the percentage of total classes from which the method is not visible. Desirable value of MHF is higher.

$$MHF = \frac{\text{sum of the invisibilities of all methods defined in all classes}}{\text{total number of methods defined in the project}}$$

➜ **4. Inheritance**

It decreases the complexity by reducing the number of operations and operators, but on the contrary, it makes the maintenance and design difficult. There are two metrics to measure the amount of inheritance in terms of depth and breadth of the inheritance hierarchy.

i. **Depth of Inheritance Tree (DIT) : :** It is the depth from the current class node to the parent class node in the hierarchy tree and is measured by the number of ancestor classes The classes involved in multiple inheritance have maximum DIT. The deeper the class is within the hierarchy, the greater is the potentiality of reusing the inherited methods but increases the complexity. Desirable value of DIT is low.

ii. **Number of Children :** It measures the number of direct subclasses for each class. A class with large number of children is difficult to modify and test. Desirable value of NOC is low.

➜ **5. Complexity**

A class with more functions than its parent class is considered to be more complex and more error prone thereby limiting the possibility of reuse.

➜ **6. Number of Classes**

The projects having more number of classes are better abstracted. Desirable value of Number of Classes is higher.

➜ **7. Lines of Code**

The projects with fewer lines of code have superior design and require less maintenance. Desirable value of LOC is lower.

Syllabus Topic : Operation-Oriented Metrics

6.7.1 Operation-oriented Metrics

Q. What is the need of OO metrics and explain in brief.

6.11 Metrics for Software Quality

Q. Explain metrics of software Quality.

The following are the software quality factors and the needed metrics and measurements for each :

1. Correctness

- Correctness is measured as the degree to which the software performs its required functionality.
- The most common measure for correctness is number of defects per KLOC (Kilo Lines Of Code) where a defect is defined as a verified lack of conformance to requirements.

Fig. C6.5 : Metrics for Software Quality

2. Maintainability

- Maintainability is the ease with which a program can be :
- An application can be corrected if an error is encountered
- a defect can be fixed
- a application is adapted in a changed environment
  - a application is enhanced.
- A simple time oriented metric is Mean Time To Change (MTTC) i.e. the time it takes to analyze the changed request, design an appropriate modification, implement the changed modification, test it and distribute the change for all users. Below is the figure which gives the scenario of online detection and offline repair. The measures MTBF (Mean Time Between Failure), MTTD (Mean Time To Detect) and MTTR (Mean Time To Repair) are the average times to failure, to detection and to repair.

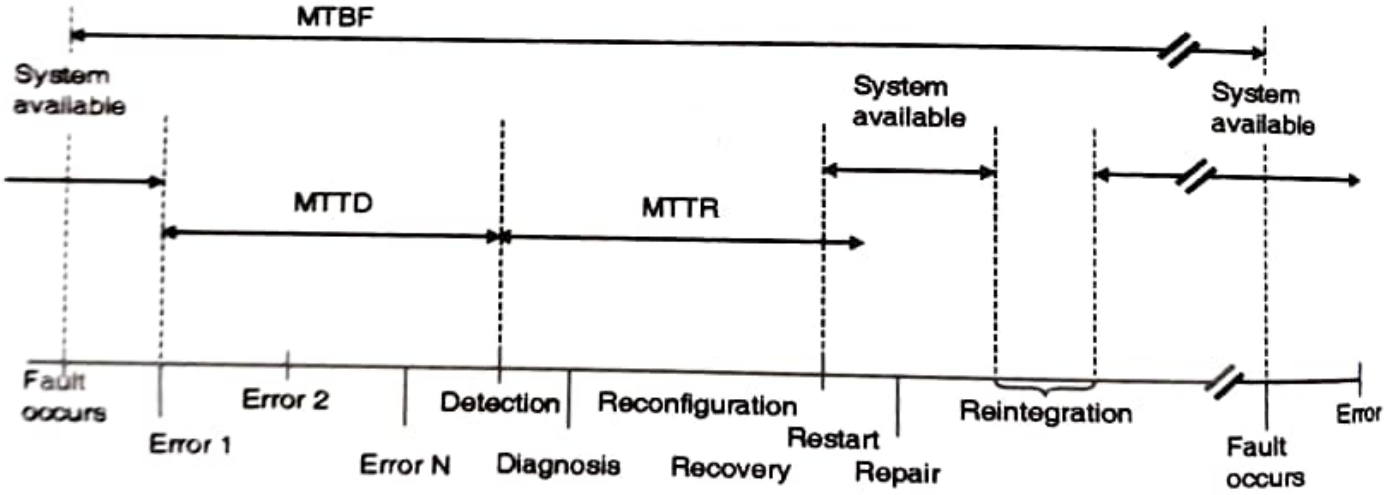


Fig. 6.11.1 : MTBF, MTTD and MTTR measures of maintainability

3. Integrity

- This attribute measures a system's ability to withstand attacks to its security. Attacks can be made on all three components of software: programs, data and documents.
- We cam measure integrity through following two additional attribute :
  1. Threat: It is the probability that an attack of a specific type will occur within a given time.
  2. Security: It is the probability that the attack of a specific type will be repelled. The integrity of a system van be then defined as :

     Integrity = Σ (1-(threat * (1-security)))

4. Usability

- If a program is not easy to use, it is often meant to failure. Even if the functions that it performs are valuable.

7.3 Cost Estimation Techniques

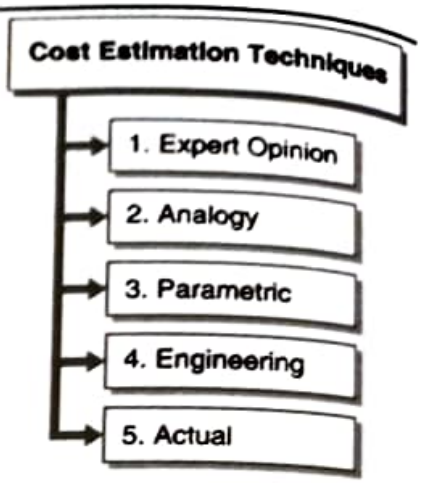


Fig. C7.1 : Cost Estimation Techniques

➜ 1. Expert Opinion

Also called as Delphi method, proposed by Dr. Barry Boehm is useful in assessing differences between past projects and new ones for which no historical precedent exists.

☛ Advantages

- Little or no historical data needed.
- Suitable for new or unique projects.

☛ Disadvantages

- Very subjective.
- Experts may do partiality
- Qualification of experts may be questioned.

➜ 2. Analogy

- Estimates costs by comparing proposed programs with similar, previously completed programs for which historical data is available.
- Actual costs of similar existing system are adjusted for complexity, technical, or physical differences to derive new cost estimates
- Analogies are used early in a program cycle when there is insufficient actual cost data to use as a detailed approach
- Compares similarities and differences
- Good choice for a new system that is derived from an existing subsystem.

☛ Advantages

- Inexpensive – Easily changed – Based on actual experience (of the analogous system)

☛ Disadvantages

- Very Subjective
- Large amount of uncertainty
- Truly similar projects must exist and can be hard to find
- Must have detailed technical knowledge of program and analogous system

➜ 3. Parametric :

Utilizes statistical techniques. Can be used prior to development.

☛ Advantages

- Can be excellent predictors when implemented correctly
- Easily changed
- Objective
- Once created, CERs are fast and simple to use
- Useful early on in a program

☛ Disadvantages

- Often lack of data on software intensive systems for statistically significant CER

Does not provide access to subtle changes
Top level; lower level may be not visible
Need to be properly validated and relevant to system

4. **Engineering :**

Also referred to as ***bottoms up*** or detailed method.

- Start at the component level and estimate the effort required for each component. Add these efforts to reach a final estimate.
- Future costs for a system are predicted with a great deal of accuracy from historical costs of that system.
- Involves examining separate work segments in detail.
- Estimate is built up from the lowest level of system costs.
- Includes all components and functions.
- Can be used during development and production.

Advantages

- Objective – Reduced uncertainty

Disadvantages

- Expensive – Time Consuming – Not useful early on
- May leave out software integration efforts

5. **Actual**

- Decides future costs on recent historical costs of same system.
- Used later in development or production.
- Costs are calibrated to actual development or production productivity for your organization.

Advantages

- Most accurate – Most objective of the five methodologies

Disadvantages

- Data not available early
- Time consuming
- Labour intensive to collect all the data necessary

Choice of methodology depends upon

- **Type of system** - software, hardware, etc
- **Phase of program** - Development, Production, Support
- **Available data** - Historical data points from earlier system versions or similar system or Technical parameters of system.

7.4 Cost Estimation Parameters

Various models (such as COCOMO, Co-star etc.) are available for estimating the cost of software development.

- All these cost estimating models can be represented on the basis of five basic parameters:

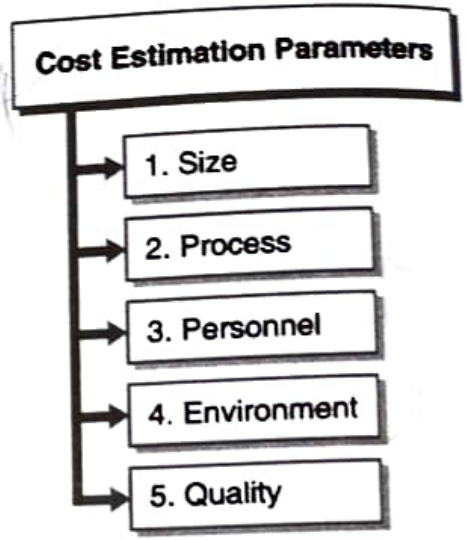


Fig. C7.2 : Cost Estimation Parameters

➜ **1. Size**

The size of the proposed software product is weighed in terms of components i.e. ultimately in terms of the number of source code instructions or the number of functions required in developing the proposed product.

➜ **2. Process**

The process includes the phases and activities carried out in each phase. So whatever process used to produce the proposed product is measured on the ability of the process to avoid unnecessary activities such as rework, bureaucratic delays, communications overhead and such other overhead activities which may delay the delivery of the product.

➜ **3. Personnel**

This deals with the capabilities and experience of software engineering *personnel* (team members) in the field of computer science issues and the applications domain issues of the project. It also depends upon the personnel's familiarity with the Development Tools and Technology Platform.

➜ **4. Environment**

The environment constitutes of the tools and techniques that are required to develop efficient software and also to automate the process.

➜ **5. Quality**

- The required quality of the proposed product depends upon the features, performance, reliability, and adaptability of the software.
- The above described five parameters (size, process, personnel, environment and quality) can be related with each other so as to calculate the estimated cost for the proposed software develop~~ment~~.

$$\text{Effort/Cost} = (\text{Personnel})(\text{Environment})(\text{Quality})(\text{Size}^{\text{process}})$$

8.1 Project Scheduling

Q. Explain the project scheduling process.

The *project schedule* is a calendar that is used to associate the tasks to be performed with the resources that will perform them. Before a project schedule is estimated, the project manager designs a work breakdown structure (WBS) which is an attempt to estimate the time needed to implement each task and the resources that are available for accomplishing each task.

Project Scheduling is dependent on project managers' intuition and experience.

Project Scheduling Process

- Divide the project into various tasks and estimate the duration and resources required to complete each task.
- Arrange the tasks so as to make optimal use of workforce.
- Reduce the task dependencies so as to avoid delays that might be caused by some task i.e. waiting for another to complete.

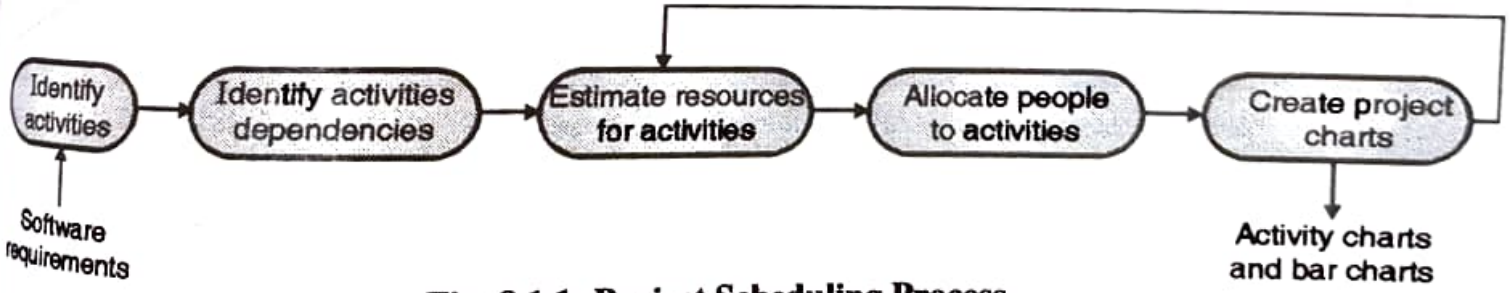


Fig. 8.1.1: Project Scheduling Process

Scheduling Problems

- Detecting the complexity of the problems and accordingly estimating the cost of developing a solution is difficult.

8.2 Basic Principles

Q. List the basic principles of Project scheduling.

Project scheduling is about distributing or allocating the estimated efforts to specific software engineering tasks across the planed project duration.

Project scheduling builds a road map for the project manager when combined with estimation methods and risk analysis.

Basic principles guide software project scheduling:

1. Compartmentalization: The project must be categorized into number of manageable activities, actions and tasks by decomposing the process into sub processes.
2. Interdependency: Few compartmentalized activities, actions, or tasks occur in sequence where they are interdependent upon each other and while others occur in parallel.
3. Time allocation: Scheduling is very important where each task must be allocated some number of work units, defining their start date and completion date especially in case of the inter dependencies and also state whether work will be conducted on a full-time or part-time basis.
4. Effort allocation: The project manager must allocate number of people that must be are present at any given point of time.
5. Effort validation: The project manager must monitor that no more than the allocated number of people are present at any given point of time.
6. Defined responsibilities: Every task that is scheduled must be assigned to a specific team member.
7. Defined outcomes: Every task that is scheduled must have a defined outcome.
8. Defined milestones: Each set of tasks must be associated with a project milestone. And the milestone is said to be accomplished only when its associated tasks are reviewed for quality.

8.3 Relationship between People and Effort

Q. Explain the relationship between people and efforts in project scheduling

- A single person or just a small team of members is enough for working on small projects. But as the scope of the project increases, team size also increases. Adding unplanned and sometimes unskilled people in the later SDLC phases causes the schedules to slip even further. The people who are added later in the project should first of all understand the system properly which requires additional effort and time.
- Project schedules are flexible. As the project deadline gets closer and closer, you reach a point where the work cannot be completed on time, regardless of the number of people working on it.

Risk involves the following two characteristics :

- **Uncertainty** : It is uncertain that risk may happen. In such case, risk may or may not happen. There is no certainty of risk to happen.
- **Loss** : If the risk becomes a reality, unwanted consequences or losses will occur.

Risk management is a software engineering practice with processes, methods and tools for managing risk in a project. It provides a disciplined environment for proactive decision making by providing access continuously what can go wrong, determining what risks are important to deal with and implementing strategies to deal with those risks.

Risk management includes the following steps during risk resolving :

a) **Risk Identification** : It helps in identifying the risk.

b) **Risk Analysis** : It helps in understanding the occurrence of risk, its impact and time of occurrence.

c) **Risk Assessment** : It is measured in terms of probability of occurrence, loss and risk exposure.

d) **Risk Planning for Resolution** : It deals with risk in terms of strategies and execution.

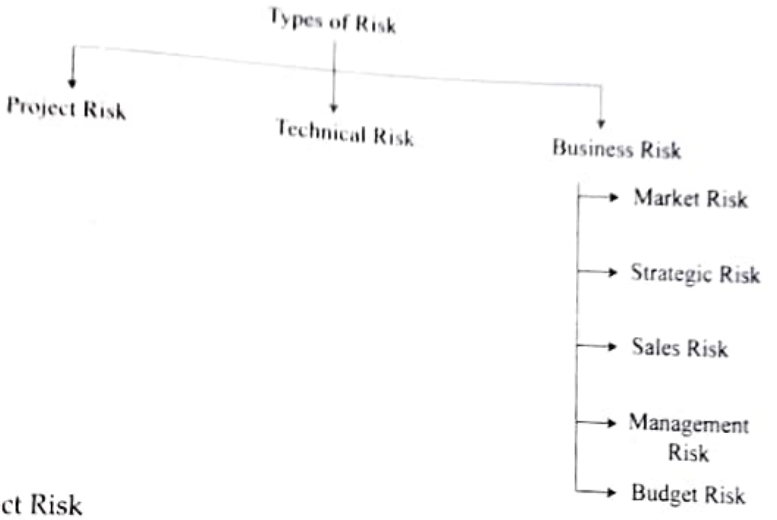
e) **Risk Monitoring** : It monitors the risk for its occurrence, impact and implication until it is resolved.

f) **Risk Mitigation** : It helps to meet the challenges of risk occurrence in effective implementation.

2. SOFTWARE RISK

Software risk is defined as a measure of the chance of the risk occurrence and the loss due to its exposure

Types of Risk



a) Project Risk

b) Technical Risk

c) Business Risk

a) **Project Risks :** Project risk is mainly concerned with different sorts of monetary fund, schedule, staffing, resource and customer-related issues. Since computer code is intangible, it's terribly tough to observe and management a computer code project i.e. it's terribly tough to manage one thing that is not visible to a naked eye and cannot be seen.

b) **Technical Risks :** Technical risk is concerned with potential style, implementation, interfacing, testing and maintenance issues. It is the degree of uncertainty between the actual and optimal design solutions. It mainly impacts on the quality and timeline of the project.

c) **Business Risks :** Business risk is a future possible risk that may prevent from achieving business goals. It includes the strategies that help in controlling business plans.

The various types of business risks are defined as :

i) **Market Risk :** It aims at building an excellent product or system that no one really wants.

ii) **Strategic Risk :** It aims at building a product that no longer fits into the overall business strategy for the Company.

iii) **Sales Risk :** It aims at building a product that the sales force doesn't understand how to sell.

iv) **Management Risk :** It aims at losing the support of senior management due to change in focus or a change in people.

v) **Budget Risk :** It aims at losing budgetary or personnel commitment.

According to Charette, risk is categorized in the following three categories :

Known risks are the risks which are identified prior during the identification of project specification and analysis of the scope and definition. They are thoroughly analysed by reviewing project plan and the environment in which project is being developed. For example, resources required for development, tools and techniques needed, lack of problem domain knowledge, expected delivery date etc.

Predictable risks are those which are foreseeable/expected and extrapolated from past project experience. For example, staff turnover, stakeholder communication, support and maintenance of the project etc.

Unpredictable risks are unforeseen, unexpected and hard, if not impossible, to identify in advance. For example, change in policies affecting business project etc.

3. RISK IDENTIFICATION

Risk identification is defined as a process of handling specific inputs that generates a set of output or risk with context to the software proposed to be developed. It is a systematic attempt to specify threats to the project plan which can be achieved by identifying known and predictable risk.

There are two distinct types of risks approaches :

1. Generic risk identification which includes potential threat identification to a software project.
2. Product-specific risk identification which includes threat identification specific to a product with clear knowledge of technology, people and the environment which are specific to the software.

Known and predictable risks are categorized in the following generic subcategories :

- Product size defines the risks which identifies the overall size of the software to be developed or modified.
- Business impact risks are associated with consistency specified by management or the marketplace.
- Stakeholder characteristic risks define the sophistication of the stakeholders and the developer's ability to communicate within a specific time duration.
- Process definition risks are related with the degree that defines the software process to be followed by the development organization.
- Development environment risks are related with the availability and quality of the tools and techniques required to build the product.
- Technology to be built risk is associated with the system complexity and technology used for system development.
- Staff size and experience risk identifies the risk associated with the experience of the personnel who are deployed on the software development.

6. RISK MITIGATION, MONITORING, AND MANAGEMENT (RMMM)

RMMM stands for risk mitigation, monitoring and management, whose main goal is to identify as many potential risks as possible, measure and monitor the severity of risk to suggest resolution strategy by generating a plan.

RMMM plan deals with risk in a systematic manner using these different strategies.

RMMM is summarized as: Risk Mitigation is a process of risk prevention i.e. it focuses on avoiding the risk to be occurred. Risk Monitoring is a phenomenon of tracking the risk occurred. Risk Management includes contingency plans to be implemented if risk occurs.

When all risks have been identified, they will then be evaluated to determine their probability of occurrence. Plans can then be created to avoid every risk, to trace every risk to see if it's a lot of or less doubtless to occur, and to set up for those risks should they occur.

RISK MITIGATION

It deals with proactive planning for risk avoidance by developing team strategies to reduce the possibility or the loss impact of a risk. Risk mitigation produces a scenario within which the risk can be eliminated or otherwise resolved.

It is the process of strategizing options and actions to enhance opportunities and reduce threats to project objectives and includes tracking identified risks, identifying new risks, and evaluating risk process effectiveness throughout the project.

Following are the steps to be taken for risk mitigation :

a) To find the probable risk, it is necessary to communicate the associated staff.

b) Before project begins, eliminate all the causes of risk creation.

c) Develop a policy beneficial for project development, in case any staff resigns.

d) Update the current status of project time to time to continue with project development smoothly.

e) Timely maintenance of documents in standard format.

f) Periodic reviews can be conducted to monitor the progress of the work.

g) Critical activities can be conducted with additional staffing, if required.

RISK MONITORING

Risk monitoring is ongoing activity of risk management. It involves identifying risk and controlling designs by tracking the execution of risk management and continuing to identifying and managing new risks.

It is necessary that the risk must be monitored continuously by the project risk manager and the project team to ensures that new and changing risks have been detected and well-managed and the corresponding actions are implemented effectively.

It also monitors the execution of planned strategies for the identified risks and evaluates their effectiveness. Risks needs to be revisited at regular intervals for the team to re-evaluate each risk to determine when new circumstances caused its probability and/or impact to change. At each interval, some risks may be added to the list and others taken away. Risks needs to be reprioritized to have action plans and which move "below the

line" and no longer need action plans. Identifying proactive actions is a key to successful risk management owned by individuals and are monitored.

If an unanticipated risk emerges, or a risk's impact is greater than expected, the planned response may not be adequate, than additional responses to control the risk should be performed.

Monitoring focuses on the following :

a) Periodic risk review.

b) Following policies and procedures.

c) Cooperation among team members.

d) Identification of the type of problem occurring.

e) Implementing a contingency plan

f) Taking corrective actions

g) Changing the project objectives

RISK MANAGEMENT

Risk management is a project tracking activity which is carried out throughout the development process to identify, manage and control risks during the development process.

It focuses on risk containment and mitigation. Management starts with identification of plan, and when risk arises it impact is minimized depending upon the experience and knowledge of the entire team.

The primary objectives are to assess whether predicted risks do, in fact, occur and to ensure that the desired actions to minimize risk are properly applied. It collects information that can be used for future risk analysis.

It includes the following activities :

a) Identify the risks and its cause.

b) Prioritize all risks.

c) Create a plan linking each risk to a mitigation.

d) During the project monitor risk triggers.

e) In case of risk materialization, implement the mitigating action.

f) Communicate risk status throughout project.

RMMM Plan

The RMMM plan documents all the work performed as a part of risk analysis and is used by project manager as a part of overall project plan. Some software team develop RMMM plan where each risk is documented individually using risk information sheet (RIS). The risk mitigation and monitoring start after the project is started and the documentation of RMMM is completed.

1. INTRODUCTION TO QUALITY

Quality varies from user to user, product to product as well as it differs from the use of the system. It is an abstract factor to be defined but is recognized as to be present. It can be defined as fitness for use or conformance to standards. A quality product is defined in terms of its fitness of purpose robustness i.e. a quality product is intended to meet the user's expectations. For software products, fitness of purpose is analysed in terms of satisfaction of the specifications as mentioned in Software Requirement Specification documentation.

Software quality can be defined based on the following :

a) Customer focus and satisfaction.

b) Functional and specification requirements.

c) Ease of use, learning and maintainability.

d) Adherence to development standards.

Software quality management provides independent checks on software development process. For small system development, software quality management is done with the help of informal communication between team members. For large system, it is done with respect to the following features:

a) **Quality Assurance :** It aims in forming the organizational procedures and standards. It is performed to ensure that the software is of high quality.

b) **Quality Planning :** Its main focus is on selecting appropriate procedures and standards for implementing the software project.

c) **Quality Control :** It ensures that the software must be developed using quality procedures and standards.

The quality management checks whether or not the software being developed is as per the organizational goals and standards.

Software quality assurance (SQA) is a process which assures that all software engineering processes, methods, activities and work items are monitored and comply with the defined standards.

It is outlined as a planned and systematic approach to the analysis of the standard of and adherence to the software standards, processes and procedures.

It ensures that a software system meets its quality goals with the use of effective tools and methods, assuring that the standards will be followed throughout the software acquisition life cycle.

A formal definition of package quality assurance can be put as "the systematic activities providing proof of the fitness to be used of the entire product." Software quality assurance is achieved through the utilization of established guidelines for internal control to make sure the Integrity and prolonged lifetime of the product.

Quality assurance is the set of support activities required to produce adequate confidence that processes area unit established and incessantly improved so as to develop a product that meet specifications and area unit acceptable use. Quality control is that the method by that product quality is compared with applicable standards and also the action taken once heresy is detected. Auditing is the inspection/ assessment activity that verifies compliance with plans, policies and procedures.

2. ELEMENTS OF SOFTWARE QUALITY ASSURANCE

The elements of SQA are listed as under :

a) Standards

A broad array of software engineering standards and related documents have been produced by the IEEE, ISO and other standards organizations. Standards may be adopted voluntarily by a software engineering organization or imposed by the customer or other stakeholders. The job of SQA is to ensure that standards that have been adopted are followed and that all work products conform to them.

b) Reviews and Audits

Technical reviews are a quality control activity performed by software engineers for assessment of software. The intention behind is to uncover errors. Audits are a type of review performed by SQA personnel with the intent of ensuring that quality guidelines are being followed for software engineering work. For example, an audit is a review process which might be conducted to ensure that reviews are being performed in a manner that will lead to the highest likelihood of uncovering errors.

c) Testing

Software testing is a quality control function with one primary goal of finding and detecting errors. The job of SQA is to ensure that testing is properly planned and efficiently conducted so that it has the highest likelihood of achieving its primary goal.

d) Error/Defect Collection and Analysis

The only way to improve is to measure how you're doing. SQA collects and analyzes error and defect data to better understand how errors are introduced and what software engineering activities are best suited for eliminating them.

e) Change Management

Change is one of the most disruptive aspects of any software project. If it is not properly managed, change can lead to confusion. The confusion in turn almost always leads to poor quality.

f) Education

Every software organization wants to improve its software engineering practices and maintain them of a high degree. A key contributor to improvement is education of software engineers, their managers and other stakeholders. The SQA organization takes the lead in software process improvement and is a key proponent and sponsor of educational programs.

g) Vendor Management

Three categories of software are acquired from external software vendors—*shrink-wrapped packages* (e.g., Microsoft *Office*), a *tailored shell* [Hor03] that provides a basic skeletal structure that is custom tailored to the needs of a purchaser and *contracted software* that is custom designed and constructed from specifications provided by the customer organization. The job of the SQA organization is to ensure that high-quality software results by suggesting specific quality practices that the vendor should follow (when possible) incorporating quality mandates as part of any contract with an external vendor.

2. ELEMENTS OF SOFTWARE QUALITY ASSURANCE

The elements of SQA are listed as under

a) Standards

A broad array of software engineering standards and related documents have been produced by the IEEE, ISO and other standards organizations. Standards may be adopted voluntarily by a software engineering organization or imposed by the customer or other stakeholders. The job of SQA is to ensure that standards that have been adopted are followed and that all work products conform to them.

b) Reviews and Audits

Technical reviews are a quality control activity performed by software engineers for assessment of software. The intention behind is to uncover errors. Audits are a type of review performed by SQA personnel with the intent of ensuring that quality guidelines are being followed for software engineering work. For example, an audit is a review process which might be conducted to ensure that reviews are being performed in a manner that will lead to the highest likelihood of uncovering errors.

c) Testing

Software testing is a quality control function with one primary goal of finding and detecting errors. The job of SQA is to ensure that testing is properly planned and efficiently conducted so that it has the highest likelihood of achieving its primary goal.

d) Error/Defect Collection and Analysis

The only way to improve is to measure how you're doing. SQA collects and analyzes error and defect data to better understand how errors are introduced and what software engineering activities are best suited for eliminating them.

e) Change Management

Change is one of the most disruptive aspects of any software project. If it is not properly managed, change can lead to confusion. The confusion in turn almost always leads to poor quality.

f) Education

Every software organization wants to improve its software engineering practices and maintain them of a high degree. A key contributor to improvement is education of software engineers, their managers and other stakeholders. The SQA organization takes the lead in software process improvement and is a key proponent and sponsor of educational programs.

g) Vendor Management

Three categories of software are acquired from external software vendors—*shrink-wrapped packages* (e.g., Microsoft *Office*), a *tailored shell* [Hor03] that provides a basic skeletal structure that is custom tailored to the needs of a purchaser and *contracted software* that is custom designed and constructed from specifications provided by the customer organization. The job of the SQA organization is to ensure that high-quality software results by suggesting specific quality practices that the vendor should follow (when possible) incorporating quality mandates as part of any contract with an external vendor.

h) Security Management

With the increase in cyber crime and new Government regulations regarding privacy, every software organization should institute policies that protect data at all levels, establish firewall protection for Web Applications and ensure that software has not been tampered with internally. SQA ensures that appropriate process and technology are used to achieve software security.

i) Safety

Since a software is almost always a pivotal component of human rated systems (e.g. automotive or aircraft applications), the impact of hidden defects can be catastrophic. SQA may be responsible for assessing the impact of software failure and for initiating those steps which are required to reduce risk.

j) Risk Management

Although the analysis and mitigation of risk is the concern of software engineers, the SQA organization ensures that risk management activities are properly conducted and that risk-related contingency plans have been established.

3. SQA TASKS, GOALS, AND METRICS

3.1 SQA Tasks

Software quality assurance comprises of a different task associated with technical engineers and an SQA team responsible for planning, record keeping, analysis and reporting.

These activities are performed by an independent SQA team that :

a) Prepares a SQA Plan for a Project

During project planning, a plan is developed and is reviewed. The project activities ensuring quality are carried out by the software engineering and SQA team which are governed by the plan. The plan identifies assessments and audits to be performed, standards applicable to the project, procedures for error reporting and tracking, documents to be produced by the SQA group and the feedback will be provided to the software system project team.

b) Reviews software engineering activities to verify compliance with the defined software process

The SQA team monitors and control the discrepancies if any in the process.

c) Audits designated software work products to verify compliance with those defined as part of the software process

The SQA team reviews selected work products, identifies, documents and tracks deviations; verifies that corrections have been made and document the result periodically.

d) Ensures that deviation in software work and work products are documented and handled according to a documented procedure

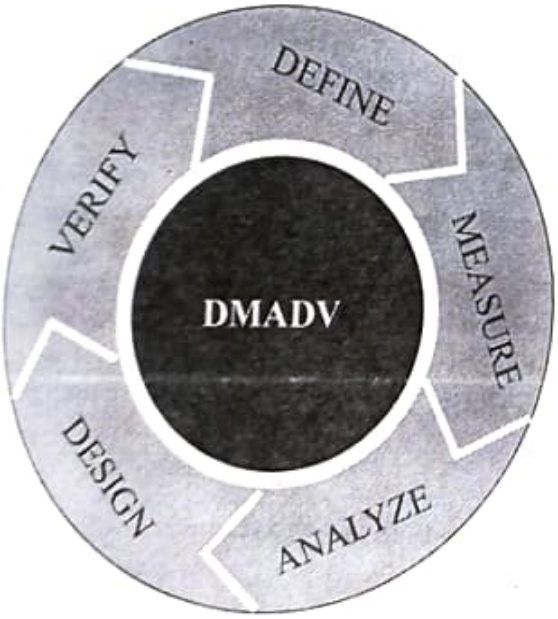
Deviations could also be encountered within the project set up, method description, applicable standards or in product technical work.

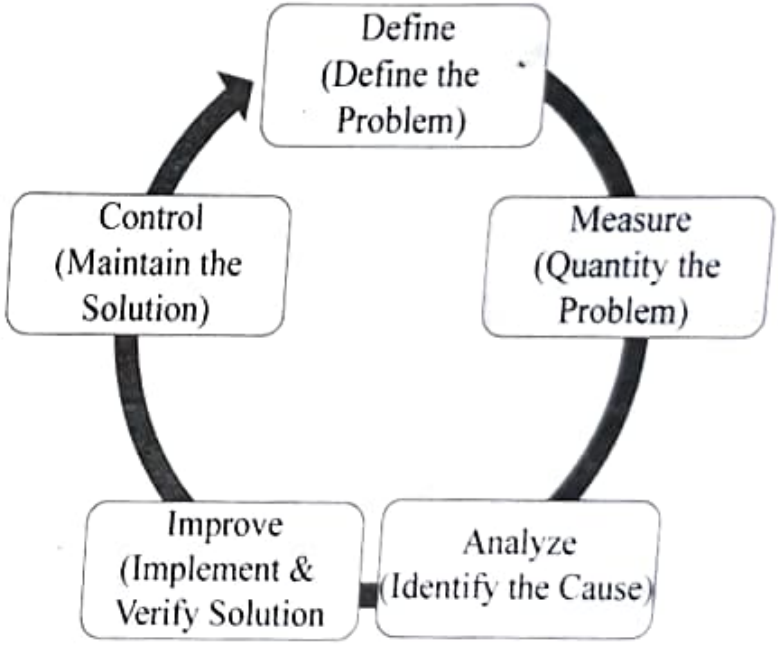
e) Records any non compliance and reports to Senior Management

Non compliances, if any are tracked until they are resolved.

5. SIX SIGMA

Six Sigma is a statistical software quality assurance discipline. It is a data-driven approach used for eliminating defects and increasing profit in any process from production to transactional and from product to service.





The main objective of Six Sigma methodology is to reduce the defects by implementing measurement-based strategy whose primary focus is on improving methods of development.

Six Sigma projects follows the following two project methodologies :

1. DMAIC (Define, Measure, Analyze, Improve, Control), is a data-driven quality strategy for improving processes and enhancing an existing business process for incremental improvement.
   - **Define** : It identifies the customer specifications, project goals and deliverables.
   - **Measure** : It helps in understanding current performance and establishing the baseline for improvement.
   - **Analyze** : The purpose of analysis id to determine and validate the root causes of any defects.
   - **Improve** : It establish the solution to eliminate defects and correct the process.
   - **Control** : It manages future process performance and controls the existing processes.
2. DMADV (Define, Measure, Analyze, Design, Verify), is an improvement system which deals with designing new products and processes which results in a more predictable, mature, and detect free performance. The primary focus on the development of a new service, product or process as opposed to improving a previously existing one.
   - **Design** : Create a process that meets customer needs and expectations.
   - **Verify** : It involves verifying that the end result must meet the customer requirements.

7. THE ISO 9000 QUALITY STANDARDS

Quality assurance is a planned and systematic approach necessary to provide a high degree of confidence in the quality of a product. It provides quality assessment of the quality control activities and determines the validity of the data for identifying the quality.

The main function of quality assurance is to define the standards as product standards and process standards.

Quality assurance is a function consisting of auditing and reporting activities which helps in determining the effectiveness of the quality system.

Importance of Standards

- Encapsulation of best practice- avoids repetition of past mistakes.
- They are a framework for quality assurance processes - they involve checking compliance to standards.
- They provide continuity - new staff can understand the organisation by understanding the standards that are used.

Problems with Standards

- They may not be seen as relevant and up-to-date by software engineers.
- They often involve too much bureaucratic form filling.
- If they are unsupported by software tools, tedious manual work is often involved to maintain the documentation associated with the standards.

Standards Development

- Involve practitioners in development. Engineers should understand the rationale underlying a standard.
- Review standards and their usage regularly. Standards will quickly become noncurrent and this reduces their believability amongst practitioners.
- Detailed standards should have associated tool support. Excessive clerical work is that the most important grievance against standards.

ISO 9000 : The ISO 9000 quality assurance models. It is an international set of standards for quality management which is applicable to a range of organisations from manufacturing to service industries and treat an enterprise as a network of interconnected processes. ISO 9001 applicable to organisations that design, develop and maintain products.

ISO 9001 is a generic model of the quality process that must be instantiated for each organisation using the standard which describes the elements of a quality assurance system. These elements embrace the structure, procedures, processes, and resources needed to implement quality planning, quality control, quality assurance, and quality improvement.

1. VERIFICATION AND VALIDATION

The Software Quality Assurance (SQA) process focuses on verification and validation process of the software code which ensures that the software will meet the customer specifications.

Verification and validation start by reviewing the requirements, designing the specifications and analysing the code up to the testing of the product. Therefore, verification focuses on checking that the program meets specified requirements whereas, validation examines that the software product meets the customers' expectations as well as a formal proof of program correctness (IEEE, 1990).

Verification and validation (V & V) are referred to the process of checking whether a product meet the system specifications and fulfills its desired purpose.

Barry Boehm described verification and validation as the following :

Verification : Are we building the product right?

Validation : Are we building the right product?

IEEE standard in its 4th edition defines the two terms as follows :

- **Validation :** The assurance that a product, service or system meets the needs of the customer and other identified stakeholders. It often involves acceptance and suitability with external customers.
- **Verification :** The evaluation of whether or not a product, service or system complies with a regulation, requirement, specification or imposed condition. It is often an internal process.

Verification : Verification refers to the set of activities ensuring that the system implements a specific function correctly.

It is a process performed at each phase of software development life cycle to identify that the outcome at the end of each phase meets the specifications as defined. It is an internal or static process.

In the course of verification, the different questions to be answered are :

- Are we building the system right?
- Are the specified requirements implemented?
- Will the software satisfy the needs of the customer as per the specifications?
- Is the data accessing the way as per the specifications?
- Are the required credentials, strategies ad stipulation considered as per requirement document?

The various activities involved in verification are feasibility reviews, requirements reviews, technical reviews, walkthrough, Inspections, Formal reviews, Informal reviews, Peer reviews and Static Code Analysis.

Validation : Validation is the process of evaluating the software at the end of the development phase in order to determine whether it satisfies the specified requirements.

The main objective of validation process is to ensure that the product actually meets the specifications and fulfills its intended use when placed in its intended environment.

Validation is often performed at different phases throughout the system development life cycle. System testing is a major technique performed to validate the system against

may be willing to tolerate the software faults.

2. INTRODUCTION TO TESTING

Testing is a process of identifying risk i.e. future negative consequence of a system. It is defined as a process of demonstrating that errors are not present in the system.

The purpose of testing is to show that a program performs its specified function correctly and meet the objectives and scope of the software.

It is also defined as a process of executing a program with the intent of finding the errors.

The goal of software testing is to convince system developers and customers that the software is good enough for operational use. Testing may be a method meant to create confidence within the code.

It is also stated as the process of verifying and validating a software product. It checks whether the software product :

- Meets the business and technical requirements that guided its design and development.
- Works as per the requirement.
- Can be implemented with the same characteristics.

It is necessary to keep in mind that alike human beings who tend to make mistakes, which when executed can lead to software failures. Some errors and mistakes are minor

but some are quite dangerous and require constant evaluation and monitoring. Therefore, it has become vital for software engineers to implement testing throughout the software life cycle, to ensure the efficiency of the developed software as well as to verify that no defects or bugs are left undetected, as they can hamper the intended quality, effectiveness, & performance of the software. Moreover, testing is a process that helps to measure the quality of the software and gives confidence in it.

Other reasons that build testing necessary throughout SDLC are :

- Makes use of the customers reliability and their satisfaction in the software/application.
- It is necessary to facilitate the customer with the delivery of high-quality product that requires low maintenance.
- Reduces overall level of risks.
- Prevents failures of software applications, which can prove to be very expensive if not prevented.
- It is required to properly understand the faults and defects within the system.
- Testing validates effective performance and functionality of the software product.
- Helps learn more about software application's reliability, stability, security and more.
- Allows us to detect defects during the early stages of software development.
- Ensures the standard, effectiveness, and efficiency of the software before its release.
- Validates its compliance with user environment, various rules and regulations, as well as necessary regulatory standards.
- It is necessary to improve business.

Basic Terminologies of Testing

Error : A Human mistake is called as an 'error'.

Defect : The discrepancy between the actual and expected results is known as 'defect'.

Bug : If testers find any mismatch in the system/application in testing phase it is termed as 'Bug'.

Failure : When build system does not meet users' requirements, it results into failure.

Test : A Test is a group of related test cases, or a group of related test cases and test procedures.

Test Set : A group of related tests is sometimes referred to as a test set.

Test Suite : A group of related tests that are associated with a database, and are usually run together, is sometimes referred to as a Test Suite.

Test Case : A test case in a practical sense is attest related item which contains the following Information such as test inputs, execution conditions and expected outputs.

Error : A Human mistake in coding is called as an 'error'.

When we run a program the error that we get during execution is termed on the basis of runtime error, compile time error, computational error, and assignment error.

3. TESTING PRINCIPLES

There are seven principles in software testing :

1. Testing shows occurrence of defects
2. Exhaustive testing is not possible
3. Early testing
4. Defect clustering
5. Pesticide paradox
6. Testing is context dependent
7. Absence of errors fallacy

- **Testing Shows Occurrence of Defects :** The definite goal of software testing is to ensure the software fails. Testing reduces the presence of defects/bugs in the program. Testing talks about the presence of defects/bugs and does not talk about the absence of defects/ bugs. Testing might ensure that defects are present but it cannot prove that software is defects free. Even after multiple levels of testing, a tester can never ensure that software is 100% bug-free. Testing can reduce the number of defects/bugs but not removes all of them. Testing always helps eliminate the number of undiscovered defects/bugs in the software, however, even no defects/bugs are found during the testing process, it doesn't mean the correctness of the software.
- **Exhaustive Testing is Not Possible :** A process of testing the functionality of a software in all probable inputs (valid or invalid) and prerequisite focusing on the important criteria and impacted domains this is known as exhaustive testing. It is practically impossible to test the Software with all the combinations of data and inputs and use cases as it requires lot of time and efforts. It can test only some test cases and assume that software is correct and it will produce the correct output in every test cases.
- **Early Testing :** With Agile methodology in place and majority of the Development industries adhering to sprint/iterative based delivery approach it is beneficial to find the defect in the software early. The defect detected in early phases of SDLC will be cheaper to be fixed. For better performance of software, software testing will start at initial phase i.e. testing will perform at the requirement analysis phase. Therefore, with the early testing, testers can detect the bugs and help development team resolve it with less costs and efforts.
- **Defect Clustering :** In a project, a small number of the module can contain most of the defects. Pareto Principle to software testing state that 80% of software defect comes from 20% of modules. This process requires the testing team to utilize their knowledge and experience to categorize the potential impacted modules to test. This categorization or targeted testing can help save the time and effort, as the team only needs to focus on the "sensitive" areas. Nevertheless, this approach also contains drawbacks: once testers only concentrate on a small area all the team, they may miss the defect/bugs from other areas.
- **Pesticide Paradox :** Or Repetitive testing, repeating the same test cases repeatedly on the limited set of modules will not find new bugs. Therefore, it is necessary to

revisit the test cases and add or update test cases to find new bugs across the system and other modules.

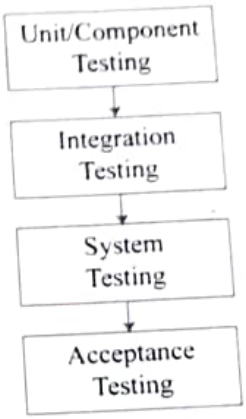
- **Testing is Context Dependent :** Testing approach depends on context of software developed. A tester cannot apply the same test approach for different project and changes. Different types of software need to perform different types of testing. For example, the testing of the e-commerce site is different from the testing of the Android application, an application in banking industry should require more testing than an entertainment software.
- **Absence of Errors Fallacy :** The software must be tested both on the functional and non-functional user requirements. If a built product is 99% bug-free but it does not follow the user requirement then it is unusable. It is not only necessary that software is 99% bug-free but it also mandatory to fulfill all the customer requirements.

Test oracles deals with only small subset of the inputs and outputs test. The tester focuses on specific set of values of some variables wit other variables of the program, due to which evaluation is based only on result set and may be incorrect or partially correct. The produced result affected with respective configuration settings, amount of available memory, and p options of the software program to be tested.

6. LEVELS OF TESTING

A level of software testing is a process in which each unit or component is tested to evaluate that the system meets with the specified needs.

The various levels of testing help to check behavior and performance for the software as they are designed to recognize the missing specifications and identify the flow of the system at the development life cycle state.



There are mainly four testing levels are :

1. Unit/component Testing
2. Integration Testing
3. System Testing
4. Acceptance Testing

1. Unit Testing

Unit testing is the smallest testable component of any software. It is performed to check whether the individual modules of the source code are working properly. In unit testing, each and every unit of the application is tested separately by the developer in the developer's environment.

Since the software comprises of various units/modules, detecting errors in individual units is simple and consumes less time, as they are small in size. However, it is possible that the outputs produced by one unit become input for another unit. The purpose is to validate that each unit of the software performs as specified.

It is also known as Module Testing or Component Testing.

Unit testing focuses on implementation and require thorough understanding of the systems functional specification. The objective is to isolate a section of code and verify its correctness.

Unit testing is done before integration testing, using Whitebox testing technique and is mainly carried out by the developer.

The main focus is to uncover the errors in design and implementation. It may be done in isolation from rest of the system depending on the development life cycle model chosen for that particular application. In such case the missing software is replaced by **Stubs** and **Drivers** and stimulate the interface between the software components in a simple manner.

A stub is called from the software component to be tested. As shown in the figure below 'Stub' is called by 'component A' and a driver calls the component to be tested. As shown in the diagram below 'component B' is called by the 'Driver'.

2. Integration Testing

Integration testing is performed to test the functionality of the components when integrated together to identify the interaction between them.

The objective is to integrate units and build a program structure that has been designed within the initial place and test whether or not it is operating within the given manner as intended to.

Integration testing focuses on reducing risk, verifying whether the functional and non-functional behaviors of the interfaces are as designed and specified, to build confidence in the quality of the interfaces, to find defects and prevent defects from escaping to higher test levels.

It is performed by developers or independent testers.

3. System Testing

System Testing is a level of software testing where a complete and integrated software is tested to identify the overall behavior of the system.

System testing is a process of evaluating the integrated hardware and software system as a whole. System testing is a level of software testing in which a complete and fully integrated software product is tested to verify that the system meets requirements and works as expected of or not. This testing comes under the black box type testing so only the working features are evaluated and there is no need to know about the internal structure or design of the code, internal path details. This testing is done from the user's point of view.

System Testing is performed by software testing team that is independent of the development team to measure the quality of the system. System testing is performed in the context of a system requirement specification (SRS) or a functional requirement specifications (FRS) or both. This testing contains both functional and Non-functional testing of an application. This testing is done after completing the unit and integration testing and before acceptance testing.

4. Acceptance Testing

After the successful completion of system testing, user is expected to take over the system for acceptance testing. A user is expected to have its own series of tests, which are close to real-life situation.

The objective is to test the system using live data and to provide the training to the end user on the proposed system. Acceptance test is also referred to as beta testing.

Acceptance testing is that the final and one in every of the foremost necessary levels of testing on prospering completion of that the appliance is discharged to production.

It aims at ensuring that the product meets the specified business requirements within the defined standard of quality.

There are two types of acceptance testing- alpha and beta testing. When acceptance testing is done at the developer's site to identify the functionality of the developed product, it is known as alpha testing. However, when testing is performed at end-user's site, it is named as beta testing.

5. White-Box Testing/Structural Testing

White Box testing is also termed as clear box testing, glass box testing , transparent box testing and structural testing, is a technique of software testing which tests the internal structure of an application.

It is a test case design philosophy that uses the control structure described as part of component-level design to derive test cases.

In tis approach, test group must have complete knowledge about the internal structure of the software. It is an approach where test cases are derived from the knowledge of the software structure and implementation. Testers can analyze the code and use knowledge about the structure of a component to drive test data.

It is generally applied to small program units or operations associated with them.

Testcases are generated to test the execution of all the independent paths, statements, loops, conditions, decisions etc.

It is applicable to unit testing, integration testing and system testing.

White box testing involves the testing of the software system code for the following :

- Internal security holes.
- Broken or poorly structured ways within the cryptography processes.
- The flow of certain inputs through the code.
- Expected output.
- The functionality of conditional loops.
- Testing of every statement, object, and performance on a personal basis.
- It checks for complexity of code.

Advantages

- White box testing optimizes code so hidden errors can be identified.
- It is easy to automate the test cases of white box testing.
- This testing is more thorough than other testing approaches as it covers all code paths.
- It can be started in the SDLC phase even without GUI.
- It helps in code optimization.

Disadvantages

- It is a time consuming, expensive and complex type of testing.
- White box testing needs professional programmers who have a detailed knowledge and understanding of programming language and implementation.
- A complex and expensive procedure which requires the adroitness of a seasoned professional, expertise in programming and understanding of internal structure of a code.
- Some conditions might be untested as it is not realistic to test every single one.
- Defects in the code may not be detected or may be introduced considering the ground rule of analysing each line by line or path by path.

6. Functional/Black-Box Testing

Black box testing is also known as functional testing, in which the structure of the program is not considered. Functional testing refers to testing that involves only observation of the output for certain input values without analysing the code. In black box testing, internal structure of the program is ignored.

Black box testing is also known as behavioral testing since the content of a black box is unknown and the function of black box is understood completely in terms of inputs and outputs.

In this testing, the tester only known input to be given and expected output to be achieved. The basis for deciding the test cases is the requirements or specification of the system or module.

Main focus of black box testing is on the external behavior of a system and its functions as per the specifications of the system.

Black box testing identifies the following types of errors :

- Incorrect or missing functions.
- Interface missing or erroneous.
- Error in data model.
- Error in access to external data sources.
- Initialization and termination error.

Advantages of Black Box Testing

- It is generally used for large applications.
- Any non- technical resource can test the application or code using black box testing techniques, but he should be well equipped with testing techniques.
- Testers do not have to wait for the development to be completed.
- Blackbox testing is unbiased since the designer and tester work independently.
- Test cases formation is faster as the testers don't have to create test cases from internal functionalities of the application but from the GUI pages.
- Tester need not to have knowledge of specific programming languages to test the reliability and functionality of an application / software.
- Test cases can be designed once specifications are completed.

Disadvantages of Black Box Testing

- Designing test cases is challenging without having clear knowledge of functional specifications.
- Limited inputs defined depending upon specification.
- Unidentified paths might occur during the testing process.
- Due to tester's lack of knowledge about internal code, there might be insufficient testing.
- Complex segment codes cannot be tested using black box testing.

Black Box Testing Techniques are :

a) Equivalence Partitioning
b) Boundary Value Analysis
c) Decision Table Testing
d) State Transition Testing
e) Error Guessing
f) Cause / Effect Graphing

7. TEST PLAN

A test plan is a document which describes the scope, approach, objectives, resources and schedule of a software testing effort and identifies the items to be tested and items not to be tested, who will perform testing, which test approach is to be followed, what will be the pass/fail criteria, training needs for team, the testing schedule and strategies etc.

It outlines what, when, how, who and more of a testing project and contains the details of what the scope of testing is, what the test items are, who will do which testing task, what the items test/pass criteria will be and what is required to set up the test environment.

Test planning is the most important activity which lists the various tasks and milestones in a baseline plan to track the progress of the project.

It is the main document often called as master test plan or a project test plan which is developed during the early phase of the project.

IEEE 829 Test Plan Structure

IEEE 829-2008, also known as the 829 Standard for Software Test Documentation, specifies the standard form of a test plan, which is to be documented for performing testing. It consists of the following parameters :

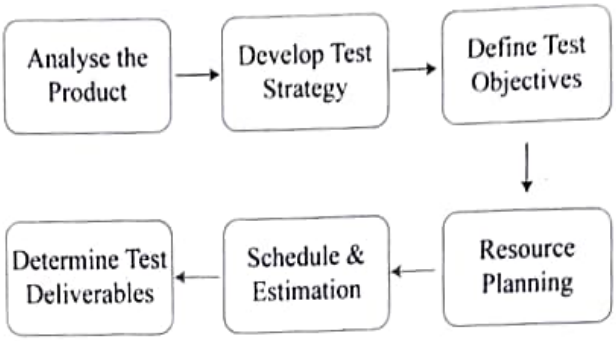
Sr. No.	Parameter	Description
1.	Test plan identifier	Unique identifying reference.
2.	Introduction	A brief introduction about the project and to the document.
3.	Test items	A test item is a software item that is the application under test.
4.	Features to be tested	A feature that needs to tested on the testware.

5.	Features not to be tested	Identify the features and the reasons for not including as part of testing.
6.	Approach	Details about the overall approach to testing.
7.	Item pass/fail criteria	Documented whether a software item has passed or failed its test.
8.	Test deliverables	The deliverables that are delivered as part of the testing process, such as test plans, test specifications and test summary reports.
9.	Testing tasks	All tasks for planning and executing the testing.
10.	Environmental needs	Defining the environmental requirements such as hardware, software, OS, network configurations and tools required.
11.	Responsibilities	Lists the roles and responsibilities of the team members.
12.	Staffing and training needs	Captures the actual staffing requirements and any specific skills and training requirements.
13.	Schedule	States the important project delivery dates and key milestones.
14.	Risks and Mitigation	High-level project risks and assumptions and a mitigating plan for each identified risk.
15.	Approvals	Captures all approvers of the document, their titles and the sign of date.

Test Planning Activities

- It determines the scope and the risks that need to be tested and NOT to be tested.
- Test Strategy is documented.
- It ensures that all the possible testing activities have been included.
- It identifies the Entry and Exit criteria.
- It evaluates the test estimate.
- Planning when and how to test and deciding how the test results will be evaluated, and defining test exit criterion.
- The Test artefacts delivered as part of test execution.
- Defining the management information, including the metrics required and defect resolution and risk issues.
- Ensuring that the test documentation generates repeatable test assets.

How to Write a Good Test Plan?



Step 1 : Analyze the test product

The first step for creating a test plan is to analyze the product, its features and functionalities by understanding the user's requirements and needs. This can be done through interview client, designer and developer, review product and projects and performing product walkthrough.

Step 2 : Develop Test Strategy

Once the product is analysed, a test strategy is designed which helps in defining the scope of testing. Test strategy outlines the different testing approach and defines the guidelines determining how testing needs to be performed.

Step 3 : Define Objectives

The objective of any project is to release a high-quality bug-free software product to market in time. Therefore, it is necessary to clearly define the testing scope and its boundaries. It can be done in following two steps to define test objectives :

- Enlist all the software features which may need testing.
- It defines the target of testing depending on previously mentioned features.

Step 4 : Resource Planning

The resource set up is essentially a close outline of all kinds of resources needed to complete the project task. The resources here may well be human, instrumentation and materials required to complete a project. It is a very important step that helps in identifying the amount of resources to be used for the project.

Step 5 : Develop a Schedule

With the knowledge of testing strategy and scope in hand, the next step is to develop a schedule for testing. Creating a schedule helps in managing the progress of the testing method. While writing up a schedule, following factors like :

- Project estimate
- Project risk estimate
- Resource estimate
- Employees role & responsibilities
- Test activity deadlines

Inaccurate or incomplete documentation

Error in programming language translation of design

Ambiguous or inconsistent human-computer interface

...us.

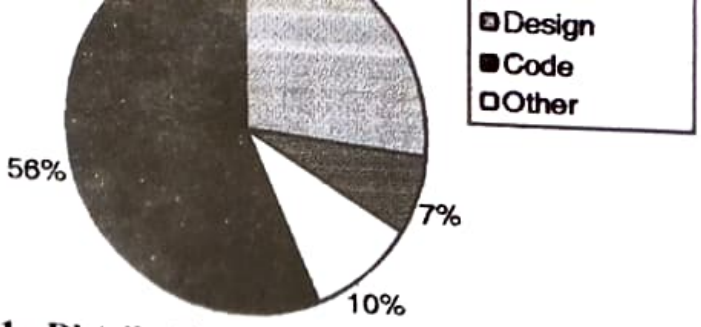


Fig. 10.1.1 : Distribution of Defects/Errors in the SDLC

...g. 10.1.1) defines the defects that are caused at different phases of Software Development ...ycle (SDLC) :

10.2 Quality Control and Quality Assurance

Q. State the difference between quality control and quality assurance.

Quality Assurance is the process of defining how software quality can be achieved and how the development organization knows that the software has the required level of quality.

Quality control and *Quality assurance* are the essential activities for any organization in the process of developing a product.

Table 10.2.1 : Software Quality Control v/s Software Quality Assurance

Sr. No.	Software Quality Control	Software Quality Assurance
1.	It is a *corrective approach* – corrects the faults when they occur.	It is an *preventive approach* – prevents the faults from occurring by providing rules and methods.
2.	*Finds defects* and then corrects it.	*Prevents defects* from occurring.
3.	It is a task conducted on the *product.*	It is a task conducted in the *process.*
4.	Gives confidence to *producer.*	Gives confidence to *customer.*
5.	Process of comparing the product quality with applicable standards and taking appropriate action when non-conformance is detected.	It is a set of planned and systematic set of activities that provides adequate confidence about establishing the requirements properly and assures that the products or services conform to specified requirements.
6.	Only tester is responsible for QC.	Entire team is responsible for QA.
7.	Detects and reports and corrects the defects.	Prevents the introduction of issues or defects.
8.	Examples: Walkthrough, inspections, testing conducted on the documents or on software product.	Examples : Defining process, automated engineering and testing tools i.e. CASE (engineering) and CAST (testing) tools, training, quality audits are used in development process.

10.3 SQA : Software Quality Assurance

- *SQA* serves as a customer's in-house representative that assists the software engineering team in achieving high-quality.
- SQA is considered as an *umbrella activity* that is applied throughout the SDLC process.
- SQA is a conformance to *explicitly stated software requirements* (functional requirements), *explicitly documented development standards, and implicit characteristics* (such as ease of use, maintainability, reliability, etc.) expected from the software.

10.3.1 Need of SQA

- SQA is a process that defines how software quality can be achieved and how the development organization knows that the software has the required level of quality.
- SQA group serves as the customer's in-house representative i.e. these people look at the software from customer's point of view.
- SQA monitors the software engineering processes and methods to ensure quality.
- SQA is a process of verifying or confirming that whether the products and services meet the customer expectations or not.
- SQA is a *process-driven approach* with specific steps to attain development goals. This process of QA considers design, development, production, and service.
- SQA group checks whether;
  - The software adequately meets the quality factors.
  - Software development has been conducted according to pre-established standards.
  - The technical disciplines (software engineers and testers) have properly performed their roles as part of the SQA activity.

Syllabus Topic : SQA Tasks

10.3.2 SQA Tasks

SQA is looked after by two different groups :

1. **Software engineers** who develop the software and have lot of technical knowledge.
   Software engineers address the quality by :
   - Applying effective technical methods and measures
   - Conducting formal technical reviews
   - Performing well-planned software testing
2. **Software quality assurance persons** who address the quality by :
   - Planning the quality assurance
   - Record keeping
   - Monitoring the development activities
   - Analysis and
   - Error reporting so as to improve the development process and prevent bugs from ever occurring.

The Software Engineering Institute (SEI) encompasses a set of SQA activities to evaluate the development process of the software product.

SQA activities

1. Prepare SQA plan for the project.

- The SQA plan is developed during project planning and after the requirement gathering.
- This SQA plan is then reviewed by all the stakeholders. Stakeholders are the persons who directly or indirectly interact with system such as customers, end-users, developers, testers, analysts, reviewers and etc.
- The SQA plan identifies :
  - Evaluations to be performed
  - Audits and reviews to be performed
  - Quality standards that are applicable to the project
  - Procedures for error reporting and tracking
- Documents to be produced by the SQA group
- Amount of feedback provided to the software project team

2. Participate in the development of the project's software process description.
   - The Software engineers select some process to develop the software and to help them, the SQA team reviews the process description to check:
     - The compliance of the development process with organizational policy
     - Internal software standards
     - Externally imposed standards
     - Other parts of the software project plan
3. Reviews the software engineering activities to verify it's compliance with the defined software process.

   The SQA team identifies, then documents and tracks the deviations from the process and verifies that the required corrections have been made.
4. Audits the designated software work products to verify their compliance with those defined as part of the software process.

☛ **SQA group audits**

- The software work product
- Identifies, documents and tracks the deviations
- Verifies that the needed corrections have been made and
- Periodically reports the reviewed results of the software work product to the project manager.

5. Ensures that deviations in software work and work products are documented and handled according to a document procedure.

   These deviations may be encountered in the project plan, process description, applicable standard or technical work products.
6. Records any non-compliance and reports to senior management.

   The SQA team keeps track of the non compliance things until they are resolved.

Syllabus Topic : Goals

10.3.3 SQA Goals

Q. State the goals of SQA.

☛ **Goals of SQA activities in terms of Software Development**

1. Assures that the software development will conform to the user requirements.
2. Assures that the software development will conform to planned schedules and budget.

3. Initiating and managing the activities to achieve greater efficiency of software development and SQA activities.

Goals of SQA activities in terms of Software Maintenance

1. Assures that the software maintenance activities will comply with the functional requirements.
2. Assures that the software maintenance activities will conform to planned schedules and budget.
3. Initiating and managing the activities to achieve greater efficiency of software maintenance and SQA activities

Syllabus Topic : Verification and Validation

11.1 Verification and Validation

V & V stands for Verification and Validation. It is a whole life-cycle process - it is applied at each stage of software development process.

- **Verification**
  - "Are we building the product right ?"
  - Software should conform to its specification (SRS). It is about confirming with the user specified requirements.
- **Validation**
  - "Are we building the right product ?"
  - Software should do what the user 'really requires'. It is about confirming with the proper functionality and performance of the system with good security of data.

Table 11.1.1 : Difference between Verification and Validation

Sr. No.	Verification	Validation
1.	It is the process of confirming whether the software meets its requirement specification.	It is the process of confirming whether the software meets user requirements.
2.	Inspections, walkthroughs and reviews are the examples of Verification.	Structural testing, black box testing, integration testing, system testing, acceptance testing are the examples of validation.

Sr. No.	Verification	Validation
3.	It is usually a process of static testing i.e. inspecting without executing on computer.	It is usually a process of dynamic testing i.e. testing by executing on computer.
4.	It is the process of confirming if the phases are completed correctly.	It is the process of determining if product as a whole satisfies the requirements.
5.	It is the process of examining the product to discover its defects.	It is the process of executing a product to expose its defects.
6.	It answers, "Are we building the product right"?	It answers, "Are we building the right product"?

V & V Goals

- To establish confidence that the software system is 'fit for purpose'. This does NOT mean system with zero defects. It ensures that system is good enough for its intended purpose.

This level of degree of confidence depends on

- ***Software's functions*** i.e. how useful the software is to an organisation.
- ***User expectations*** usually users have low expectations from the software and they are not at all surprised if the system fails during the use.
- ***Marketing environment*** While marketing the software product, the sellers must also consider the competing software, the price that the customers are willing to pay for the software and the required schedule for delivering the system.

Objectives of V&V Process

- Discover the defects in a system;
- Assess whether or not the system is useful.

Advantages of V and V Process

- Prevents faults and stops fault multiplication.
- Avoids the downward flow of errors.
- Earlier detection of errors leads to Lower defect Resolution cost.
- Improves quality and reliability.
- Reduces the amount of Re-work.
- Improves Risk Management
- Allows testers to be active in the early phases of the project's lifecycle.

11.10 Black Box vs. White Box

Q. Differentiate: whitebox and blackbox testing

Sr. No.	Back-box (Functional) Testing	White-box (Structural) Testing
1.	Black Box Testing is a software testing method in which the internal structure/ design/ implementation of the item being tested is NOT known to the tester	White Box Testing is a software testing method in which the internal structure/ design/ implementation of the item being tested is known to the tester.
2.	Knowledge of Programming language is not required	Deep knowledge of programming language is required
3.	Implementation knowledge is not required	Implementation knowledge is required
4.	The tester examines the application's external functional behavior and GUI features.	The tester has to correct the code also.
5.	done by independent Software Testers	done by developers
6.	Black box can be applicable at all levels but dominates at higher levels of testing.	White box dominates at lower levels.
7.	Requirement specification is the basis for conducting black-box test	Detail design specification is the basis for conducting white-box test
8.	in black box testing, sample test cases are selected and applied every time when required.	We have to write large quantity of test cases for white-box testing.

Syllabus Topic : Test Plan

11.11 Test Plan

- Test Plan tells **"What to Test"**
- The purpose of the Test Plan document is to :
  - Specify the approach to test the product
  - It specifies the deliverables extracted from the Test Approach.
  - Breakdown the product and identify features to be

Bug Life Cycle

Q. Define the terms : Bug life cycle

- Bugs can occur at any stage of SDLC process. The bug then attains different states which can be termed as its life cycle.
- The bug life cycle begins when the developer makes mistakes and the cycle ends when the bug is fixed and it is no longer in existence.
- The bug life cycle is shown diagrammatically as follows :

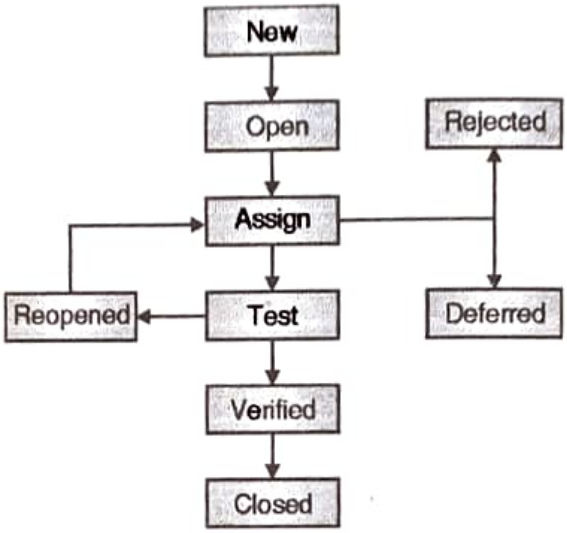


Fig. 11.2.2 : Bug Life Cycle

The different states of a bug in the can be described as follows :

1. **New :** When the bug is detected for the first time, its state is called as NEW. This means that the bug is not yet approved as a bug.
2. **Open :** After detecting the bug, the leader of the tester approves that the bug is genuine needs to be fixed; thus, he changes its state as OPEN.
3. **Assign :** After changing its state as OPEN, the test leader assigns the bug to corresponding developer or developer team. The state of the bug is now as ASSIGN.
4. **Test :** The developer then fixes the bug and re-assigns the bug to the testing team for next round of testing. The state of the bug is now as TEST. It specifies that the bug has been fixed and is released to testing team for re-testing.
5. **Deferred :** The bug changed to DEFERRED state means that the bug is expected to be fixed in next releases. The reasons for changing the bug to this state might be that the priority of the bug may be low, or lack of time in release of the software or the bug may not have major affect on the software.
6. **Rejected :** If the developer feels that the bug is not genuine, he rejects the bug. Then the state of the bug is now as REJECTED.
7. **Duplicate :** If the same bug repeats twice then that bug status is changed to DUPLICATE.
8. **Verified :** Once the bug is fixed, the tester tests the bug. If the bug is no more present, the tester changes its state to VERIFIED.
9. **Reopened :** If the bug still exists in the software even after the bug is fixed by the developer, then the tester changes its state to REOPENED. The bug traverses through the bug life cycle once again by assigning it to the developer.

10. **Closed :** Once the bug is fixed, it is tested by the tester and if he feels that the bug no longer exists in the software and it will not even crop up in the future, then he changes the bug state to CLOSED which means that the bug is fixed, tested and approved.

The tester or the developer decides the severity of the bug on the basis of the priorities assigned to the bugs. ***There are different types of priorities assigned to the bugs.***

1. **Critical** (Show) Stop

Use case diagram for car rental system :

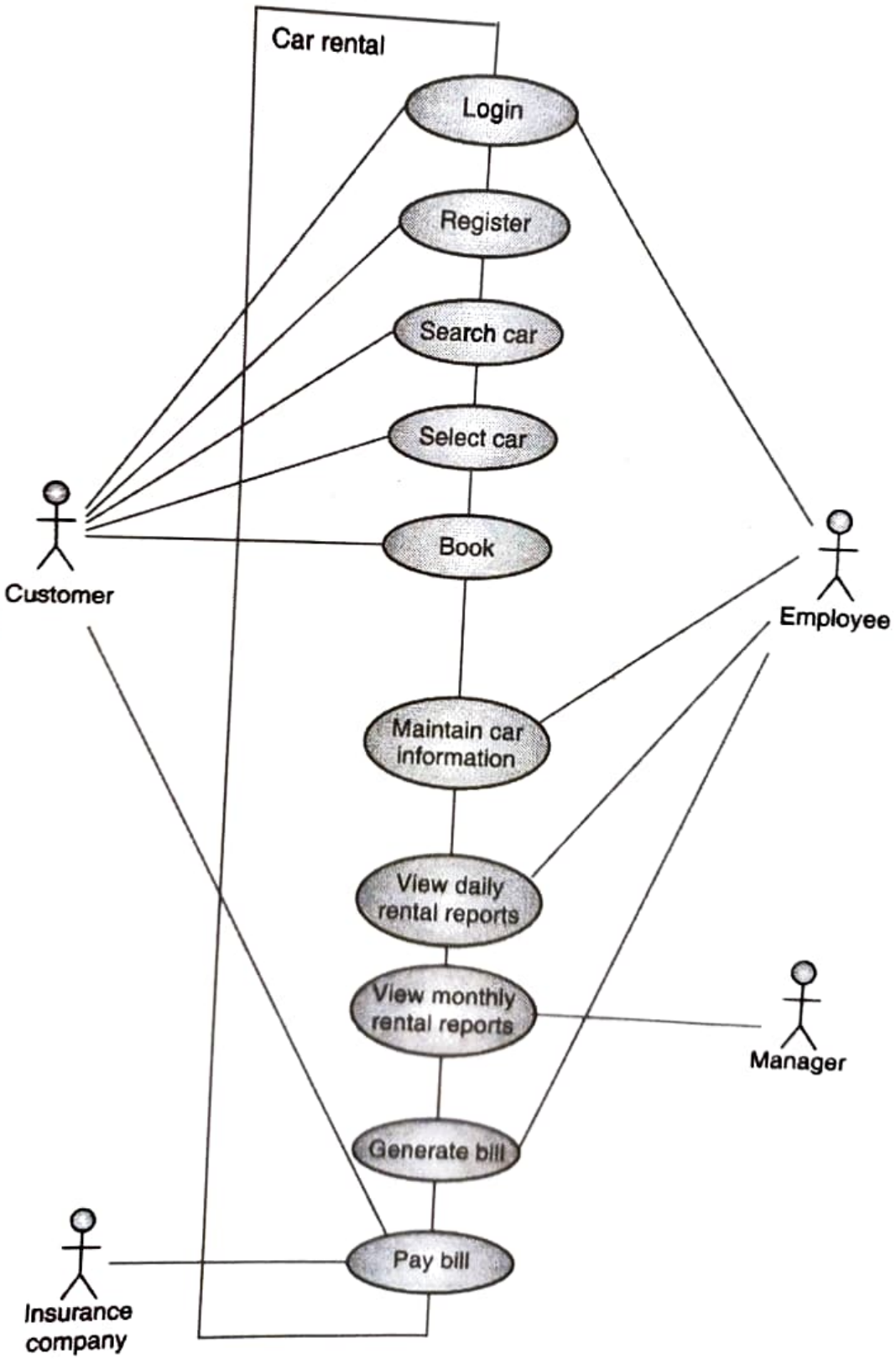


Fig. 1 Q. 2(b) : Use case diagram for car rental