Software Engineering

CHAPTER 1

According to the chapter Wise Question and Answer.

Introduction: Software Engineering

Definition of software Engineering:
Software:

- **it is more than just a program code.**
- **A program is an executable code, which serves some computational purpose.**
- **It is collection of executable programming code, associated libraries and documentations.**

Engineering:

- **Developing products, using well-defined, scientific principles and methods.**

Software product:

Software Engineering:
Is an engineering branch associated with development of software product using well-defined scientific principles, methods and procedure
The outcome of software engineering is an efficient and reliable software product.

ACCORDING TO IEEE:
IEEE= Institute of Electrical and Electronics Engineers
The application of a systematic, disciplined quantifiable approach to the development, operation and maintenance of software; that is application of engineering to software

Fritz Bauer, a German computer scientist, defines software engineering as:
->Software engineering is the establishment and use of sound engineering principles in order to obtain economically software that is reliable and work efficiently on real machines.

Evolving Role of Software: IN SUMMARY

- Requirement gathering process,
- Developer create a prototype and show to user(customers) and take feedback
- Changes, updates, maintenance keep on changing too until this process leads to original software till the desired software is accomplished
- After handover to user maintenance in time ,update the existing software that matches the latest requirements.

Evolving Role of Software:

Software Evolution

The process of developing a software product using software engineering principles and methods is referred to as Software Evolution. This includes the initial development of software and its maintenance and updates, till desired software product is developed, which satisfies the expected requirements.

Evolution starts from the requirement gathering process. After which developers create a prototype of the intended software and show it to the users to get their feedback at the early stage of the software product development. The users suggest changes, on which several consecutive updates and maintenance keep on changing too. This process changes to the original software, till the desired software is accomplished. Even after the user has the desired software in hand, the advancing technology and the changing requirements force the software product to change accordingly. Re-creating software from scratch and to go one-on-one with the requirement is not feasible. The only feasible and economical solution is to update the existing software so that it matches the latest requirements.

Changing Nature of Software:

Nowadays, seven broad categories of computer software present continuing challenges for software engineers, which is given below:

1.System Software:

System software is a collection of programs which are written to service other programs. Some system software processes complex but determinate, information structures. Other system application process largely indeterminate data. Sometimes when, the system software area is characterized by the heavy interaction with computer hardware that requires scheduling, resource sharing, and sophisticated process management.

2.Application Software:

Application software is defined as programs that solve a specific business need. Application in this area process business or technical data in a way that facilitates business operation or management technical decision making. In addition to convention data processing application, application software is used to control business function in real time.

3.Engineering and Scientific Software:

This software is used to facilitate the engineering function and task. however modern application within the engineering and scientific area are moving away from the conventional numerical algorithms. Computer-aided design, system simulation, and other interactive applications have begun to take a real-time and even system software characteristic.

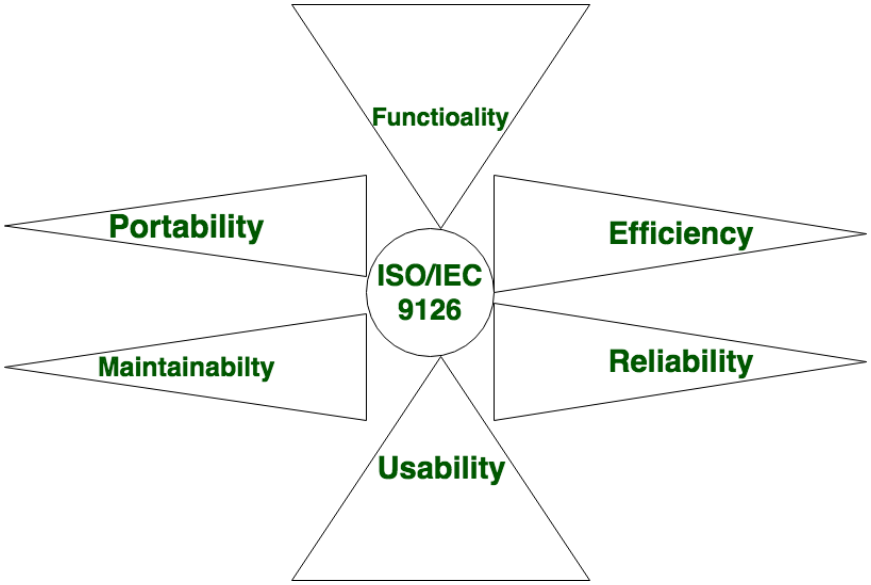
4.Embedded Software:

Embedded software resides within the system or product and is used to implement and control feature and function for the end-user and for the system itself. Embedded software can perform the limited and esoteric function or provided significant function and control capability.

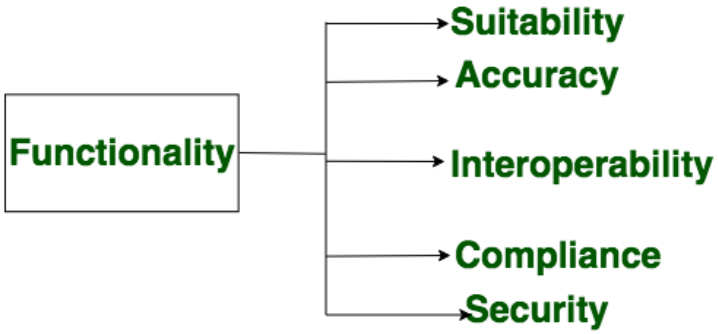
5.Product-line Software:
Designed to provide a specific capability for use by many different customers, product line software can focus on the limited and esoteric marketplace or address the mass consumer market.
6.Web Application:
It is a client-server computer program which the client runs on the web browser. In their simplest form, Web apps can be little more than a set of linked hypertext files that present information using text and limited graphics. However, as e-commerce and B2B application grow in importance. Web apps are evolving into a sophisticate computing environment that not only provides a standalone feature, computing function, and content to the end user.
7.Artificial Intelligence Software:
Artificial intelligence software makes use of a nonnumerical algorithm to solve a complex problem that is not amenable to computation or straightforward analysis. Application within this area includes robotics, expert system, pattern recognition, artificial neural network, theorem proving and game playing.

CHARACTERISTICS OF SOFTWARE

Software is defined as collection of computer programs, procedures, rules and data. Software Characteristics are classified into six major components:

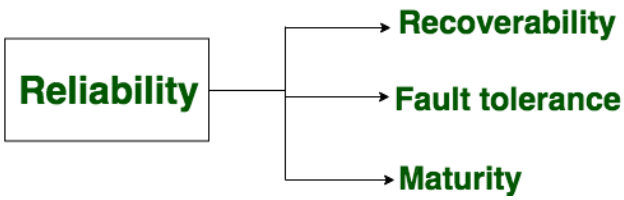


1.Functionality:
It refers to the degree of performance of the software against its intended purpose.
Required functions are:

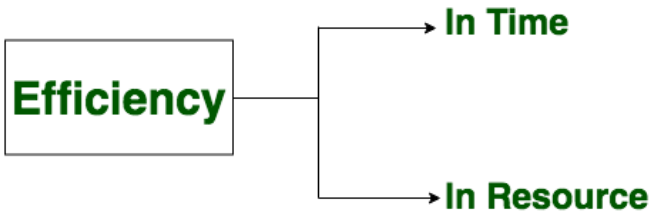


2.Reliability:
A set of attributes that bear on capability of software to maintain its level of performance under the given condition for a stated period of time.

Required functions are:

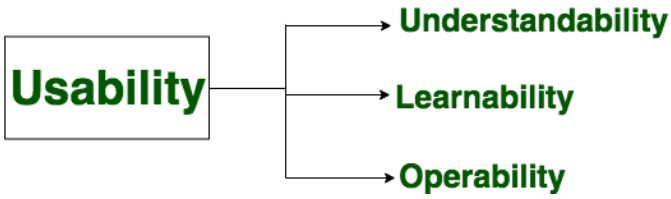


3.Efficiency:
It refers to the ability of the software to use system resources in the most effective and efficient manner. the software should make effective use of storage space and executive command as per desired timing requirement.
Required functions are:

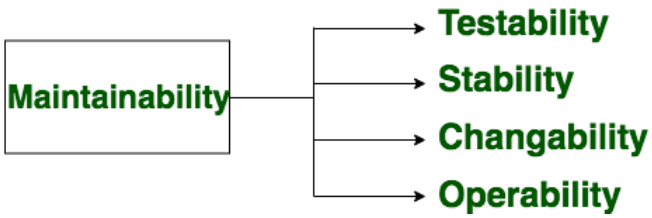


4.Usability:
It refers to the extent to which the software can be used with ease the amount of effort or time required to learn how to use the software.

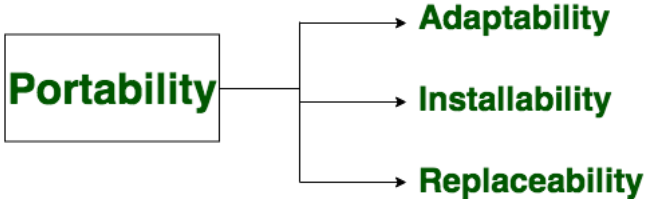
Required functions are:



5.Maintainability:
It refers to the ease with which the modifications can be made in a software system to extend its functionality, improve its performance, or correct errors.
Required functions are:



6.Portability:
A set of attribute that bear on the ability of software to be transferred from one environment to another, without or minimum changes.
Required functions are:



Generic View of Software Engineering

The process of a software development has three Generic views which are:

1. **Definition Phase** - It is the base of Definition phase. The experts get the knowledge about "What".
    - Information needed for processing.
    - Which functions are required?
    - Expectations about the capacity.
    - Interface which is established.
    - Area of the validation.

    This phase defines all the expectations depending on the standard of the software Engineering. It contains three steps.
    - Analysis of system
    - Planning of project
    - Requirement Analysis
2. **Development phase** - Focus point of development phase is "How". After the explanation of "What" it turns to "How". Various type of question raised in developer mind that how to design the data structure and Architecture of software, Procedural detail how to implemented and how design convert in a programming language and testing of software how to perform. Three special steps always taken in this phase which are
    - Design of software
    - Coding
    - testing of software system
3. **Maintenance phase** - The main focus of maintenance phase is change which cause is correction of errors, adaption of new idea, According to the needs of software after change in customer mood.

Software Engineering-Layered Technology

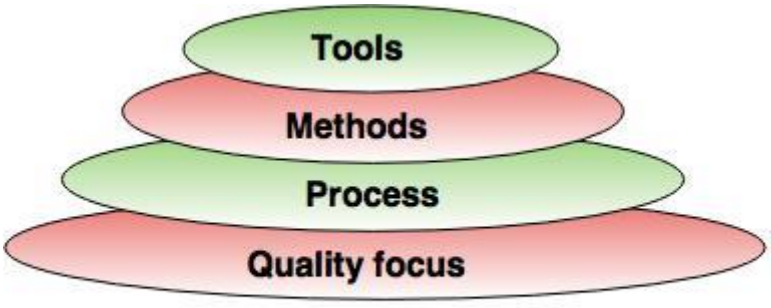


Fig. - Software Engineering Layers

The layered technology consists of:

1. Quality focus

The characteristics of good quality software are:

Correctness of the functions required to be performed by the software.

Maintainability of the software
Integrity i.e. providing security so that the unauthorized user cannot access information or data.
Usability i.e. the efforts required to use or operate the software.

Every organization is rest on its commitment to quality. Total quality management, six sigma, or similar continuous improvement culture and it is this culture ultimately leads to development of increasingly more effective approaches to software engineering.

2. Process

It is the base layer or foundation layer for the software engineering.
The software process is the key to keep all levels together.
It defines a framework that includes different activities and tasks.
In short, it covers all activities, actions and tasks required to be carried out for software development.

It is a foundation layer for software engineering. It defines framework for a set of key process areas (KPA) for effectively manage and deliver quality software in a cost-effective manner. The processes define the task to be performed and the order in which they are to be performed.

3. Methods

The method provides the answers of all 'how-to' that are asked during the process.
It provides the technical way to implement the software.
It includes collection of tasks starting from communication, requirement analysis, analysis and design modelling, program construction, testing and support.
It provides the technical how-to's for building software.
Methods encompass a broad array of tasks that include requirements analysis, design, program construction, testing, and support.
There could be more than one technique to perform a task and different technique could be used in different situations

4. Tools

The software engineering tool is an automated support for the software development.
The tools are integrated i.e. the information created by one tool can be used by the other tool.
For example: The Microsoft publisher can be used as a web designing tool.
Provide automated or semi-automated support for the process, methods and quality control.
When tools are integrated so that information created by one tool can be used by another, a system for the support of software development, called computer-aided software engineering (CASE)

QUESTIONS

2015 Q.N.9) A
What is software Engineering? Discuss professional responsibilities.

2018 Q.N.) 4
What is software? Explain the quality of good software.

What are characteristic of software? Explain the dual role of software.

Process Model

A software process is a set of activities that leads to the production of software product. These activities may involve the development of software from scratch in a standard programming language.

1.**The Waterfall Model**

Waterfall model is the simplest model of software development paradigm. All the phases of SDLC will function one after another in linear manner. That is, when the first phase is finished then only the second phase will start and so on.

This model assumes that everything is carried out and taken place perfectly as planned in the previous stage and there is no need to think about the past issues that may arise in the next phase. This model does not work smoothly if there are some issues left at the previous step. The sequential nature of model does not allow us to go back and undo or redo our actions. This model is best suited when developers already have designed and developed similar software in the past and are aware of all its domains.

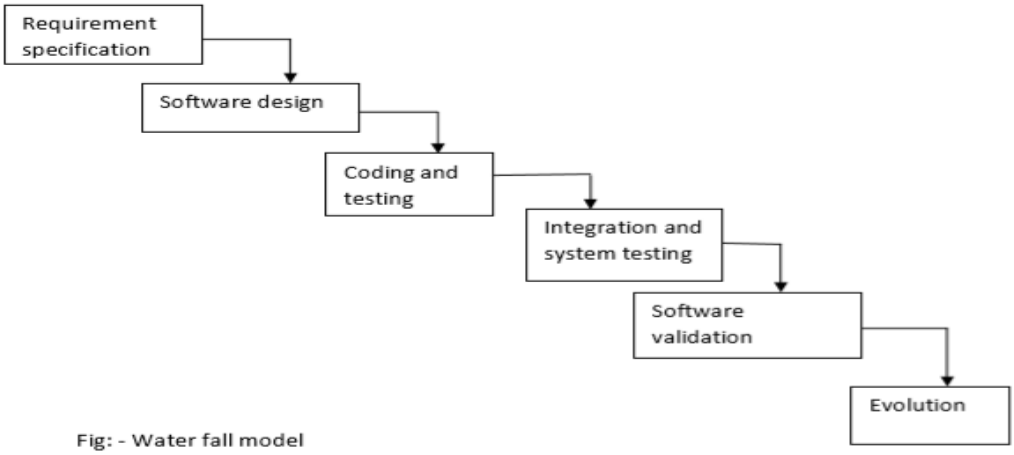


Fig: - Water fall model

In the waterfall model, the fundamental process activity: specification, development, validation, and evolution are represented as a separate process phases such as requirement specification, software design, coding, and testing, integration and system testing, implementation and evolution. In waterfall model, we complete one phase before going to next phases and we cannot return back to the previous phase.

1.Requirement Specification The system services, constraint and goals are established by consultation with the system user. User's requirements are identified and analyzed. At the end of this phase, a written document is generated called requirement specification. This phase consists two activities: requirement gathering and analysis and requirement specification. The goal of the requirement gathering activity is to collect all relevant information from the user regarding the product to be developed. The data collected from the users usually contains several contradictions since each user typically has only a partial and incomplete view of the system. Therefore, it is necessary to identify contradiction and ambiguities in the requirements and resolve them through further discussion with the customer. After that the requirement specification activity can start. During this activity, the user's requirements are systematically organized into a software requirement specification document.

2. Software design

The goal of the software design phase is to transform the requirement specification into a structure that is suitable for

implementation in some programming language. In technical term, during the design phase, the software architecture is derived from the software requirement specification document. Software design involves identifying and describing the software components and their relationship. During this phase, the algorithm and flow chart for each component of software are designed.

3. Coding and Testing

The procedure for coding and unit testing phase of the software development is to translate the software design into source code of programming language. Each component of the design is implemented as a program module. The end product of this phase is s set of programs that has been tested individually. During this phase, end module is tested to determine the current functioning of the individual module and remove error is any. Unit testing is performed to verify that each unit needs its specification.

4. Integration and System Testing

Interpretation of different module is undertaken, once they have been coded and tested individually. During the integration and system testing phase, the modules are integrated in a planned manner. The different modules making up a software product an almost never integrated into one shot. Integration is normally carried out inclemently over a number of steps. During each integration step, the partially integrated system is tested and shapes of previously planned modules are added with. Finally, after all the modules have been successfully integrated and tested, system testing is carried out.

The goal of system testing is to ensure that the developed software confirms to its requirement specification. System testing is normally carried out in a planned manner according to the system test planned document. The system test plan document identifies all testing related activities that must be performed, specify the schedule of testing and allocated required resources.

5. Implementation

After testing the s**ys**tem, the software is implemented for operational use. During the software implementation phase, the software is installed in a suitable machine environment according to the implementation phase. During implementation phase, the file conversion activities are performed.

6. Evaluation

After a reasonable period of time, the system is evaluated to improve its efficiency and performance. During this phase, the system is changed or modified to adopt the changing requirement of user and organization.

Advantages of Waterfall model The advantage of waterfall model is that documentation is produced at each phase and that is fit with other engineering process models. Its major problem is it's inflexible partitioning of the software development process into distinct stage. Commitments must be made at an early stage in the process which makes it difficult to respond to changing customer requirement

Therefore, the waterfall model should only be used when the requirements are well understood.

Advantages of Waterfall model

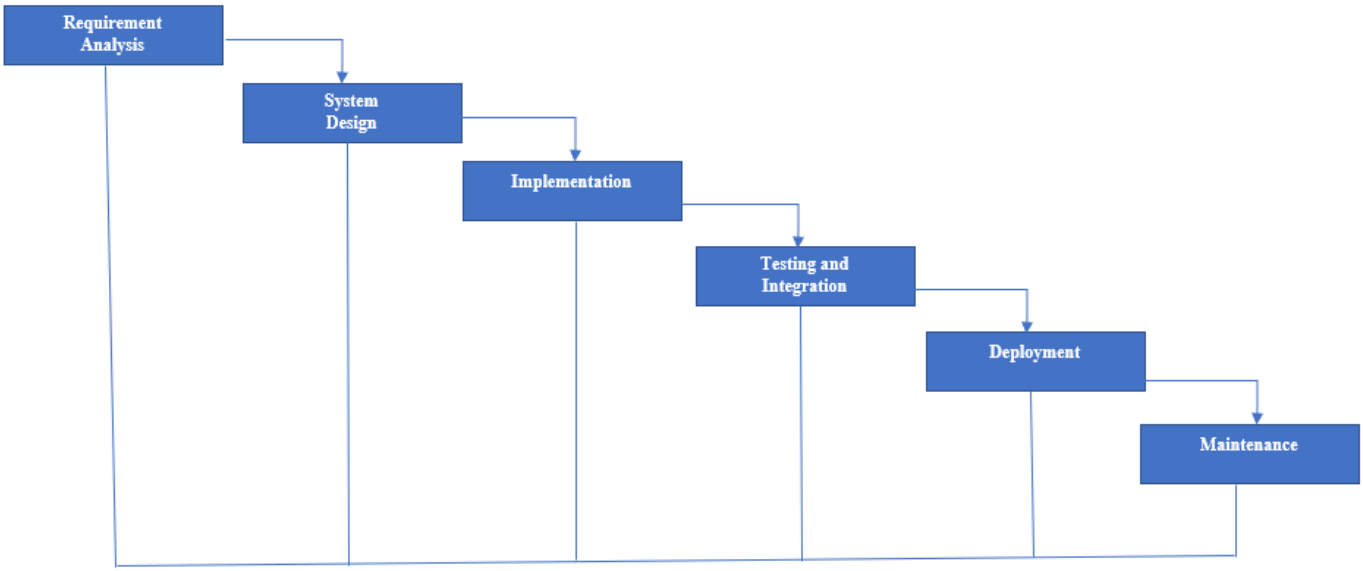
a) This model is simple to implement also the number of resources that are required for it is minimal.
b) The requirements are simple and explicitly declared; they remain unchanged during the entire project development.
c) The start and end points for each phase is fixed, which makes it easy to cover progress.
d) The release date for the complete product, as well as its final cost, can be determined before development.
e) It gives easy to control and clarity for the customer due to a strict reporting system.

Disadvantages of Waterfall model

a) In this model, the risk factor is higher, so this model is not suitable for more significant and complex projects.
b) This model cannot accept the changes in requirements during development.

c) It becomes tough to go back to the phase. For example, if the application has now shifted to the coding phase, and there is a change in requirement, It becomes tough to go back and change it.
d) Since the testing done at a later stage, it does not allow identifying the challenges and risks in the earlier phase, so the risk reduction strategy is difficult to prepare.

OR FROM YOUTUBE:



In waterfall model the iteration is completely done and jump to next iteration
at first requirement analysis: the data, information from customer is collected, then after collecting requirement is analysis and developer design a system then when complete implementation means make a small program called unit and tested each unit then this these tested unit are integrated to a single system and tested for a final product. After every functional or nonfunctional testing is done Then this product is deployed or handover to customer or release in a market. Maintenance after publishing the developer maintain the software product quality.

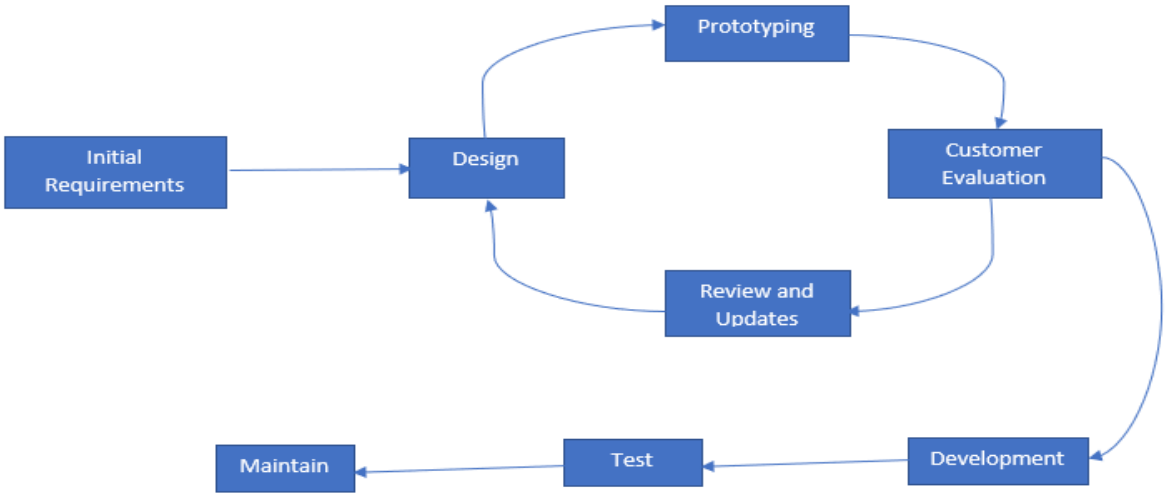
Advantage

1.Easy to understand, manage
2.Phases are proceeded and completed one at a time.

Disadvantage

1.high amount of risk
2. Complex and not for object-oriented program
3. Poor model for large and ongoing project.

Prototyping Model



Process: At first raw data , information requirement is collected then developer design a software product and make prototype of product which will be shown to customer, customer evaluate the product ,any things they don't like want to add features they review and update the developer again developer start to design and make prototype of the product and show to customer if customer is not satisfied again customer gives review and updates to developer and again start from designing this process is repeated until customer satisfied for prototype model when customer satisfy then the developer start to code a program then test it, and hand over the software product to customer. And further they maintain the product quality.

Advantages:

1. customer can see a steady progress of product.
2. Useful when customer requirements are changing rapidly.

Disadvantages:

1. Impossible to know how long it will take.
2. There is no way to know the number of iterations required.

RAD model

RAD or Rapid Application Development process is an adoption of the waterfall model; it targets at developing software in a short span of time.

RAD is a linear sequential software development process model that emphasizes a concise development cycle using an element-based construction approach. If the requirements are well understood and described, and the project scope is a constraint, the RAD process enables a development team to create a fully functional system within a concise time period.

RAD (**Rapid Application Development**) is a concept that products can be developed faster and of higher quality through:

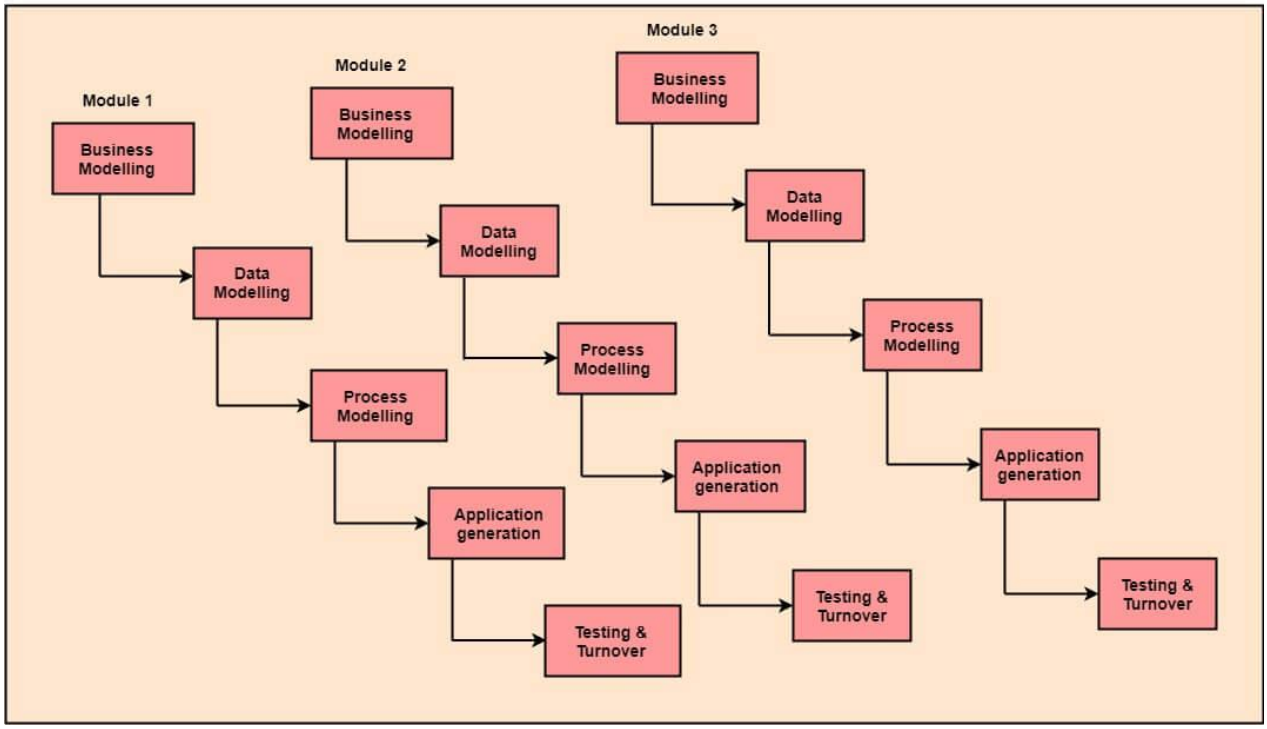
- Gathering requirements using workshops or focus groups
- Prototyping and early, reiterative user testing of designs
- The re-use of software components
- A rigidly paced schedule that refers design improvements to the next product version
- Less formality in reviews and other team communication

RAD follow the iterative

a) Business Modeling
b) Data Modeling
c) Process Modeling
d)Application Generation
e)Testing and Turnover

It focuses on input-output source and destination of the information. It emphasizes on delivering projects in small pieces; the larger projects are divided into a series of smaller projects. The main features of RAD model are that it focuses on the reuse of templates, tools, processes, and code.

Fig: RAD Model



The various phases of RAD are as follows:

1.Business Modelling: The information flow among business functions is defined by answering questions like what data drives the business process, what data is generated, who generates it, where does the information go, who process it and so on.

2. Data Modelling: The data collected from business modeling is refined into a set of data objects (entities) that are needed to support the business. The attributes (character of each entity) are identified, and the relation between these data objects (entities) is defined.

3. Process Modelling: The information object defined in the data modeling phase are transformed to achieve the data flow necessary to implement a business function. Processing descriptions are created for adding, modifying, deleting, or retrieving a data object.

4. Application Generation: Automated tools are used to facilitate construction of the software; even they use the 4th GL techniques.

5. Testing & Turnover: Many of the programming components have already been tested since RAD emphasis reuse. This reduces the overall testing time. But the new part must be tested, and all interfaces must be fully exercised.

When to use RAD Model?

- When the system should need to create the project that modularizes in a short span time (2-3 months).
- When the requirements are well-known.
- When the technical risk is limited.
- When there's a necessity to make a system, which modularized in 2-3 months of period.
- It should be used only if the budget allows the use of automatic code generating tools.

Advantages

1.Flexible and adaptable to changes
2.It is useful when you have to reduce the overall project risk
3.It is adaptable and flexible to changes
4.It is easier to transfer deliverables as scripts, high-level abstractions and intermediate codes are used
5.Due to code generators and code reuse, there is a reduction of manual coding
6.Due to prototyping in nature, there is a possibility of lesser defects
7.Each phase in RAD delivers highest priority functionality to client
8.With less people, productivity can be increased in short time

Disadvantages

1.It can't be used for smaller projects
2.Not all application is compatible with RAD
3.When technical risk is high, it is not suitable
4.If developers are not committed to delivering software on time, RAD projects can fail
5.Reduced features due to time boxing, where features are pushed to a later version to finish a release in short period
6.Reduced scalability occurs because a RAD developed application begins as a prototype and evolves into a finished application
7.Progress and problems accustomed are hard to track as such there is no documentation to demonstrate what has been done
8.Requires highly skilled designers or developers
9.It is incremental model that emphasis on extremely short development cycle.

Spiral Model

The spiral model, initially proposed by Boehm, is an evolutionary software process model that couples the iterative feature of prototyping with the controlled and systematic aspects of the linear sequential model. It implements the potential for rapid development of new versions of the software. Using the spiral model, the software is developed in a series of incremental releases. During the early iterations, the additional release may be a paper model or prototype. During later iterations, more and more complete versions of the engineered system are produced.

Each cycle in the spiral is divided into four parts:

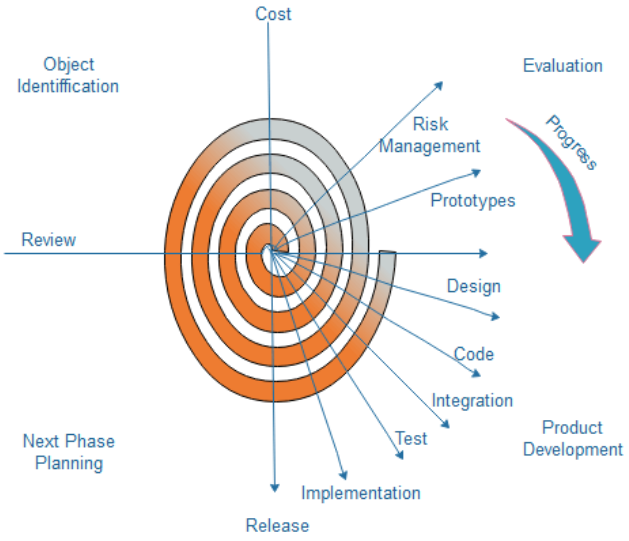


Fig. Spiral Model

1) **Objective setting:** Each cycle in the spiral starts with the identification of purpose for that cycle, the various alternatives that are possible for achieving the targets, and the constraints that exists.
2) **Risk Assessment and reduction:** The next phase in the cycle is to calculate these various alternatives based on the goals and constraints. The focus of evaluation in this stage is located on the risk perception for the project.
3) **Development and validation:** The next phase is to develop strategies that resolve uncertainties and risks. This process may include activities such as benchmarking, simulation, and prototyping.
4) **Planning:** Finally, the next step is planned. The project is reviewed, and a choice made whether to continue with a further period of the spiral. If it is determined to keep, plans are drawn up for the next step of the project.

The development phase depends on the remaining risks. For example, if performance or user-interface risks are treated more essential than the program development risks, the next phase may be an evolutionary development that includes developing a more detailed prototype for solving the risks.
The **risk-driven** feature of the spiral model allows it to accommodate any mixture of a specification-oriented, prototype-oriented, simulation-oriented, or another type of approach. An essential element of the model is that each period of the spiral is completed by a review that includes all the products developed during that cycle, including plans for the next cycle. The spiral model works for development as well as enhancement projects.

When to use Spiral Model?
-When deliverance is required to be frequent.
-When the project is large
-When requirements are unclear and complex
-When changes may require at any time
-Large and high budget projects
Advantages
-High amount of risk analysis
-Useful for large and mission-critical projects.
Disadvantages
-Can be a costly model to use.
-Risk analysis needed highly particular expertise
-Doesn't work well for smaller projects.

SOFTWARE PROJECT MANAGEMENT

Syllabus:

PEOPLE
PRODUCT
PROCESS,
PROJECT IN SOFTWAREPROJECT MANAGEMENT
ACTIVITY OF PROJECT PLANNING
PROJECT ESTIMATION TECHNIQUE
COCOMO,
RISK MANAGEMENT
PROJECT SCHEDULING
STAFFING

The four P's

1.people- the most important element of successful project
2.Product- the software to be built
3.Process- the set of frameworks of activities and software engineering tasks to get the job done
4.Project- all work required to make the product a reality

1.PEOPLE

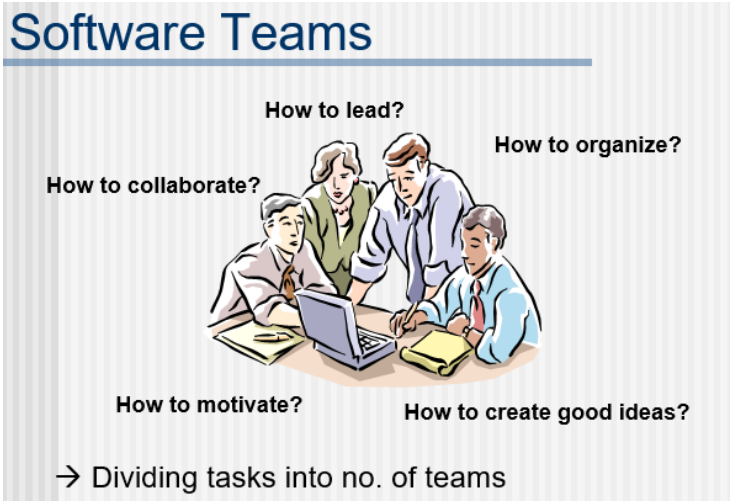
The People Factor is so important that Software Engineering institution has developed a People Management Capability Maturity Model. (PM-CMM). • PM-CMM Defines the following key practice areas for Software People: -

-Recruiting
- Selection
- Performance Management
- Training
- Compensation
- Career Development
- Organization and work Design
- Team Culture Development

Organizations that achieve high level of Maturity in People Management area have a higher likelihood of implementing effective Software Engineering Practices.

PEOPLE (Stakeholders)

a. Senior managers who define the business issues that often have significant influence on the project.
b. Project (technical) managers who must plan, motivate, organize, and control the practitioners who do software work.
c. Practitioners who deliver the technical skills that are necessary to engineer a product or application.
d. Customers who specify the requirements for the software to be engineered and other stakeholders who have a peripheral interest in the outcome.
e. End-users who interact with the software once it is released for production use.



Software Teams

- **How to lead?**
- **How to collaborate?**
- **How to motivate?**
- **How to organize?**
- **How to create good ideas?**

The MOI Model

- **Motivation:** The ability to encourage (by "push or pull") technical people to produce to their best ability.
- **Organization:** The ability to mold existing processes (or invent new ones) that will *enable the initial concept to be translated into a final product.*
- **Ideas or innovation:** The ability to encourage people to create and feel creative even when they must work within bounds established for a particular software product or application.

Software Teams (selecting sw eng. team)

The following factors must be considered when selecting a software project team structure ...

- the difficulty of the problem to be solved
- the size of the resultant program(s) **in *lines of code* or *function points***
- the time that the team will stay together **(*team lifetime*)**
- the degree to which the problem can be ***modularized***
- the required ***quality and reliability*** of the system to **be *built***
- the rigidity of the ***delivery date***
- the degree of ***sociability (communication)*** required for the project

2. Product — the software to be built (product objectives, scope, alternative solutions, constraint tradeoffs)

The Product Scope (Software Scope)

2.1 Context: How does the software to be built fit into a larger system, product, or business context.

2.3 Information objectives

-What customer-visible data objects are produced as output from the software?
-What data objects are required for input?

2.3 Function and performance.

-What function does the software perform to transform input data into output?
-Are any special performance characteristics to be addressed?
-Software project scope must be **unambiguous and understandable** at the management and technical levels**.**

Once product scope is defined …

-It is decomposed into **constituent functions**

-It is decomposed into **user-visible data objects** *or* into a **set of problem classes**
-Decomposition process continues until all functions or problem classes have been defined

3)Process — the set of framework activities and software engineering tasks to get the job done (framework activities, milestones, work products and QA points)
a) Linear Sequential Model
b) Prototyping Model
c) Incremental Model
d) RAD Model
e) Spiral Model
f) And other…

Once a process framework has been established
Consider project characteristics
Define a task set for each software engineering activity
Task set =

a) Software engineering tasks
b) Work products
c) Quality assurance points
d) Milestones

4) Project — all work required to make the product a reality (planning, monitoring, controlling)
Projects get into trouble when …

a) Software people don't understand their customer's needs.
b) The product scope is poorly defined.
c) Changes are managed poorly.
d) The chosen technology changes.
e) Business needs change [or are ill-defined].
f) Deadlines are unrealistic.
g) Users are resistant.
h) Sponsorship is lost [or was never properly obtained].
i) The project team lacks people with appropriate skills.
j) Managers [and practitioners] avoid best practices and lessons learned.

Common-Sense Approach to Projects
a) Start on the right foot. This is accomplished by working hard (very hard) to understand the problem that is to be solved and then setting realistic objectives and expectations.
b) Maintain momentum. The project manager must provide incentives to keep turnover of personnel to an absolute minimum, the team should emphasize quality in every task it performs, and senior management should do everything possible to stay out of the team's way.
c) Track progress. For a software project, progress is tracked as work products (e.g., models, source code, sets of test cases) are produced and approved (using formal technical reviews) as part of a quality assurance activity.
d) Make smart decisions. In essence, the decisions of the project manager and the software team should be to "keep it simple."
e) Conduct a postmortem analysis. Establish a consistent mechanism for extracting lessons learned for each project.

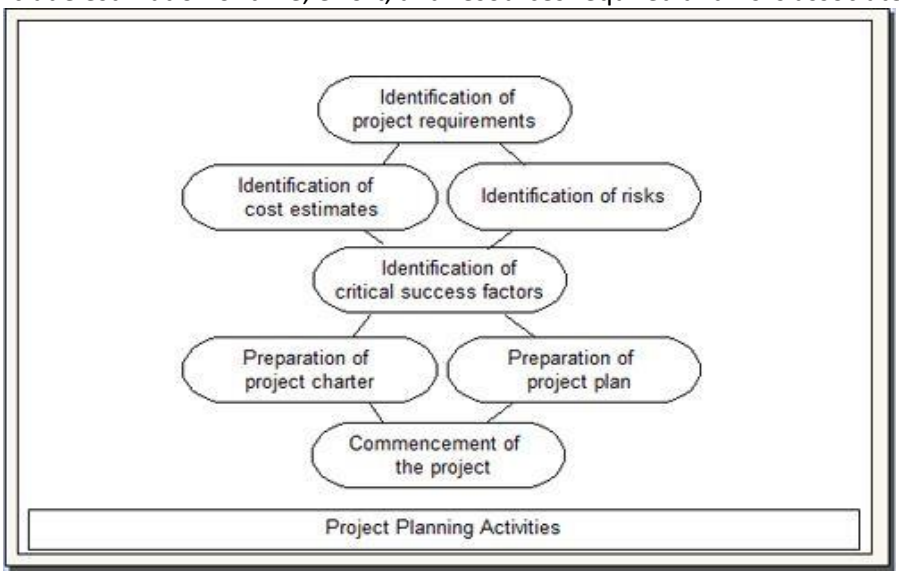
ACTIVITY OF PROJECT PLANNING

The project planning process involves a set of interrelated activities followed in an orderly manner to implement user requirements in software and includes the description of a series of project planning activities and individual(s) responsible for performing these activities.
In addition, the project planning process comprises the following.

1. Objectives and scope of the project
2. Techniques used to perform project planning
3. Effort (in time) of individuals involved in project
4. Project schedule and milestones

5. Resources required for the project
6. Risks associated with the project.

Project planning process comprises several activities, which are essential for carrying out a project systematically. These activities refer to the series of tasks performed over a period of time for developing the software. These activities include estimation of time, effort, and resources required and risks associated with the project.



Project planning process consists of the following activities.

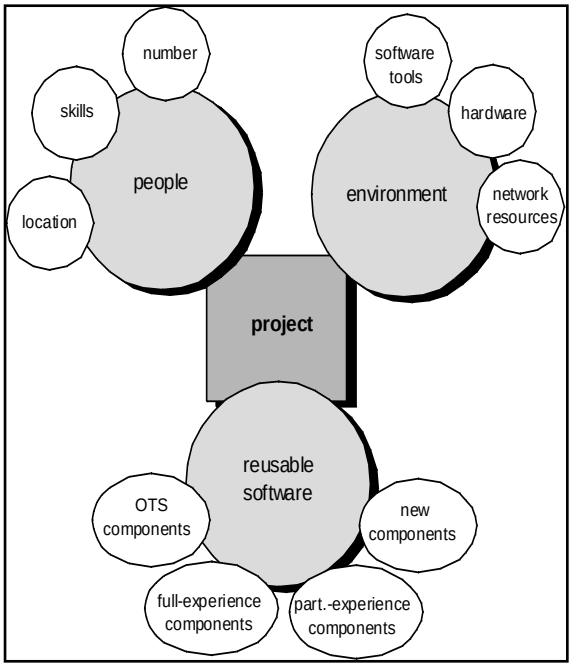
1. **Identification of project requirements:** Before starting a project, it is essential to identify the project requirements as identification of project requirements helps in performing the activities in a systematic manner. These requirements comprise information such as ***project scope, data and functionality required in the software, and roles of the project management team members.***
2. **Identification of cost estimates:** Along with the estimation of effort and time, it is necessary to estimate the cost. The cost estimation includes *the* ***cost of hardware, network connections, and the cost required for the maintenance of hardware components.***
3. **Identification of risks:** ***Risks are unexpected events*** that have an adverse effect on the project. Software project involves several risks (like ***technical risks and business risks***) that affect the project schedule and increase the cost of the project.
4. **Identification of critical success factors:** For making a project successful, critical success factors are followed. These factors refer to the conditions that ensure greater chances of success of a project. Generally, these ***factors include support from management, appropriate budget, appropriate schedule, and skilled software engineers*.**
5. **Preparation of project charter:** A project charter provides a ***brief description of the project scope, quality, time, cost, and resource constraints*** as described during project planning. It is prepared by the management for approval from the sponsor of the project.
6. **Preparation of project plan:** A project plan provides information about the resources that are available for the project, individuals involved in the project, and the schedule according to which the project is to be carried out.
7. **Commencement of the project:** Once the project planning is complete and resources are assigned to team members, the software project commences.

Once the project objectives and business objectives are determined, the project end date is fixed.
The project management team prepares the project plan and schedule according to the end date of the project. After analyzing the project plan, the project manager communicates the project plan and end date to the senior management. The progress of the project is reported to the management from time to time. Similarly, when the project is complete, senior management is informed about it. In case of delay in completing the project, the project plan is re-analyzed and corrective actions are taken to complete the project. The project is tracked regularly and when the project plan is modified, the senior management is informed.

Project Plan

As stated earlier, a project plan stores the outcome of project planning. It provides information about the end date, milestones, activities, and deliverables of the project. A project plan helps a project manager to understand, monitor, and control the development of software project. This plan is used as a means of communication between the users

and project management team. There are various advantages associated with a project plan, some of which are listed below.



A) It ensures that software is developed according to the user requirements, objectives, and scope of the project.
B) It identifies the role of each project management team member involved in the project.
C) It monitors the progress of the project according to the project plan.
D) It determines the available resources and the activities to be performed during software development.
E) It provides an overview to management about the costs of the software project, which are estimated during project planning.

A typical project plan is divided into the following sections.

1) Introduction: Describes the objectives of the project and provides information about the constraints that affect the software project.
2) Project organization: Describes the responsibilities assigned to the project management team members for completing the project.
3) Risk analysis: Describes the risks that can possibly arise during software development as well as explains how to assess and reduce the effect of risks.
4) Resource requirements: Specifies the hardware and software required to carry out the software project. Cost estimation is done according to these resource requirements.
5) Work breakdown: Describes the activities into which the project is divided. It also describes the milestones and deliverables of the project activities.
6) Project schedule: Specifies the dependencies of activities on each other. Based on this, the time required by the project management team members to complete the project activities is estimated.

PROJECT ESTIMATION TECHNIQUE

There are many different types of estimation techniques used in project management. A Project manager is often challenged to align mainly six project constraints –

a) *Scope,*
b) *Time,*
c) *Cost,*
d) *Quality,*
e) *Resources and Risk* in order to accurately estimate the project.

The common questions that come into the mind of a project manager at the start of the project is –

a) how much work is to be estimated (scope)?
b) how to estimate the project (techniques),
c) how much time it will require to complete the project (Schedule)
d) who will be doing the project (resources),
e) what is the budget required to deliver the project (cost), and

Any intermediary dependencies that may delay or impact the project (Risks).

Estimation of resources, cost, and schedule for a software engineering effort requires

a) experience
b) access to good historical information (metrics)
c) the courage to commit to quantitative predictions when qualitative information is all that exists

Estimation carries inherent risk and this risk leads to uncertainty

What to estimate?

a) Resources
b) Times
c) Human Skills
d) Cost
e) Environment
f) Reusable Software

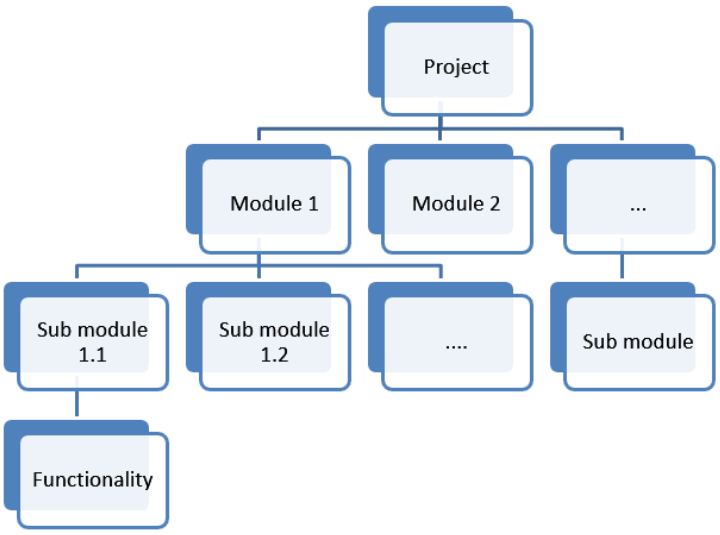
a) Resources: Resources are required to carry out any project tasks. They can be people, equipment, facilities, funding, or anything else capable of definition required for the completion of a project activity.
b) Times : Time is the most valuable resource in a project. Every project has a deadline to delivery.
Human
c) Skills : Human skills mean the knowledge and the experience of the Team members. They affect to your estimation. For example, a team, whose members have low testing skills, will take more time to finish the project than the one which has high testing skills.
d) Cost: Cost is the project budget. Generally speaking, it means how much money it takes to finish the project.

Estimation Techniques

a) Past (similar) project experience
b) Conventional estimation techniques
   A) task breakdown and effort estimates
   B) size (e.g., FP) estimates
c) Empirical models
d) Automated tools

How to estimate?

Work Breakdown Structure (*Decomposition Technique*)
3-Point Software Testing Estimation Technique (*Best case, most likely and worst case estimate*)
Function Point / Testing Point Analysis
Work Breakdown Structure (*Decomposition Technique*)
Following is the 4 Step process to arrive at an estimate
Step1) Divide the whole project task into subtasks
Task is a piece of work that has been given to someone. To do this, you can use the **Work Breakdown Structure** technique.
In this technique, a complex project is divided into modules. The modules are divided into sub-modules. Each sub-module is further divided into functionality. It means divide the whole project task into the **smallest** tasks.



The purpose of this activity is create task as **detailed** as **possible**.

Task	Sub task
Analyze software requirement specification	Investigate the soft requirement specs
	Interview with the developer & other stakeholders to know more about the website
Create the Test Specification	Design test scenarios
	Create test cases
	Review and revise test cases
Execute the test cases	Build up the test environment

	Execute the test cases
	Review test execution results
Report the defects	Create the Defect reports
	Report the defects

Step 2) Allocate each task to team member
In this step, each task is assigned to the **appropriate** member in the project team. You can assigned task as follows

Task	Members
Analyze software requirement specification	All the members
Create the test specification	Tester/Test Analyst
Build up the test environment	Test Administrator
Execute the test cases	Tester, Test Administrator
Report defects	Tester

Step 3) Effort Estimation For Tasks
Functional Point Method
In this method, the Test Manager estimates Size, Duration, and Cost for the tasks
Based on the complex of software functions,

Group	Weightage
Complex	5
Medium	3
Simple	1

STEP A) Estimate size for the task
STEP B) Estimate duration for the task
STEP C) Estimate the cost for the tasks
Three Point Estimation
When estimating a task, the Test Manager needs to provide three values, as specified above. The three values identified, estimate what happens in an **optimal state**, what is the **most likely**, or what we think it would be the **worst case** scenario.
You can estimate as following
The best case to complete this task is 120 man-hours (around 15 days). In this case, you have a talented team, they can finish the task in smallest time.
The most likely case to complete this task is 170 man-hours (around 21 days). This is a normal case, you have enough resource and ability to complete the task
The worst case to complete this task is 200 man-hours (around 25 days). You need to perform much more work because your team members are not experienced.

$$E = (a + 4m + b)/6$$

$$E = (120 + 4 * 170 + 200)/6$$

$$E = 166.6\ (man - hours)$$

Step 4) Validate the estimation
Once you create an aggregate estimate for all the tasks mentioned in the WBS, you need to forward it to the **management board**, who will **review** and **approve** it.

COCOMO Model

The Constructive Cost Model (COCOMO) is an algorithmic software cost estimation model developed by Barry W. Boehm. The model uses a basic regression formula with parameters that are derived from historical project data and current project characteristics

The Constructive cost model (COCOMO) was developed by Boehm. This model also estimates the total effort in terms of person-months of the technical project staff. The effort estimate includes development, management, and support tasks but does not include the cost of the secretarial and other staff that might be needed in an organization. The basic steps in this model are: -

The **COstructive COst MOdel** (COCOMO) is the most widely used software estimation model in the world. It The COCOMO model predicts the effort and duration of a project based on inputs relating to the size of the resulting systems and a number of "cost drives" that affect productivity.

Software Cost Estimation

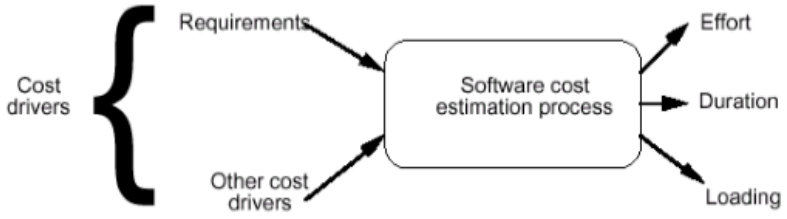


Figure 1. Classical view of software estimation process.

Effort:

- Effort Equation
  - **PM = C * (KDSI)^n (**person-months)
    - where **PM** = number of person-month (=152 working hours),
    - **C** = a constant,
    - **KDSI** = thousands of "delivered source instructions" (DSI) and
    - **n** = a constant.

Productivity:

- Productivity equation
  - **(DSI) / (PM)**
    - where **PM** = number of person-month (=152 working hours),
    - **DSI** = "delivered source instructions"

Schedule:

- Schedule equation
  - **TDEV = C * (PM)^n (**months)
  - Where TDEV = number of months estimated for software development.

Average Staffing:

- Average Staffing Equation
  - **(PM) / (TDEV)** **(**FSP)
  - Where FSP means Full-time-equivalent Software Personnel.

COCOMO is defined in terms of three different models:

1. Basic model

2. Intermediate model
3. Detailed model.

The more complex models account for more factors that influence software projects, and make more accurate estimates

Project Scheduling

Project Scheduling in a project refers to roadmap of all activities to be done with specified order and within time slot allotted to each activity. Project managers tend to define various tasks, and project milestones and arrange them keeping various factors in mind. They look for tasks lie in critical path in the schedule, which are necessary to complete in specific manner (because of task interdependency) and strictly within the time allocated. Arrangement of tasks which lies out of critical path are less likely to impact over all schedule of the project.

For scheduling a project, it is necessary to -

i) Break down the project tasks into smaller, manageable form
ii) Find out various tasks and correlate them
iii) Estimate time frame required for each task
iv) Divide time into work-units
v) Assign adequate number of work-units for each task
vi) Calculate total time required for the project from start to finish

Project Risk Management

Risk management involves all activities pertaining to identification, analyzing and making provision for predictable and non-predictable risks in the project. Risk may include the following:

- Experienced staff leaving the project and new staff coming in.
- Change in organizational management.
- Requirement change or misinterpreting requirement.
- Under-estimation of required time and resources.
- Technological changes, environmental changes, business competition.

Risk Management Process

There are following activities involved in risk management process:

- **Identification -** Make note of all possible risks, which may occur in the project.
- **Categorize -** Categorize known risks into high, medium and low risk intensity as per their possible impact on the project.
- **Manage -** Analyze the probability of occurrence of risks at various phases. Make plan to avoid or face risks. Attempt to minimize their side-effects.
- **Monitor -** Closely monitor the potential risks and their early symptoms. Also monitor the effects of steps taken to mitigate or avoid them.

Staffing is the most important part of project management. It is the staff who will actually complete the project work. Staff will also consume the majority of project cost. Hence it is extremely important to be very precise in planning and acquiring the right staff at the right time for the right duration. It is also important to keep the staff members motivated and ensure their safety and well- being. The staffing management plan help capture all these aspects precisely for effecting staff management for the project.

Staffing Management Plan

Staffing management plan and Resource management plans are important part of project resource management. Every project will require resources for executing project activities. There will be a need for both man power resources and physical resources. The resource requirement for each activity will be estimated. The resources will be acquired during project execution as per the schedule.

Planning for resources, acquiring resources, developing team and managing team are the important activities to be carried out as part of project resource management. A resource management plan will contain all the necessary guidelines for project resource management. A staffing management plan will also be part of the overall resource management plan. Staffing management plan, which part of overall resource management plan will specifically focus on the man power aspects of the project. Staffs are the most important part of project. It is important to select and acquire the right staff with right skills at the right time. A staffing management plan contains a plan for addressing all the aspects of man power and will include below information:

- Identification of human resources
- How the human resources will be acquired

- Criteria to be used for how the human resources will be selected
- From where the human resources will be acquired
- How to acquire resources from within the organization
- How to acquire resources from external sources
- When the resources will be acquired (based on the project schedule)
- When the resources will be released (based on the project schedule)
- Process for maintaining the resource calendars
- Resource loading table depicting total number of resources needed at different points in the project
- Safety and security guidelines for the human resources
- Identification of training needs and plan for fulfilling the training needs of the team
- Rewards and recognition plan for the team
- How to build the team and enhance team performance
- How to monitor the performance of each team member and help keeping them motivated

1.Functional and non-functional requirement

According to IEEE standard 729, a requirement is defined as follows:
A condition or capability needed by a user to solve a problem or achieve an objective
A condition or capability that must be met or possessed by a system or system component to satisfy a contract, standard, specification or other formally imposed documents
A documented representation of a condition or capability as in 1 and 2.

A software requirement can be of 3 types:

1) Functional requirements
2) Non-functional requirements
3) Domain requirements

1) Functional Requirements:
These are the requirements that the end user specifically demands as basic facilities that the system should offer. All these functionalities need to be necessarily incorporated into the system as a part of the contract. These are represented or stated in the form of input to be given to the system, the operation performed and the output expected. They are basically the requirements stated by the user which one can see directly in the final product, unlike the non-functional requirements.
For example, in a hospital management system, a doctor should be able to retrieve the information of his patients. Each high-level functional requirement may involve several interactions or dialogues between the system and the outside world. In order to accurately describe the functional requirements, all scenarios must be enumerated.

There are many ways of expressing functional requirements e.g., natural language, a structured or formatted language with no rigorous syntax and formal specification language with proper syntax.

2) Non-functional requirements: These are basically the quality constraints that the system must satisfy according to the project contract. The priority or extent to which these factors are implemented varies from one project to other. They are also called non-behavioral requirements.
They basically deal with issues like:

- Portability
- Security
- Maintainability
- Reliability
- Scalability
- Performance
- Reusability
- Flexibility

NFR's are classified into following types:

- Interface constraints
- Performance constraints: response time, security, storage space, etc.
- Operating constraints
- Life cycle constraints: maintainability, portability, etc.
- Economic constraints

The process of specifying non-functional requirements requires the knowledge of the functionality of the system, as well as the knowledge of the context within which the system will operate.

3) Domain requirements: Domain requirements are the requirements which are characteristic of a particular category or domain of projects. The basic functions that a system of a specific domain must necessarily exhibit come under this category. For instance, in an academic software that maintains records of a school or college, the functionality of being able to access the list of faculty and list of students of each grade is a domain requirement. These requirements are therefore identified from that domain model and are not user specific

2. Requirements engineering process (feasibility studies, requirement elicitation and analysis, requirement validation, Requirements management)

Requirement Engineering is the process of defining, documenting and maintaining the requirements. It is a process of gathering and defining service provided by the system. Requirements Engineering Process consists of the following main activities:

- Requirements elicitation
- Requirements specification
- Requirements verification and validation
- Requirements management

Requirements Elicitation:
It is related to the various ways used to gain knowledge about the project domain and requirements. The various sources of domain knowledge include customers, business manuals, the existing software of same type, standards and other stakeholders of the project.
The techniques used for requirements elicitation include interviews, brainstorming, task analysis, Delphi technique, prototyping, etc. Elicitation does not produce formal models of the requirements understood. Instead, it widens the knowledge domain of the analyst and thus helps in providing input to the next stage.

Requirements specification:
This activity is used to produce formal software requirement models. All the requirements including the functional as well as the non-functional requirements and the constraints are specified by these models in totality. During specification, more knowledge about the problem may be required which can again trigger the elicitation process.
The models used at this stage include ER diagrams, data flow diagrams (DFDs), function decomposition diagrams (FDDs), data dictionaries, etc.

Requirements verification and validation:
Verification: It refers to the set of tasks that ensure that software correctly implements a specific function.
Validation: It refers to a different set of tasks that ensure that the software that has been built is traceable to customer requirements.
If requirements are not validated, errors in the requirements definitions would propagate to the successive stages resulting in a lot of modification and rework.
The main steps for this process include:

- The requirements should be consistent with all the other requirements i.e no two requirements should conflict with each other.
- The requirements should be complete in every sense.
- The requirements should be practically achievable.

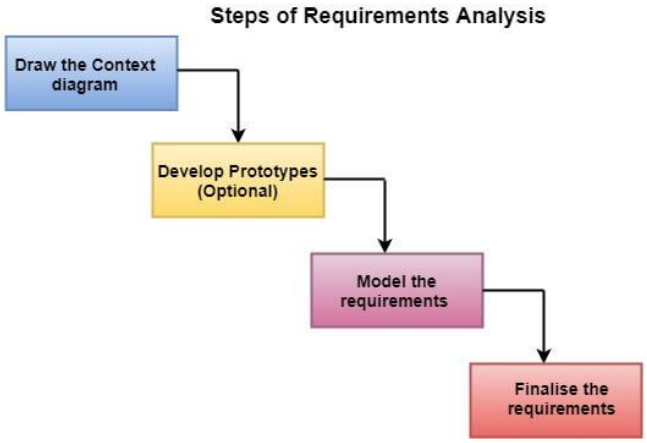
Reviews, buddy checks, making test cases, etc. are some of the methods used for this.

Requirements management:
Requirement management is the process of analyzing, documenting, tracking, prioritizing and agreeing on the requirement and controlling the communication to relevant stakeholders. This stage takes care of the changing nature of requirements. It should be ensured that the SRS is as modifiable as possible so as to incorporate changes in requirements specified by the end users at later stages too. Being able to modify the software as per requirements in a systematic and controlled manner in an extremely important part of the requirements engineering process.

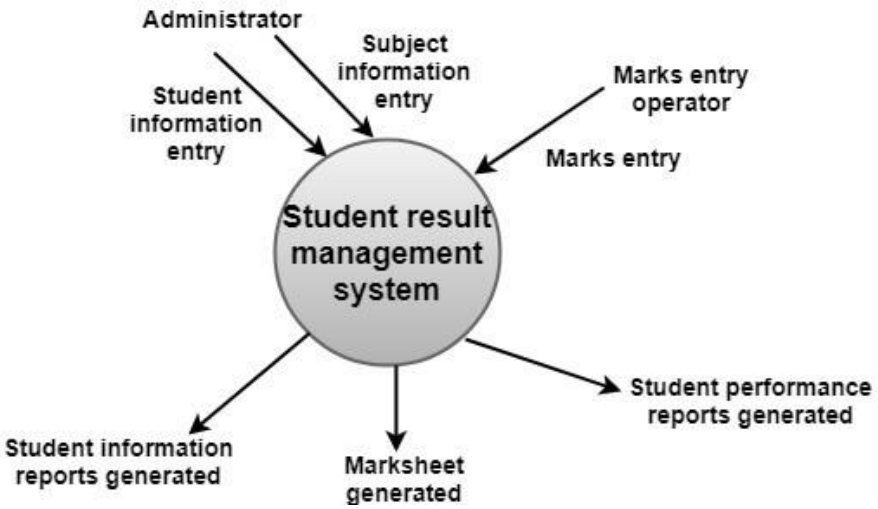
Requirements Analysis

Requirement analysis is significant and essential activity after elicitation. We analyze, refine, and scrutinize the gathered requirements to make consistent and unambiguous requirements. This activity reviews all requirements and may provide a graphical view of the entire system. After the completion of the analysis, it is expected that the understandability of the project may improve significantly. Here, we may also use the interaction with the customer to clarify points of confusion and to understand which requirements are more important than others.

The various steps of requirement analysis are shown in fig:



(i) Draw the context diagram: The context diagram is a simple model that defines the boundaries and interfaces of the proposed systems with the external world. It identifies the entities outside the proposed system that interact with the system. The context diagram of student result management system is given below:



(ii) Development of a Prototype (optional): One effective way to find out what the customer wants is to construct a prototype, something that looks and preferably acts as part of the system they say they want.

We can use their feedback to modify the prototype until the customer is satisfied continuously. Hence, the prototype helps the client to visualize the proposed system and increase the understanding of the requirements. When developers and users are not sure about some of the elements, a prototype may help both the parties to take a final decision.

Some projects are developed for the general market. In such cases, the prototype should be shown to some representative sample of the population of potential purchasers. Even though a

person who tries out a prototype may not buy the final system, but their feedback may allow us to make the product more attractive to others.

The prototype should be built quickly and at a relatively low cost. Hence it will always have limitations and would not be acceptable in the final system. This is an optional activity.

(iii) Model the requirements: This process usually consists of various graphical representations of the functions, data entities, external entities, and the relationships between them. The graphical view may help to find incorrect, inconsistent, missing, and superfluous requirements. Such models include the Data Flow diagram, Entity-Relationship diagram, Data Dictionaries, State-transition diagrams, etc.

(iv) Finalize the requirements: After modeling the requirements, we will have a better understanding of the system behavior. The inconsistencies and ambiguities have been identified and corrected. The flow of data amongst various modules has been analyzed. Elicitation and analyze activities have provided better insight into the system. Now we finalize the analyzed requirements, and the next step is to document these requirements in a prescribed format.

he software requirements are description of features and functionalities of the target system. Requirements convey the expectations of users from the software product. The requirements can be obvious or hidden, known or unknown, expected or unexpected from client's point of view.

Requirement Engineering

The process to gather the software requirements from client, analyze and document them is known as requirement engineering.

The goal of requirement engineering is to develop and maintain sophisticated and descriptive 'System Requirements Specification' document.

Requirement Engineering Process

It is a four-step process, which includes –

- Feasibility Study
- Requirement Gathering
- Software Requirement Specification
- Software Requirement Validation

Let us see the process briefly -

Feasibility study

When the client approaches the organization for getting the desired product developed, it comes up with rough idea about what all functions the software must perform and which all features are expected from the software.

Referencing to this information, the analysts does a detailed study about whether the desired system and its functionality are feasible to develop.

This feasibility study is focused towards goal of the organization. This study analyzes whether the software product can be practically materialized in terms of implementation, contribution of project to organization, cost constraints and as per values and objectives of the organization. It explores technical aspects of the project and product such as usability, maintainability, productivity and integration ability.

The output of this phase should be a feasibility study report that should contain adequate comments and recommendations for management about whether or not the project should be undertaken.

Requirement Gathering

If the feasibility report is positive towards undertaking the project, next phase starts with gathering requirements from the user. Analysts and engineers communicate with the client and end-users to know their ideas on what the software should provide and which features they want the software to include.

Software Requirement Specification

SRS is a document created by system analyst after the requirements are collected from various stakeholders.

SRS defines how the intended software will interact with hardware, external interfaces, speed of operation, response time of system, portability of software across various platforms, maintainability, speed of recovery after crashing, Security, Quality, Limitations etc.

The requirements received from client are written in natural language. It is the responsibility of system analyst to document the requirements in technical language so that they can be comprehended and useful by the software development team.

SRS should come up with following features:

- User Requirements are expressed in natural language.
- Technical requirements are expressed in structured language, which is used inside the organization.
- Design description should be written in Pseudo code.
- Format of Forms and GUI screen prints.
- Conditional and mathematical notations for DFDs etc.

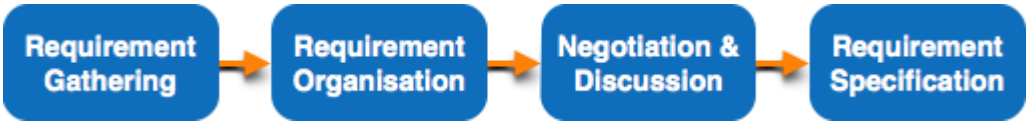
Software Requirement Validation

After requirement specifications are developed, the requirements mentioned in this document are validated. User might ask for illegal, impractical solution or experts may interpret the requirements incorrectly. This results in huge increase in cost if not nipped in the bud. Requirements can be checked against following conditions -

- If they can be practically implemented
- If they are valid and as per functionality and domain of software
- If there are any ambiguities
- If they are complete
- If they can be demonstrated

Requirement Elicitation Process

Requirement elicitation process can be depicted using the folloiwng diagram:



- **Requirements gathering -** The developers discuss with the client and end users and know their expectations from the software.
- **Organizing Requirements -** The developers prioritize and arrange the requirements in order of importance, urgency and convenience.
- **Negotiation & discussion -** If requirements are ambiguous or there are some conflicts in requirements of various stakeholders, if they are, it is then negotiated and discussed with stakeholders. Requirements may then be prioritized and reasonably compromised.

  The requirements come from various stakeholders. To remove the ambiguity and conflicts, they are discussed for clarity and correctness. Unrealistic requirements are compromised reasonably.

- **Documentation -** All formal & informal, functional and non-functional requirements are documented and made available for next phase processing.

Requirement Elicitation Techniques

Requirements Elicitation is the process to find out the requirements for an intended software system by communicating with client, end users, system users and others who have a stake in the software system development.

There are various ways to discover requirements

Interviews

Interviews are strong medium to collect requirements. Organization may conduct several types of interviews such as:

- Structured (closed) interviews, where every single information to gather is decided in advance, they follow pattern and matter of discussion firmly.
- Non-structured (open) interviews, where information to gather is not decided in advance, more flexible and less biased.
- Oral interviews
- Written interviews
- One-to-one interviews which are held between two persons across the table.
- Group interviews which are held between groups of participants. They help to uncover any missing requirement as numerous people are involved.

Surveys

Organization may conduct surveys among various stakeholders by querying about their expectation and requirements from the upcoming system.

Questionnaires

A document with pre-defined set of objective questions and respective options is handed over to all stakeholders to answer, which are collected and compiled.

A shortcoming of this technique is, if an option for some issue is not mentioned in the questionnaire, the issue might be left unattended.

Task analysis

Team of engineers and developers may analyze the operation for which the new system is required. If the client already has some software to perform certain operation, it is studied and requirements of proposed system are collected.

Domain Analysis

Every software falls into some domain category. The expert people in the domain can be a great help to analyze general and specific requirements.

Brainstorming

An informal debate is held among various stakeholders and all their inputs are recorded for further requirements analysis.

Prototyping

Prototyping is building user interface without adding detail functionality for user to interpret the features of intended software product. It helps giving better idea of requirements. If there is no software installed at client's end for developer's reference and the client is not aware of its own requirements, the developer creates a prototype based on initially mentioned requirements. The prototype is shown to the client and the feedback is noted. The client feedback serves as an input for requirement gathering.

Observation

Team of experts visit the client's organization or workplace. They observe the actual working of the existing installed systems. They observe the workflow at client's end and how execution problems are dealt. The team itself draws some conclusions which aid to form requirements expected from the software.

Software Requirements Characteristics

Gathering software requirements is the foundation of the entire software development project. Hence they must be clear, correct and well-defined.

A complete Software Requirement Specifications must be:

- Clear
- Correct
- Consistent
- Coherent
- Comprehensible
- Modifiable
- Verifiable
- Prioritized
- Unambiguous
- Traceable
- Credible source

Software Requirements

We should try to understand what sort of requirements may arise in the requirement elicitation phase and what kinds of requirements are expected from the software system.

Broadly software requirements should be categorized in two categories:

Functional Requirements

Requirements, which are related to functional aspect of software fall into this category.

They define functions and functionality within and from the software system.

Examples -

- Search option given to user to search from various invoices.
- User should be able to mail any report to management.
- Users can be divided into groups and groups can be given separate rights.
- Should comply business rules and administrative functions.
- Software is developed keeping downward compatibility intact.

Non-Functional Requirements

Requirements, which are not related to functional aspect of software, fall into this category. They are implicit or expected characteristics of software, which users make assumption of.

Non-functional requirements include -

- Security
- Logging
- Storage
- Configuration
- Performance

- Cost
- Interoperability
- Flexibility
- Disaster recovery
- Accessibility

Requirements are categorized logically as

- **Must Have** : Software cannot be said operational without them.
- **Should have** : Enhancing the functionality of software.
- **Could have** : Software can still properly function with these requirements.
- **Wish list** : These requirements do not map to any objectives of software.

While developing software, 'Must have' must be implemented, 'Should have' is a matter of debate with stakeholders and negation, whereas 'could have' and 'wish list' can be kept for software updates.

User Interface requirements

UI is an important part of any software or hardware or hybrid system. A software is widely accepted if it is -

- easy to operate
- quick in response
- effectively handling operational errors
- providing simple yet consistent user interface

User acceptance majorly depends upon how user can use the software. UI is the only way for users to perceive the system. A well performing software system must also be equipped with attractive, clear, consistent and responsive user interface. Otherwise the functionalities of software system can not be used in convenient way. A system is said be good if it provides means to use it efficiently. User interface requirements are briefly mentioned below -

- Content presentation
- Easy Navigation
- Simple interface
- Responsive
- Consistent UI elements
- Feedback mechanism
- Default settings
- Purposeful layout
- Strategical use of color and texture.
- Provide help information
- User centric approach
- Group based view settings.

Software System Analyst

System analyst in an IT organization is a person, who analyzes the requirement of proposed system and ensures that requirements are conceived and documented properly & correctly. Role of an analyst starts during Software Analysis Phase of SDLC. It is the responsibility of analyst to make sure that the developed software meets the requirements of the client.

System Analysts have the following responsibilities:

- Analyzing and understanding requirements of intended software

- Understanding how the project will contribute in the organization objectives
- Identify sources of requirement
- Validation of requirement
- Develop and implement requirement management plan
- Documentation of business, technical, process and product requirements
- Coordination with clients to prioritize requirements and remove and ambiguity
- Finalizing acceptance criteria with client and other stakeholders

Software Metrics and Measures

Software Measures can be understood as a process of quantifying and symbolizing various attributes and aspects of software.

Software Metrics provide measures for various aspects of software process and software product.

Software measures are fundamental requirement of software engineering. They not only help to control the software development process but also aid to keep quality of ultimate product excellent.

According to Tom DeMarco, a (Software Engineer), "You cannot control what you cannot measure." By his saying, it is very clear how important software measures are.

Let us see some software metrics:

- **Size Metrics -** LOC (Lines of Code), mostly calculated in thousands of delivered source code lines, denoted as KLOC.

  Function Point Count is measure of the functionality provided by the software. Function Point count defines the size of functional aspect of software.

- **Complexity Metrics -** McCabe's Cyclomatic complexity quantifies the upper bound of the number of independent paths in a program, which is perceived as complexity of the program or its modules. It is represented in terms of graph theory concepts by using control flow graph.
- **Quality Metrics -** Defects, their types and causes, consequence, intensity of severity and their implications define the quality of product.

  The number of defects found in development process and number of defects reported by the client after the product is installed or delivered at client-end, define quality of product.

- **Process Metrics -** In various phases of SDLC, the methods and tools used, the company standards and the performance of development are software process metrics.
- **Resource Metrics -** Effort, time and various resources used, represents metrics for resource measurement.

3. Data modeling and flow diagram

4. Software prototyping techniques

Software Prototyping Techniques

Software prototyping is the activity of creating prototypes of software applications, i.e., incomplete versions of the software program being developed. It is an activity that can occur in software development and is comparable to prototyping as known from other fields, such as mechanical engineering or manufacturing.

A prototype typically simulates only a few aspects of, and may be completely different from, the final product.

Outline of the prototyping process

The process of prototyping involves the following steps

1. Identify basic requirements Determine basic requirements including the input and output information desired. Details, such as security, can typically be ignored.

2. Develop Initial Prototype The initial prototype is developed that includes only user interfaces. (See Horizontal Prototype, below)
3. Review The customers, including end-users, examine the prototype and provide feedback on additions or changes.
4. Revise and Enhance the Prototype

Using the feedback both the specifications and the prototype can be improved. Negotiation about what is within the scope of the contract/product may be necessary. If changes are introduced then a repeat of steps #3 and #4 may be needed.

Advantages of prototyping

1. **Reduced time and costs:** Prototyping can improve the quality of requirements and specifications provided to developers. Because changes cost exponentially more to implement as they are detected later in development, the early determination of what the user really wants can result in faster and less expensive software.

2. Improved and increased user involvement: Prototyping requires user involvement and allows them to see and interact with a prototype allowing them to provide better and more complete feedback and specifications. The presence of the prototype being examined by the user prevents many misunderstandings and miscommunications that occur when each side believe the other understands what they said. Since users know the problem domain better than anyone on the development team does, increased interaction can result in final product that has greater tangible and intangible quality. The final product is more likely to satisfy the user's desire for look, feel and performance.
Dimensions of prototypes

3. Horizontal Prototype

A common term for a user interface prototype is the horizontal prototype. It provides a broad view of an entire system or subsystem, focusing on user interaction more than low-level system functionality, such as database access. Horizontal prototypes are useful for:

- Confirmation of user interface requirements and system scope
- Demonstration version of the system to obtain buy-in from the business
- Develop preliminary estimates of development time, cost and effort.
-

4. Vertical Prototype

A vertical prototype is a more complete elaboration of a single subsystem or function. It is useful for obtaining detailed requirements for a given function, with the following benefits: Refinement database design Obtain information on data volumes and system interface needs, for network sizing and performance engineering Clarifies complex requirements by drilling down to actual system functionality

Requirement definition and specification

The introduction to the Software Requirement Specification (SRS) document should provide an overview of the complete SRS documents. While writing this document please remember that this document should contain all of the information needed by a software engineer to adequately design and implement the software product described by the requirements listed in this document.

5. Requirement definition and specification

Software Design

Introduction to software design
good software design
Design principle
Design concepts
Designs strategy
design process
design quality
software architecture and its types

1.Introduction to software design

Software design is a process to transform user requirements into some suitable form, which helps the programmer in software coding and implementation.

For assessing user requirements, an SRS (Software Requirement Specification) document is created whereas for coding and implementation, there is a need of more specific and detailed requirements in software terms. The output of this process can directly be used into implementation in programming languages.

Software design is the first step in SDLC (Software Design Life Cycle), which moves the concentration from problem domain to solution domain. It tries to specify how to fulfill the requirements mentioned in SRS.

Software Design Levels

Software design yields three levels of results:

- **Architectural Design -** The architectural design is the highest abstract version of the system. It identifies the software as a system with many components interacting with each other. At this level, the designers get the idea of proposed solution domain.
- **High-level Design-** The high-level design breaks the 'single entity-multiple component' concept of architectural design into less-abstracted view of sub-systems and modules and depicts their interaction with each other. High-level design focuses on how the system along with all of its components can be implemented in forms of modules. It recognizes modular structure of each sub-system and their relation and interaction among each other.
- **Detailed Design-** Detailed design deals with the implementation part of what is seen as a system and its sub-systems in the previous two designs. It is more detailed towards modules and their implementations. It defines logical structure of each module and their interfaces to communicate with other modules.

Modularization

Modularization is a technique to divide a software system into multiple discrete and independent modules, which are expected to be capable of carrying out task(s) independently. These modules may work as basic constructs for the entire software. Designers tend to design modules such that they can be executed and/or compiled separately and independently.

Modular design unintentionally follows the rules of 'divide and conquer' problem-solving strategy this is because there are many other benefits attached with the modular design of a software.

Advantage of modularization:

- Smaller components are easier to maintain
- Program can be divided based on functional aspects
- Desired level of abstraction can be brought in the program
- Components with high cohesion can be re-used again

- Concurrent execution can be made possible
- Desired from security aspect

Concurrency

Back in time, all software are meant to be executed sequentially. By sequential execution we mean that the coded instruction will be executed one after another implying only one portion of program being activated at any given time. Say, a software has multiple modules, then only one of all the modules can be found active at any time of execution.

In software design, concurrency is implemented by splitting the software into multiple independent units of execution, like modules and executing them in parallel. In other words, concurrency provides capability to the software to execute more than one part of code in parallel to each other.

It is necessary for the programmers and designers to recognize those modules, which can be made parallel execution.

Example

The spell check feature in word processor is a module of software, which runs along side the word processor itself.

Coupling and Cohesion

When a software program is modularized, its tasks are divided into several modules based on some characteristics. As we know, modules are set of instructions put together in order to achieve some tasks. They are though, considered as single entity but may refer to each other to work together. There are measures by which the quality of a design of modules and their interaction among them can be measured. These measures are called coupling and cohesion.

Cohesion

Cohesion is a measure that defines the degree of intra-dependability within elements of a module. The greater the cohesion, the better is the program design.

There are seven types of cohesion, namely –

- **Co-incidental cohesion -** It is unplanned and random cohesion, which might be the result of breaking the program into smaller modules for the sake of modularization. Because it is unplanned, it may serve confusion to the programmers and is generally not-accepted.
- **Logical cohesion -** When logically categorized elements are put together into a module, it is called logical cohesion.
- **Temporal Cohesion -** When elements of module are organized such that they are processed at a similar point in time, it is called temporal cohesion.
- **Procedural cohesion -** When elements of module are grouped together, which are executed sequentially in order to perform a task, it is called procedural cohesion.
- **Communicational cohesion -** When elements of module are grouped together, which are executed sequentially and work on same data (information), it is called communicational cohesion.
- **Sequential cohesion -** When elements of module are grouped because the output of one element serves as input to another and so on, it is called sequential cohesion.
- **Functional cohesion -** It is considered to be the highest degree of cohesion, and it is highly expected. Elements of module in functional cohesion are grouped because they all contribute to a single well-defined function. It can also be reused.

Coupling

Coupling is a measure that defines the level of inter-dependability among modules of a program. It tells at what level the modules interfere and interact with each other. The lower the coupling, the better the program.

There are five levels of coupling, namely -

- **Content coupling -** When a module can directly access or modify or refer to the content of another module, it is called content level coupling.
- **Common coupling-** When multiple modules have read and write access to some global data, it is called common or global coupling.
- **Control coupling-** Two modules are called control-coupled if one of them decides the function of the other module or changes its flow of execution.
- **Stamp coupling-** When multiple modules share common data structure and work on different part of it, it is called stamp coupling.
- **Data coupling-** Data coupling is when two modules interact with each other by means of passing data (as parameter). If a module passes data structure as parameter, then the receiving module should use all its components.

Ideally, no coupling is considered to be the best.

Design Verification

The output of software design process is design documentation, pseudo codes, detailed logic diagrams, process diagrams, and detailed description of all functional or non-functional requirements.

The next phase, which is the implementation of software, depends on all outputs mentioned above.

It is then becomes necessary to verify the output before proceeding to the next phase. The early any mistake is detected, the better it is or it might not be detected until testing of the product. If the outputs of design phase are in formal notation form, then their associated tools for verification should be used otherwise a thorough design review can be used for verification and validation.

By structured verification approach, reviewers can detect defects that might be caused by overlooking some conditions. A good design review is important for good software design, accuracy and quality.

2.Good software design

Characteristics of a good Software Design

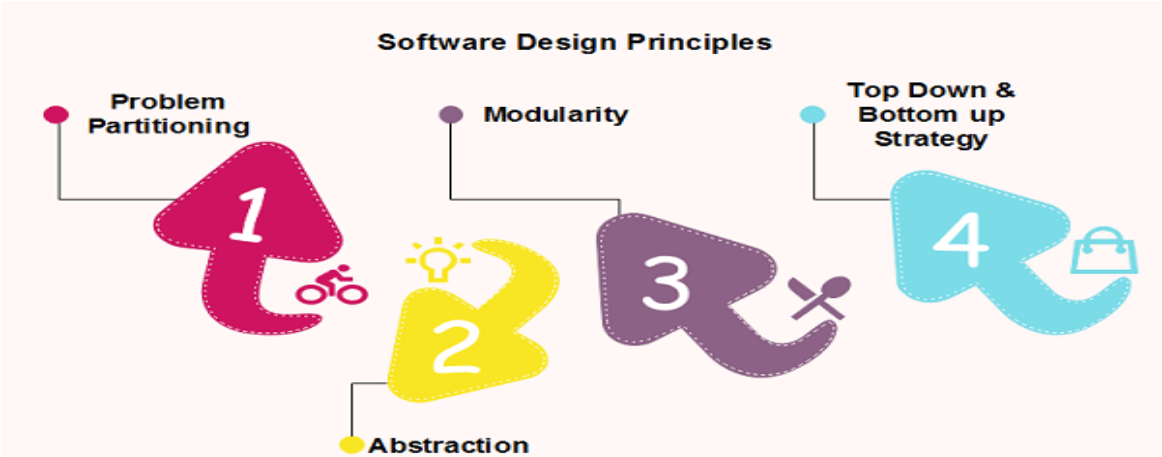
The design needs to be

1. Correct & complete
2. Understandable
3. At the right level
4. Maintainable

Software Design Principles

Software design principles are concerned with providing means to handle the complexity of the design process effectively. Effectively managing the complexity will not only reduce the effort needed for design but can also reduce the scope of introducing errors during design.

Following are the principles of Software Design



A Problem Partitioning

For small problem, we can handle the entire problem at once but for the significant problem, divide the problems and conquer the problem it means to divide the problem into smaller pieces so that each piece can be captured separately.

For software design, the goal is to divide the problem into manageable pieces.

Benefits of Problem Partitioning

1. Software is easy to understand
2. Software becomes simple
3. Software is easy to test
4. Software is easy to modify
5. Software is easy to maintain
6. Software is easy to expand

These pieces cannot be entirely independent of each other as they together form the system. They have to cooperate and communicate to solve the problem. This communication adds complexity.

Note: ***As the number of partition increases = Cost of partition and complexity increases***

B Abstraction

An abstraction is a tool that enables a designer to consider a component at an abstract level without bothering about the internal details of the implementation. Abstraction can be used for existing element as well as the component being designed.

Here, there are two common abstraction mechanisms

1. Functional Abstraction
2. Data Abstraction

Functional Abstraction

i. A module is specified by the method it performs.

ii. The details of the algorithm to accomplish the functions are not visible to the user of the function.

Functional abstraction forms the basis for **Function oriented design approaches**.

Data Abstraction

Details of the data elements are not visible to the users of data. Data Abstraction forms the basis for **Object Oriented design approaches**.

C Modularity

Modularity specifies to the division of software into separate modules which are differently named and addressed and are integrated later on in to obtain the completely functional software. It is the only property that allows a program to be intellectually manageable. Single large programs are difficult to understand and read due to a large number of reference variables, control paths, global variables, etc.

The desirable properties of a modular system are:

- Each module is a well-defined system that can be used with other applications.
- Each module has single specified objectives.
- Modules can be separately compiled and saved in the library.
- Modules should be easier to use than to build.
- Modules are simpler from outside than inside.

Advantages and Disadvantages of Modularity

In this topic, we will discuss various advantage and disadvantage of Modularity.

Advantages of Modularity

There are several advantages of Modularity

- It allows large programs to be written by several or different people
- It encourages the creation of commonly used routines to be placed in the library and used by other programs.
- It simplifies the overlay procedure of loading a large program into main storage.
- It provides more checkpoints to measure progress.
- It provides a framework for complete testing, more accessible to test
- It produced the well designed and more readable program.

Disadvantages of Modularity

There are several disadvantages of Modularity

- Execution time maybe, but not certainly, longer
- Storage size perhaps, but is not certainly, increased

- Compilation and loading time may be longer
- Inter-module communication problems may be increased
- More linkage required, run-time may be longer, more source lines must be written, and more documentation has to be done

Modular Design

Modular design reduces the design complexity and results in easier and faster implementation by allowing parallel development of various parts of a system. We discuss a different section of modular design in detail in this section:

1. Functional Independence: Functional independence is achieved by developing functions that perform only one kind of task and do not excessively interact with other modules. Independence is important because it makes implementation more accessible and faster. The independent modules are easier to maintain, test, and reduce error propagation and can be reused in other programs as well. Thus, functional independence is a good design feature which ensures software quality.

It is measured using two criteria:

- **Cohesion:** It measures the relative function strength of a module.
- **Coupling:** It measures the relative interdependence among modules.

2. Information hiding: The fundamental of Information hiding suggests that modules can be characterized by the design decisions that protect from the others, i.e., In other words, modules should be specified that data include within a module is inaccessible to other modules that do not need for such information.

The use of information hiding as design criteria for modular system provides the most significant benefits when modifications are required during testing's and later during software maintenance. This is because as most data and procedures are hidden from other parts of the software, inadvertent errors introduced during modifications are less likely to propagate to different locations within the software.

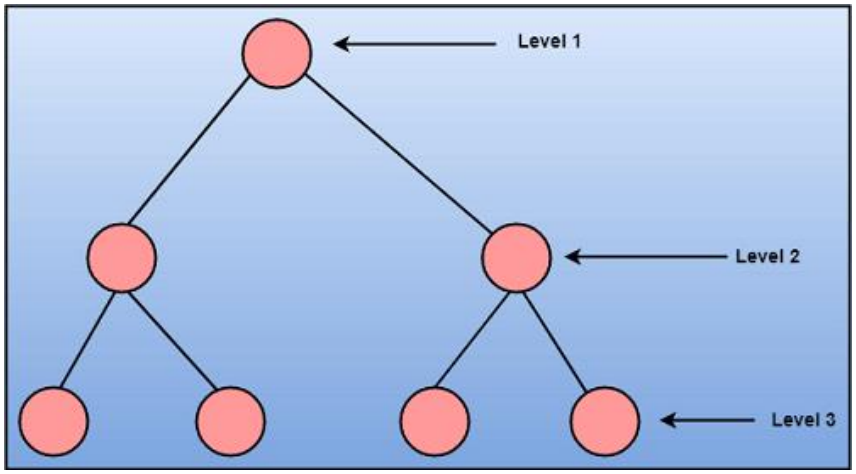
D Strategy of Design

A good system design strategy is to organize the program modules in such a method that are easy to develop and latter too, change. Structured design methods help developers to deal with the size and complexity of programs. Analysts generate instructions for the developers about how code should be composed and how pieces of code should fit together to form a program.

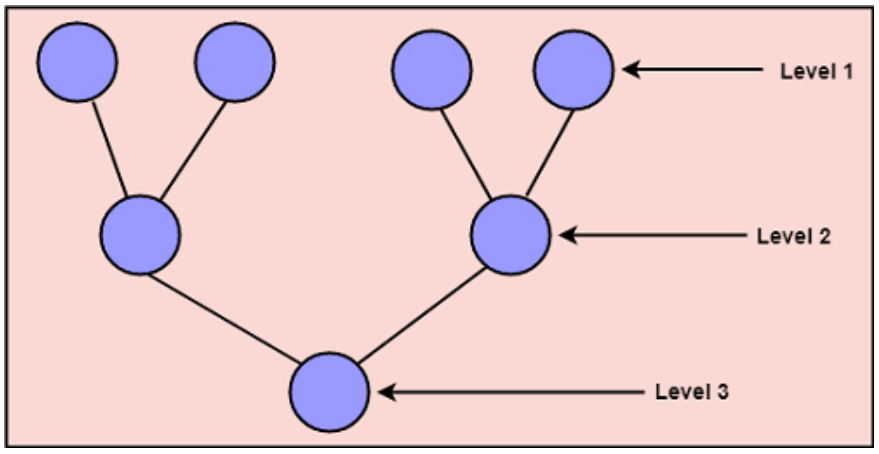
To design a system, there are two possible approaches:

1. Top-down Approach
2. Bottom-up Approach

1. Top-down Approach: This approach starts with the identification of the main components and then decomposing them into their more detailed sub-components.



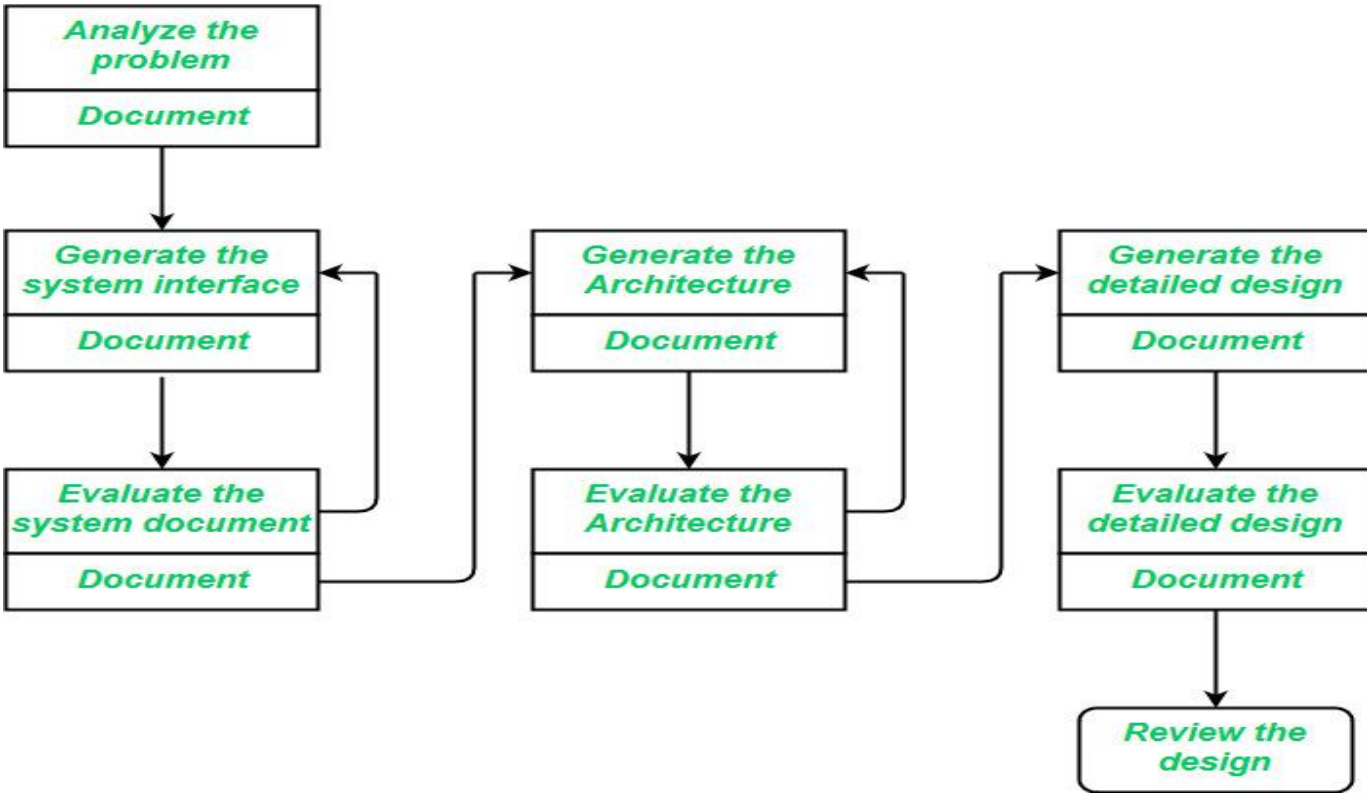
2. Bottom-up Approach: A bottom-up approach begins with the lower details and moves towards up the hierarchy, as shown in fig. This approach is suitable in case of an existing system.



Software Engineering | Software Design Process

The design phase of software development deals with transforming the customer requirements as described in the SRS documents into a form implementable using a programming language.
The software design process can be divided into the following three levels of phases of design:

1. Interface Design
2. Architectural Design
3. Detailed Design



Interface Design:
Interface design is the specification of the interaction between a system and its environment. this phase proceeds at a high level of abstraction with respect to the inner workings of the system i.e, during interface design, the internal of the systems are completely ignored and the system is treated as a black box. Attention is focussed on the dialogue between the target system and the users, devices, and other systems with which it interacts. The design problem statement produced during the problem analysis step should identify the people, other systems, and devices which are collectively called *agents*.

Interface design should include the following details:

- Precise description of events in the environment, or messages from agents to which the system must respond.
- Precise description of the events or messages that the system must produce.
- Specification on the data, and the formats of the data coming into and going out of the system.
- Specification of the ordering and timing relationships between incoming events or messages, and outgoing events or outputs.

Architectural Design:
Architectural design is the specification of the major components of a system, their responsibilities, properties, interfaces, and the relationships and interactions between them. In architectural design, the overall structure of the system is chosen, but the internal details of major components are ignored.

Issues in architectural design includes:

- Gross decomposition of the systems into major components.
- Allocation of functional responsibilities to components.
- Component Interfaces
- Component scaling and performance properties, resource consumption properties, reliability properties, and so forth.
- Communication and interaction between components.

The architectural design adds important details ignored during the interface design. Design of the internals of the major components is ignored until the last phase of the design.

Detailed Design:
Design is the specification of the internal elements of all major system components, their properties, relationships, processing, and often their algorithms and the data structures.
The detailed design may include:

- Decomposition of major system components into program units.
- Allocation of functional responsibilities to units.
- User interfaces
- Unit states and state changes
- Data and control interaction between units
- Data packaging and implementation, including issues of scope and visibility of program elements
- Algorithms and data structures

Introduction

1.Software Testing process,
2. principle of testing,

Testing is the process of exercising the program with the specific intent of finding the errors prior to delivery to the end user.

Testing Objectives

a) Testing is a process of executing a program with the intent of finding an error.
b) A good test case is one that has a high probability of finding an as-yet undiscovered error.
c) A successful test is one that uncovers an as-yet-undiscovered error.

Testing Principles

i) All tests should be traceable to customer requirements
ii) Tests should be planned long before testing begins. Test planning can begin as soon as the requirements model is complete. All tests can be planned and designed before any code has been generated.
iii) The Pareto principle applies to software testing. It states that 80 percent of all errors uncovered during testing will likely be traceable to 20 percent of all program components.
iv) Testing should begin "in the small" and progress toward testing "in the large." The first tests planned and executed generally focuses on individual (small) components. As testing progresses, focus shifts in an attempt to find errors in the integrated components and ultimately in the entire.
v) Exhaustive testing is not possible. It is impossible to execute every combination of paths during testing. It is possible, however, to adequately cover program logic and to ensure that all conditions in the design have been exercised.
vi) To be most effective, testing should be conducted by an independent third party.

Testability

Software testability is simply how easily a computer program can be tested.

Characteristics (OOSSDUC)

" Operability "The better it works, the more efficiently it can be tested."
"Observability "What you see is what you test.
" Simplicity "The less there is to test, the more quickly we can test it.
" Stability "The fewer the changes, the fewer the disruptions to testing.
" Decomposability "By controlling the scope of testing, we can more quickly isolate the errors and conduct smart testing.
"Understandability "The more information we have, the smarter we will test.
" Controllability "The better we can control the software, the more the testing can be optimized"

Good Test Case

1. A good test has a high probability of finding an error.
2. A good test is not redundant
3. A good test should be "best of breed"
4. A good test should be neither too simple nor too complex.

3. test case design,

TEST CASE DESIGN STRATEGIES

a) **Black-box or behavioral testing** (knowing the specified function a product is to perform and demonstrating correct operation based solely on its specification without regard for its internal logic)
b) **White-box or glass-box** testing (knowing the internal workings of a product, tests are performed to check the workings of all independent logic paths)

4. Black-box testing (boundary value analysis, equivalence class portioning),

BLACK-BOX TESTING In this method the analyst examines the specifications to see what the program must do under various conditions. Test cases are then developed for a condition or a combination of conditions and submitted for processing. By examining the results, the analyst can determine if the specified requirements are satisfied. Black-box testing attempts to find errors in the following categories:

1. Incorrect or missing functions,
2. Interface errors,
3. Errors in data structures or external data base access,
4. Behavior or performance errors, and
5. Initialization and termination errors.

TYPES OF BLACK BOX TESTING

1. Graph-Based Testing Methods: The first step in black-box testing is to understand the objects (nodes) that are modeled in software and the relationships (links) that connect these objects. Once this has been accomplished, the next step is to define a series of tests that verify "all objects have the expected relationship to one another. Nodes are represented by circles and are connected via links. Types of Links – Unidirectional Bidirectional Parallel (more than one relationship between two nodes)

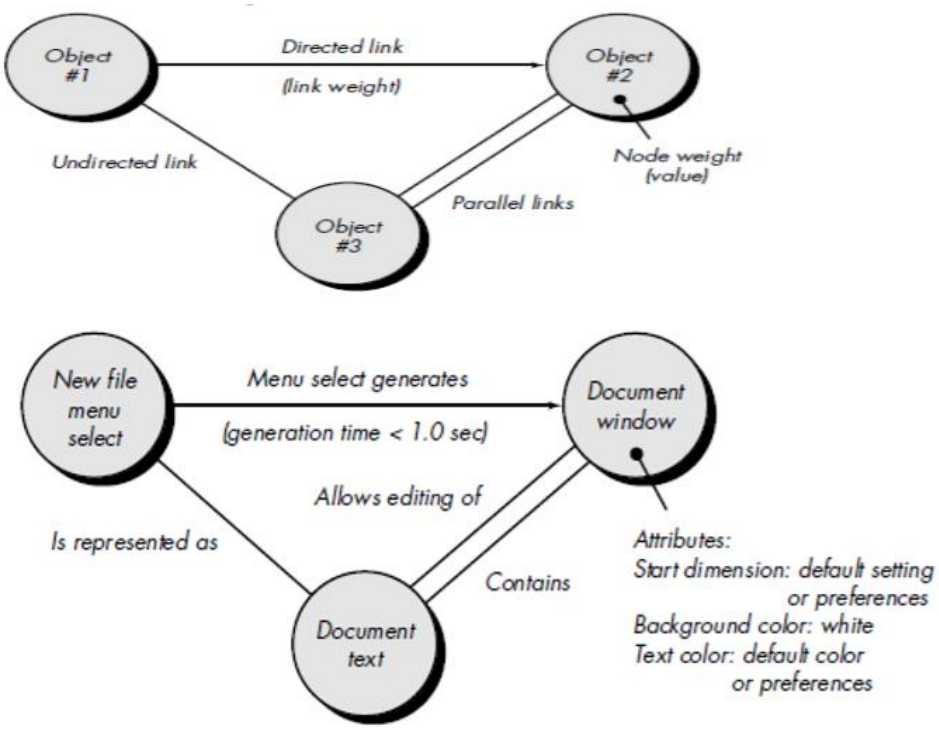


fig: An example for a word processing application

(a) Transaction Flow Modeling Nodes represent the steps in the transactions & links represent logical connections. – Data Flow Diagrams
(b) Finite State Modeling Nodes represent the user observable states & links represents the transitions. - State diagrams
(c) Data flow Modeling Nodes represent the data objects & links represent the transformations from one data object to another.
(d) Timing Modeling Nodes represent the objects and links represent the sequential connections between the objects. link weights are required execution times.

2. Equivalence Partitioning: Divide the input domain of the program into classes of data. Derive test cases from these classes. Such a test case would single handedly uncover a class of errors (say incorrect processing of all the character data). Thus, equivalence portioning strives to uncover classes of errors, thereby reducing the no of test

cases to be developed. If a set of objects are linked by relationships that are transitive, reflexive and symmetric, then equivalence is present. An equivalence class represents a set of valid or invalid states.

Equivalence classes may be defined according to the following guidelines: 1. If an input condition specifies a range, one valid and two invalid equivalence classes are defined. 2. If an input condition requires a specific value, one valid and two invalid equivalence classes are defined. 3. If an input condition specifies a member of a set, one valid and one invalid equivalence class are defined. 4. If an input condition is Boolean, one valid and one invalid class are defined.

3. Boundary Value Analysis: It is generally observed that greater no of errors occurs at the boundaries of the input domain rather than at the center. This technique leads to the development of the test cases that exercise the boundary values. Boundary value analysis is a test case design technique that complements equivalence partitioning. Rather than selecting any element of an equivalence class, BVA leads to the selection of test cases at the "edges" of the class. Rather than focusing solely on input conditions, BVA derives test cases from the output domain as well.

Guidelines a. If the input condition specifies a range bounded by values a & b, then test cases must be designed with inputs as a & b, just above and below a & b. b. Similarly, if the input condition specifies a no of values, then test cases must be developed that exercise the minimum, maximum values & also the values just above and below the maximum and the minimum. c. Apply the above guidelines also to the output i.e. design test cases such that you get the minimum and the maximum values of the outputs. d. If the internal data structures have prescribed boundaries, then test the data structures at their boundaries (array limit of 100).

4. Comparison Testing: There are some situations (e.g., aircraft avionics, automobile braking systems) in which the reliability of software is absolutely critical. In such applications redundant hardware and software are often used to minimize the possibility of error. When redundant software is developed, separate software engineering teams develop independent versions of an application using the same specification. In such situations, each version can be tested with the same test data to ensure that all provide identical output. Then all versions are executed in parallel with real-time comparison of results to ensure consistency. Comparison testing is not foolproof. If the specification from which all versions have been developed is in error, all versions will likely reflect the error. In addition, if each of the independent versions produces identical but incorrect results, condition testing will fail to detect the error.

5. Orthogonal Array Testing: Orthogonal array testing can be applied to problems in which the input domain is relatively small but too large to accommodate exhaustive testing. The orthogonal array testing method is particularly useful in finding errors associated with region faults—an error category associated with faulty logic within a software component. It is a Black-box technique that enables the design of a reasonably small set of test cases that provide maximum test coverage. Focus is on categories of faulty logic likely to be present in the software component (without examining the code) Priorities for assessing tests using an orthogonal array 1. Detect and isolate all single mode faults 2. Detect all double mode faults 3. Multimode faults

5.white-box testing(statement coverage, path coverage, cyclomatic complexity)

software verification and validation

Software Measurement

Software Measurement: A measurement is an manifestation of the size, quantity, amount or dimension of a particular attributes of a product or process. Software measurement is a titrate impute of a characteristic of a software product or the software process. It is an authority within software engineering. Software measurement process is defined and governed by ISO Standard.

Need of Software Measurement:

Software is measured to:

1. Create the quality of the current product or process.
2. Anticipate future qualities of the product or process.
3. Enhance the quality of a product or process.
4. Regulate the state of the project in relation to budget and schedule.

Classification of Software Measurement:

There are 2 types of software measurement:

1)Direct Measurement:

In direct measurement the product, process or thing is measured directly using standard scale.

2)Indirect Measurement:

In indirect measurement the quantity or quality to be measured is measured using related parameter i.e. by use of reference.

Metrics:

A metrics is a measurement of the level that any impute belongs to a system product or process. There are 4 functions related to software metrics:

1. Planning
2. Organizing
3. Controlling
4. Improving

Classification of Software Metrics:

There are 2 types of software metrics:

1. **Product Metrics:**
   Product metrics are used to evaluate the state of the product, tracing risks and undercovering prospective problem areas. The ability of team to control quality is evaluated.
2. **Process Metrics:**
   Process metrics pay particular attention on enhancing the long term process of the team or organization.
3. **Project Metrics:**
   Project matrix is describing the project characteristic and execution process.
   - Number of software developer
   - Staffing pattern over the life cycle of software
   - Cost and schedule
   - Productivity

1) Metrics for software quality

Software metrics is a standard of measure that contains many activities which involve some degree of measurement. It can be classified into three categories: product metrics, process metrics, and project metrics.

- **Product metrics** describe the characteristics of the product such as size, complexity, design features, performance, and quality level.
- **Process metrics** can be used to improve software development and maintenance. Examples include the effectiveness of defect removal during development, the pattern of testing defect arrival, and the response time of the fix process.

- **Project metrics** describe the project characteristics and execution. Examples include the number of software developers, the staffing pattern over the life cycle of the software, cost, schedule, and productivity.

Scope of Software Metrics

Software metrics contains many activities which include the following −

- Cost and effort estimation
- Productivity measures and model
- Data collection
- Quantity models and measures
- Reliability models
- Performance and evaluation models
- Structural and complexity metrics
- Capability – maturity assessment
- Management by metrics
- Evaluation of methods and tools

Software measurement is a diverse collection of these activities that range from models predicting software project costs at a specific stage to measures of program structure.

Software metrics can be classified into three categories −

- **Product metrics** − Describes the characteristics of the product such as size, complexity, design features, performance, and quality level.
- **Process metrics** − These characteristics can be used to improve the development and maintenance activities of the software.
- **Project metrics** − This metrics describe the project characteristics and execution. Examples include the number of software developers, the staffing pattern over the life cycle of the software, cost, schedule, and productivity.

Some metrics belong to multiple categories. For example, the in-process quality metrics of a project are both process metrics and project metrics.

1.Software quality metrics are a subset of software metrics that focus on the quality aspects of the product, process, and project. These are more closely associated with process and product metrics than with project metrics.

Software quality metrics can be further divided into three categories −

- Product quality metrics
- In-process quality metrics
- Maintenance quality metrics

Product Quality Metrics

This metrics include the following −

- Mean Time to Failure
- Defect Density
- Customer Problems
- Customer Satisfaction

a) Mean Time to Failure

It is the time between failures. This metric is mostly used with safety critical systems such as the airline traffic control systems, avionics, and weapons.

b) Defect Density

It measures the defects relative to the software size expressed as lines of code or function point, etc. i.e., it measures code quality per unit. This metric is used in many commercial software systems.

c) Customer Problems

It measures the problems that customers encounter when using the product. It contains the customer's perspective towards the problem space of the software, which includes the non-defect oriented problems together with the defect problems.

The problems metric is usually expressed in terms of **Problems per User-Month (PUM)**.

```
PUM = Total Problems that customers reported (true defect and non-defect
oriented
problems) for a time period + Total number of license months of the
software during
the period
```

Where,

```
Number of license-month of the software = Number of install license of the
software ×
Number of months in the calculation period
```

PUM is usually calculated for each month after the software is released to the market, and also for monthly averages by year.

d) Customer Satisfaction

Customer satisfaction is often measured by customer survey data through the five-point scale −

- Very satisfied
- Satisfied
- Neutral
- Dissatisfied
- Very dissatisfied

Satisfaction with the overall quality of the product and its specific dimensions is usually obtained through various methods of customer surveys. Based on the five-point-scale data, several metrics with slight variations can be constructed and used, depending on the purpose of analysis. For example −

- Percent of completely satisfied customers
- Percent of satisfied customers
- Percent of dis-satisfied customers
- Percent of non-satisfied customers

Usually, this percent satisfaction is used.

In-process Quality Metrics

In-process quality metrics deals with the tracking of defect arrival during formal machine testing for some organizations. This metric includes −

- Defect density during machine testing

- Defect arrival pattern during machine testing
- Phase-based defect removal pattern
- Defect removal effectiveness

a) Defect density during machine testing

Defect rate during formal machine testing (testing after code is integrated into the system library) is correlated with the defect rate in the field. Higher defect rates found during testing is an indicator that the software has experienced higher error injection during its development process, unless the higher testing defect rate is due to an extraordinary testing effort.

This simple metric of defects per KLOC or function point is a good indicator of quality, while the software is still being tested. It is especially useful to monitor subsequent releases of a product in the same development organization.

b) Defect arrival pattern during machine testing

The overall defect density during testing will provide only the summary of the defects. The pattern of defect arrivals gives more information about different quality levels in the field. It includes the following −

- The defect arrivals or defects reported during the testing phase by time interval (e.g., week). Here all of which will not be valid defects.
- The pattern of valid defect arrivals when problem determination is done on the reported problems. This is the true defect pattern.
- The pattern of defect backlog overtime. This metric is needed because development organizations cannot investigate and fix all the reported problems immediately. This is a workload statement as well as a quality statement. If the defect backlog is large at the end of the development cycle and a lot of fixes have yet to be integrated into the system, the stability of the system (hence its quality) will be affected. Retesting (regression test) is needed to ensure that targeted product quality levels are reached.

c) Phase-based defect removal pattern

This is an extension of the defect density metric during testing. In addition to testing, it tracks the defects at all phases of the development cycle, including the design reviews, code inspections, and formal verifications before testing.

Because a large percentage of programming defects is related to design problems, conducting formal reviews, or functional verifications to enhance the defect removal capability of the process at the front-end reduces error in the software. The pattern of phase-based defect removal reflects the overall defect removal ability of the development process.

With regard to the metrics for the design and coding phases, in addition to defect rates, many development organizations use metrics such as inspection coverage and inspection effort for in-process quality management.

d) Defect removal effectiveness

It can be defined as follows −

$$DRE=DefectremoveddduringadevelopmentphaseDefectslatentintheproduct\times100\%DRE=Defectremoveddduringadevelopmentphase Defectslatentintheproduct\times100\%$$

This metric can be calculated for the entire development process, for the front-end before code integration and for each phase. It is called **early defect removal** when used for the front-end and **phase effectiveness** for specific phases. The higher the value of the metric, the more effective the development process and the fewer the defects passed to the next phase or to the field. This metric is a key concept of the defect removal model for software development.

Maintenance Quality Metrics

Although much cannot be done to alter the quality of the product during this phase, following are the fixes that can be carried out to eliminate the defects as soon as possible with excellent fix quality.

- Fix backlog and backlog management index
- Fix response time and fix responsiveness
- Percent delinquent fixes
- Fix quality

a) Fix backlog and backlog management index

Fix backlog is related to the rate of defect arrivals and the rate at which fixes for reported problems become available. It is a simple count of reported problems that remain at the end of each month or each week. Using it in the format of a trend chart, this metric can provide meaningful information for managing the maintenance process.

Backlog Management Index (BMI) is used to manage the backlog of open and unresolved problems.

$$BMI=\frac{Numberofproblemsclosedduringthemonth}{Numberofproblemsarrivedduringthemonth}\times 100\%$$

If BMI is larger than 100, it means the backlog is reduced. If BMI is less than 100, then the backlog increased.

b) Fix response time and fix responsiveness

The fix response time metric is usually calculated as the mean time of all problems from open to close. Short fix response time leads to customer satisfaction.

The important elements of fix responsiveness are customer expectations, the agreed-to fix time, and the ability to meet one's commitment to the customer.

c) Percent delinquent fixes

It is calculated as follows −

$$PercentDelinquentFixes=\frac{Numberoffixesthatexceededtheresponsetimecriteriabyceveritylevel}{Numberoffixesdeliveredinaspecifiedtime}\times 100\%$$

d) Fix Quality

Fix quality or the number of defective fixes is another important quality metric for the maintenance phase. A fix is defective if it did not fix the reported problem, or if it fixed the original problem but injected a new defect. For mission-critical software, defective fixes are detrimental to customer satisfaction. The metric of percent defective fixes is the percentage of all fixes in a time interval that is defective.

A defective fix can be recorded in two ways: Record it in the month it was discovered or record it in the month the fix was delivered. The first is a customer measure; the second is a process measure. The difference between the two dates is the latent period of the defective fix.

Usually the longer the latency, the more will be the customers that get affected. If the number of defects is large, then the small value of the percentage metric will show an optimistic picture. The quality goal for the maintenance process, of course, is zero defective fixes without delinquency.

2) Software quality assurance (SQA)

Software Quality Assurance (SQA) is simply a way to assure quality in the software. It is the set of activities which ensure processes, procedures as well as standards suitable for the project and implemented correctly.
Software Quality Assurance is a process which works parallel to development of a software. It focuses on improving the process of development of software so that problems can be prevented before they become a major issue. Software Quality Assurance is a kind of an Umbrella activity that is applied throughout the software process.

Software Quality Assurance have:

1. A quality management approach
2. Formal technical reviews
3. Multi testing strategy
4. Effective software engineering technology
5. Measurement and reporting mechanism

Major Software Quality Assurance Activities:

1. **SQA Management Plan:**
   Make a plan how you will carry out the SQA throughout the project. Think which set of software engineering activities are the best for project. Check level of SQA team skills.
2. **Set the Check Points:**
   SQA team should set checkpoints. Evaluate the performance of the project on the basis of collected data on different check points.
3. **Multi testing Strategy:**
   Do not depend on single testing approach. When you have lot of testing approaches available use them.
4. **Measure Change Impact:**
   The changes for making the correction of an error sometimes re introduces more errors keep the measure of impact of change on project. Reset the new change to change check the compatibility of this fix with whole project.
5. **Manage Good Relations:**
   In the working environment managing the good relation with other teams involved in the project development is mandatory. Bad relation of SQA team with programmers' team will impact directly and badly on project. Don't play politics.

Benefits of Software Quality Assurance (SQA):

1. SQA produce high quality software.
2. High quality application saves time and cost.
3. SQA is beneficial for better reliability.
4. SQA is beneficial in the condition of no maintenance for long time.
5. High quality commercial software increase market share of company.
6. Improving the process of creating software.
7. Improves the quality of the software.

3)Software reliability

4)The IS09000 quality standards

ISO 9000 Certification

ISO (International Standards Organization) is a group or consortium of 63 countries established to plan and fosters standardization. ISO declared its 9000 series of standards in 1987. It serves as a reference for the contract between independent parties. The ISO 9000 standard determines the guidelines for maintaining a quality system. The ISO standard mainly addresses operational methods and organizational methods such as responsibilities, reporting, etc. ISO 9000 defines a set of guidelines for the production process and is not directly concerned about the product itself.

Types of ISO 9000 Quality Standards

a) ISO 9001
b) ISO 9002
c) ISO 9003

The ISO 9000 series of standards is based on the assumption that if a proper stage is followed for production, then good quality products are bound to follow automatically. The types of industries to which the various ISO standards apply are as follows.

1. **ISO 9001:** This standard applies to the organizations engaged in design, development, production, and servicing of goods. This is the standard that applies to most software development organizations.
2. **ISO 9002:** This standard applies to those organizations which do not design products but are only involved in the production. Examples of these category industries contain steel and car manufacturing industries that buy the product and plants designs from external sources and are engaged in only manufacturing those products. Therefore, ISO 9002 does not apply to software development organizations.
3. **ISO 9003:** This standard applies to organizations that are involved only in the installation and testing of the products. For example, Gas companies.

How to get ISO 9000 Certification?

An organization determines to obtain ISO 9000 certification applies to ISO registrar office for registration. The process consists of the following stages:
1)Application: Once an organization decided to go for ISO certification, it applies to the registrar for registration.
2)**Pre-Assessment:** During this stage, the registrar makes a rough assessment of the organization.
3)**Document review and Adequacy of Audit:** During this stage, the registrar reviews the document submitted by the organization and suggest an improvement.
4)**Compliance Audit:** During this stage, the registrar checks whether the organization has compiled the suggestion made by it during the review or not.
5)**Registration:** The Registrar awards the ISO certification after the successful completion of all the phases.
6)**Continue Inspection:** The registrar continued to monitor the organization time by time.

1. Object Oriented Concepts Used in UML –

1. **Class –** A class defines the blue print i.e. structure and functions of an object.
2. **Objects –** Objects help us to decompose large systems and help us to modularize our system. Modularity helps to divide our system into understandable components so that we can build our system piece by piece. An object is the fundamental unit (building block) of a system which is used to depict an entity.
3. **Inheritance –** Inheritance is a mechanism by which child classes inherit the properties of their parent classes.
4. **Abstraction –** Mechanism by which implementation details are hidden from user.
5. **Encapsulation –** Binding data together and protecting it from the outer world is referred to as encapsulation.
6. **Polymorphism –** Mechanism by which functions or entities are able to exist in different forms.

2. Unified Modeling Language (UML) | An Introduction

Unified Modeling Language (UML) is a general-purpose modelling language. The main aim of UML is to define a standard way to **visualize** the way a system has been designed. It is quite similar to blueprints used in other fields of engineering. Unified Modeling Language (UML) is a standardized general-purpose modeling language in the field of object-oriented software engineering. UML includes a set of graphic notation techniques to create visual models of object-oriented software systems. UML combines techniques from data modeling, business modeling, object modeling, and component modeling and can be used throughout the software development life-cycle and across different implementation technologies.
UML is **not a programming language**; it is rather a visual language. We use UML diagrams to portray the **behavior and structure** of a system. UML helps software engineers, businessmen and system architects with modelling, design and analysis. The Object Management Group (OMG) adopted Unified Modelling Language as a standard in 1997. It's been managed by OMG ever since. International Organization for Standardization (ISO) published UML as an approved standard in 2005. UML has been revised over the years and is reviewed periodically.
Do we really need UML?

- Complex applications need collaboration and planning from multiple teams and hence require a clear and concise way to communicate amongst them.
- Businessmen do not understand code. So, UML becomes essential to communicate with non-programmer essential requirements, functionalities and processes of the system.
- A lot of time is saved down the line when teams are able to visualize processes, user interactions and static structure of the system.

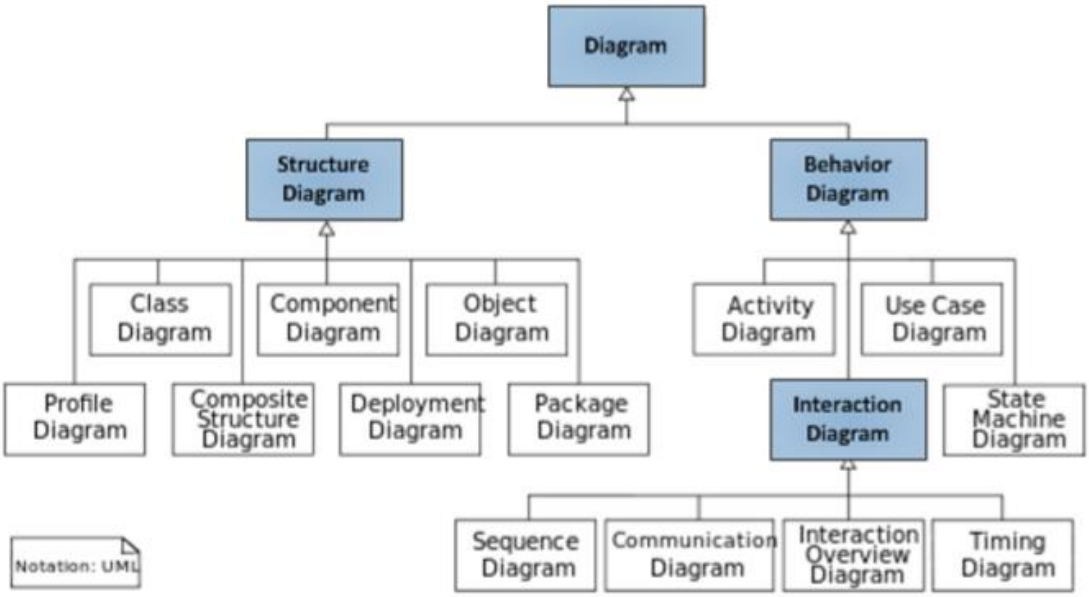
UML is linked with **object-oriented** design and analysis. UML makes the use of elements and forms associations between them to form diagrams. Diagrams in UML can be broadly classified as:

1. **Static or Structural Diagrams –** Capture static aspects or structure of a system. Structural Diagrams include: Component Diagrams, Object Diagrams, Class Diagrams and Deployment Diagrams.
2. **Dynamic or Behavior Diagrams –** Capture dynamic aspects or behavior of the system. Behavior diagrams include: Use Case Diagrams, State Diagrams, Activity Diagrams and Interaction Diagrams.

The image below shows the hierarchy of diagrams according to UML 2.2

Modeling There is a difference between a UML model and the set of diagrams of a system. A diagram is a partial graphic representation of a system's model. The model also contains documentation that drives the model elements and diagrams (such as written use cases).

UML diagrams represent two different views of a system model: Static (or structural) view This view emphasizes the static structure of the system using objects, attributes, operations, and relationships. Ex: Class diagram, Composite Structure diagram. Dynamic (or behavioral) view This view emphasizes the dynamic behavior of the system by showing collaborations among objects and changes to the internal states of objects. Ex: Sequence diagram, Activity diagram, State Machine diagram.



Structural UML Diagrams –

1. **Class Diagram –** The most widely use UML diagram is the class diagram. It is the building block of all object-oriented software systems. We use class diagrams to depict the static structure of a system by showing system's classes, their methods and attributes. Class diagrams also help us identify relationship between different classes or objects.
2. **Composite Structure Diagram –** We use composite structure diagrams to represent the internal structure of a class and its interaction points with other parts of the system. A composite structure diagram represents relationship between parts and their configuration which determine how the classifier (class, a component, or a deployment node) behaves. They represent internal structure of a structured classifier making the use of parts, ports, and connectors. We can also model collaborations using composite structure diagrams. They are similar to class diagrams except they represent individual parts in detail as compared to the entire class.
3. **Object Diagram –** An Object Diagram can be referred to as a screenshot of the instances in a system and the relationship that exists between them. Since object diagrams depict behavior when objects have been instantiated, we are able to study the behavior of the system at a particular instant. An object diagram is similar to a class diagram except it shows the instances of classes in the system. We depict actual classifiers and their relationships making the use of class diagrams. On the other hand, an Object Diagram represents specific instances of classes and relationships between them at a point of time.
4. **Component Diagram –** Component diagrams are used to represent the how the physical components in a system have been organized. We use them for modelling implementation details. Component Diagrams depict the structural relationship between software system elements and help us in understanding if functional requirements have been covered by planned development. Component Diagrams become essential to use when we design and

build complex systems. Interfaces are used by components of the system to communicate with each other.

5. **Deployment Diagram –** Deployment Diagrams are used to represent system hardware and its software. It tells us what hardware components exist and what software components run on them. We illustrate system architecture as distribution of software artifacts over distributed targets. An artifact is the information that is generated by system software. They are primarily used when a software is being used, distributed or deployed over multiple machines with different configurations.
6. **Package Diagram –** We use Package Diagrams to depict how packages and their elements have been organized. A package diagram simply shows us the dependencies between different packages and internal composition of packages. Packages help us to organize UML diagrams into meaningful groups and make the diagram easy to understand. They are primarily used to organize class and use case diagrams.

Behavior Diagrams –

1. **State Machine Diagrams –** A state diagram is used to represent the condition of the system or part of the system at finite instances of time. It's a behavioral diagram and it represents the behavior using finite state transitions. State diagrams are also referred to as **State machines** and **State-chart Diagrams.** These terms are often used interchangeably. So simply, a state diagram is used to model the dynamic behavior of a class in response to time and changing external stimuli.
2. **Activity Diagrams –** We use Activity Diagrams to illustrate the flow of control in a system. We can also use an activity diagram to refer to the steps involved in the execution of a use case. We model sequential and concurrent activities using activity diagrams. So, we basically depict workflows visually using an activity diagram. An activity diagram focuses on condition of flow and the sequence in which it happens. We describe or depict what causes a particular event using an activity diagram.
3. **Use Case Diagrams –** Use Case Diagrams are used to depict the functionality of a system or a part of a system. They are widely used to illustrate the functional requirements of the system and its interaction with external agents(actors). A use case is basically a diagram representing different scenarios where the system can be used. A use case diagram gives us a high-level view of what the system or a part of the system does without going into implementation details.
4. **Sequence Diagram –** A sequence diagram simply depicts interaction between objects in a sequential order i.e. the order in which these interactions take place. We can also use the terms event diagrams or event scenarios to refer to a sequence diagram. Sequence diagrams describe how and in what order the objects in a system function. These diagrams are widely used by businessmen and software developers to document and understand requirements for new and existing systems.
5. **Communication Diagram –** A Communication Diagram (known as Collaboration Diagram in UML 1.x) is used to show sequenced messages exchanged between objects. A communication diagram focuses primarily on objects and their relationships. We can represent similar information using Sequence diagrams; however, communication diagrams represent objects and links in a free form.
6. **Timing Diagram –** Timing Diagram are a special form of Sequence diagrams which are used to depict the behavior of objects over a time frame. We use them to show time and duration constraints which govern changes in states and behavior of objects.
7. **Interaction Overview Diagram –** An Interaction Overview Diagram models a sequence of actions and helps us simplify complex interactions into simpler occurrences. It is a mixture of activity and sequence diagrams.