Software Metrics

A software metric is a measure of software characteristics which are measurable or countable. Software metrics are valuable for many reasons, including measuring software performance, planning work items, measuring productivity, and many other uses.

Within the software development process, many metrics are that are all connected. Software metrics are similar to the four functions of management: Planning, Organization, Control, or Improvement.

Classification of Software Metrics

Software metrics can be classified into two types as follows:

1. Product Metrics: These are the measures of various characteristics of the software product. The two important software characteristics are:

1. Size and complexity of software.
2. Quality and reliability of software.

These metrics can be computed for different stages of SDLC.

2. Process Metrics: These are the measures of various characteristics of the software development process. For example, the efficiency of fault detection. They are used to measure the characteristics of methods, techniques, and tools that are used for developing software.

Types of Metrics

Internal metrics: Internal metrics are the metrics used for measuring properties that are viewed to be of greater importance to a software developer. For example, Lines of Code (LOC) measure.

External metrics: External metrics are the metrics used for measuring properties that are viewed to be of greater importance to the user, e.g., portability, reliability, functionality, usability, etc.

Hybrid metrics: Hybrid metrics are the metrics that combine product, process, and resource metrics. For example, cost per FP where FP stands for Function Point Metric.

Project metrics: Project metrics are the metrics used by the project manager to check the project's progress. Data from the past projects are used to collect various metrics, like time and cost; these estimates are used as a base of new software. Note that as the project proceeds, the project manager will check its progress from time-to-time and will compare the effort, cost, and time with the original effort, cost and time. Also understand that these metrics are used to decrease the development costs, time efforts and risks. The project quality can also be improved. As quality improves, the number of errors and time, as well as cost required, is also reduced.

Advantage of Software Metrics

Comparative study of various design methodology of software systems.

For analysis, comparison, and critical study of different programming language concerning their characteristics.

In comparing and evaluating the capabilities and productivity of people involved in software development.

In the preparation of software quality specifications.

In the verification of compliance of software systems requirements and specifications.

In making inference about the effort to be put in the design and development of the software systems.

In getting an idea about the complexity of the code.

In taking decisions regarding further division of a complex module is to be done or not.

In guiding resource manager for their proper utilization.

In comparison and making design tradeoffs between software development and maintenance cost.

In providing feedback to software managers about the progress and quality during various phases of the software development life cycle.

In the allocation of testing resources for testing the code.

Disadvantage of Software Metrics

The application of software metrics is not always easy, and in some cases, it is difficult and costly.

The verification and justification of software metrics are based on historical/empirical data whose validity is difficult to verify.

These are useful for managing software products but not for evaluating the performance of the technical staff.

The definition and derivation of Software metrics are usually based on assuming which are not standardized and may depend upon tools available and working environment.

Most of the predictive models rely on estimates of certain variables which are often not known precisely.

Software Cost Estimation

For any new software project, it is necessary to know how much it will cost to develop and how much development time will it take. These estimates are needed before development is initiated, but how is

this done? Several estimation procedures have been developed and are having the following attributes in common.

1. Project scope must be established in advanced.
2. Software metrics are used as a support from which evaluation is made.
3. The project is broken into small PCs which are estimated individually.
   To achieve true cost & schedule estimate, several option arise.
4. Delay estimation
5. Used symbol decomposition techniques to generate project cost and schedule estimates.
6. Acquire one or more automated estimation tools.

Uses of Cost Estimation

1. During the planning stage, one needs to choose how many engineers are required for the project and to develop a schedule.
2. In monitoring the project's progress, one needs to access whether the project is progressing according to the procedure and takes corrective action, if necessary.

Cost Estimation Models

A model may be static or dynamic. In a static model, a single variable is taken as a key element for calculating cost and time. In a dynamic model, all variable are interdependent, and there is no basic variable.

Static, Single Variable Models: When a model makes use of single variables to calculate desired values such as cost, time, efforts, etc. is said to be a single variable model. The most common equation is:

$$\mathbf{C=aL^b}$$

Where C = Costs
L= size
a and b are constants

The Software Engineering Laboratory established a model called SEL model, for estimating its software production. This model is an example of the static, single variable model.

$$E=1.4L^{0.93}$$
$$DOC=30.4L^{0.90}$$
$$D=4.6L^{0.26}$$

Where E= Efforts (Person Per Month)
DOC=Documentation (Number of Pages)
D = Duration (D, in months)
L = Number of Lines per code

Static, Multivariable Models: These models are based on method (1), they depend on several variables describing various aspects of the software development environment. In some model,

several variables are needed to describe the software development process, and selected equation combined these variables to give the estimate of time & cost. These models are called multivariable models.

WALSTON and FELIX develop the models at IBM provide the following equation gives a relationship between lines of source code and effort:

$$\mathbf{E=5.2L^{0.91}}$$

In the same manner duration of development is given by

$$\mathbf{D=4.1L^{0.36}}$$

The productivity index uses 29 variables which are found to be highly correlated productivity as follows:

Where $\mathbf{W_i}$ is the weight factor for the $\mathbf{i^{th}}$variable and $\mathbf{X_i=\{-1,0,+1\}}$ the estimator gives $\mathbf{X_i}$one of the values **-1, 0 or +1** depending on the variable decreases, has no effect or increases the productivity.

Example: Compare the Walston-Felix Model with the SEL model on a software development expected to involve 8 person-years of effort.

a. Calculate the number of lines of source code that can be produced.
b. Calculate the duration of the development.
c. Calculate the productivity in LOC/PY
d. Calculate the average manning

Solution:

The amount of manpower involved = 8PY=96persons-months

(a)Number of lines of source code can be obtained by reversing equation to give:

Then

L (SEL) = (96/1.4)1/0.93=94264 LOC
L (SEL) = (96/5.2)1/0.91=24632 LOC

(b)Duration in months can be calculated by means of equation

D (SEL) = 4.6 (L) 0.26
= 4.6 (94.264)0.26 = 15 months
D (W-F) = 4.1 $L^{0.36}$
= 4.1 (24.632)0.36 = 13 months

(c) Productivity is the lines of code produced per persons/month (year)

(d)Average manning is the average number of persons required per month in the project

COCOMO Model

Boehm proposed COCOMO (Constructive Cost Estimation Model) in 1981.COCOMO is one of the most generally used software estimation models in the world. COCOMO predicts the efforts and schedule of a software product based on the size of the software.

The necessary steps in this model are:

1. Get an initial estimate of the development effort from evaluation of thousands of delivered lines of source code (KDLOC).
2. Determine a set of 15 multiplying factors from various attributes of the project.
3. Calculate the effort estimate by multiplying the initial estimate with all the multiplying factors i.e., multiply the values in step1 and step2.

The initial estimate (also called nominal estimate) is determined by an equation of the form used in the static single variable models, using KDLOC as the measure of the size. To determine the initial effort E_i in person-months the equation used is of the type is shown below

E_i=a*(KDLOC)b

The value of the constant a and b are depends on the project type.

In COCOMO, projects are categorized into three types:

1. Organic
2. Semidetached
3. Embedded

1.Organic: A development project can be treated of the organic type, if the project deals with developing a well-understood application program, the size of the development team is reasonably small, and the team members are experienced in developing similar methods of projects. **Examples of this type of projects are simple business systems, simple inventory management systems, and data processing systems.**

2. Semidetached: A development project can be treated with semidetached type if the development consists of a mixture of experienced and inexperienced staff. Team members may have finite experience in related systems but may be unfamiliar with some aspects of the order being developed. **Example of Semidetached system includes developing a new operating system (OS), a Database Management System (DBMS), and complex inventory management system.**

3. Embedded: A development project is treated to be of an embedded type, if the software being developed is strongly coupled to complex hardware, or if the stringent regulations on the operational method exist. **For Example:** ATM, Air Traffic control.

For three product categories, Bohem provides a different set of expression to predict effort (in a unit of person month)and development time from the size of estimation in KLOC(Kilo Line of code) efforts estimation takes into account the productivity loss due to holidays, weekly off, coffee breaks, etc.

According to Boehm, software cost estimation should be done through three stages:

1. Basic Model
2. Intermediate Model
3. Detailed Model

1. Basic COCOMO Model: The basic COCOMO model provide an accurate size of the project parameters. The following expressions give the basic COCOMO estimation model:

$$\textbf{Effort}=a_1*(KLOC)^{a_2}\ \textbf{PM}$$
$$\textbf{Tdev}=b_1*(efforts)^{b_2}\ \textbf{Months}$$

Where

KLOC is the estimated size of the software product indicate in Kilo Lines of Code,

a_1,a_2,b_1,b_2 are constants for each group of software products,

Tdev is the estimated time to develop the software, expressed in months,

Effort is the total effort required to develop the software product, expressed in **person months (PMs)**.

Estimation of development effort

For the three classes of software products, the formulas for estimating the effort based on the code size are shown below:

Organic: Effort = 2.4(KLOC) 1.05 PM

Semi-detached: Effort = 3.0(KLOC) 1.12 PM

Embedded: Effort = 3.6(KLOC) 1.20 PM

Estimation of development time

For the three classes of software products, the formulas for estimating the development time based on the effort are given below:

Organic: Tdev = 2.5(Effort) 0.38 Months

Semi-detached: Tdev = 2.5(Effort) 0.35 Months

Embedded: Tdev = 2.5(Effort) 0.32 Months

Some insight into the basic COCOMO model can be obtained by plotting the estimated characteristics for different software sizes. Fig shows a plot of estimated effort versus product size. From fig, we can observe that the effort is somewhat superliner in the size of the software product. Thus, the effort required to develop a product increases very rapidly with project size.

The development time versus the product size in KLOC is plotted in fig. From fig it can be observed that the development time is a sub linear function of the size of the product, i.e. when the size of the product increases by two times, the time to develop the product does not double but rises moderately. This can be explained by the fact that for larger products, a larger number of activities which can be carried out concurrently can be identified. The parallel activities can be carried out simultaneously by the engineers. This reduces the time to complete the project. Further, from fig, it can be observed that the development time is roughly the same for all three categories of products. For example, a 60 KLOC program can be developed in approximately 18 months, regardless of whether it is of organic,

semidetached, or embedded type.

From the effort estimation, the project cost can be obtained by multiplying the required effort by the manpower cost per month. But, implicit in this project cost computation is the assumption that the entire project cost is incurred on account of the manpower cost alone. In addition to manpower cost, a project would incur costs due to hardware and software required for the project and the company overheads for administration, office space, etc.

It is important to note that the effort and the duration estimations obtained using the COCOMO model are called a nominal effort estimate and nominal duration estimate. The term nominal implies that if anyone tries to complete the project in a time shorter than the estimated duration, then the cost will increase drastically. But, if anyone completes the project over a longer period of time than the estimated, then there is almost no decrease in the estimated cost value.

Example1: Suppose a project was estimated to be 400 KLOC. Calculate the effort and development time for each of the three model i.e., organic, semi-detached & embedded.

Solution: The basic COCOMO equation takes the form:

$\text{Effort}=a_1*(\text{KLOC})\ a_2\ \text{PM}$
$\text{Tdev}=b_1*(\text{efforts})b_2\ \text{Months}$
Estimated Size of project= 400 KLOC

(i)Organic Mode

$E = 2.4 * (400)1.05 = 1295.31\ \text{PM}$
$D = 2.5 * (1295.31)0.38=38.07\ \text{PM}$

(ii)Semidetached Mode

$E = 3.0 * (400)1.12=2462.79\ \text{PM}$
$D = 2.5 * (2462.79)0.35=38.45\ \text{PM}$

(iii) Embedded Mode

$E = 3.6 * (400)1.20 = 4772.81\ \text{PM}$
$D = 2.5 * (4772.8)0.32 = 38\ \text{PM}$

Example2: A project size of 200 KLOC is to be developed. Software development team has average experience on similar type of projects. The project schedule is not very tight. Calculate the Effort, development time, average staff size, and productivity of the project.

Solution: The semidetached mode is the most appropriate mode, keeping in view the size, schedule and experience of development time.

Hence E=3.0(200)1.12=1133.12PM

D=2.5(1133.12)0.35=29.3PM

P = 176 LOC/PM

2. Intermediate Model: The basic Cocomo model considers that the effort is only a function of the number of lines of code and some constants calculated according to the various software systems. The intermediate COCOMO model recognizes these facts and refines the initial estimates obtained through the basic COCOMO model by using a set of 15 cost drivers based on various attributes of software engineering.

Classification of Cost Drivers and their attributes:

(i) Product attributes -

- Required software reliability extent
- Size of the application database
- The complexity of the product

Hardware attributes -

- Run-time performance constraints
- Memory constraints
- The volatility of the virtual machine environment
- Required turnabout time

Personnel attributes -

- Analyst capability
- Software engineering capability
- Applications experience
- Virtual machine experience
- Programming language experience

Project attributes -

- Use of software tools
- Application of software engineering methods
- Required development schedule

The cost drivers are divided into four categories:

Intermediate COCOMO equation:

$$E=a_i\ (KLOC)\ b_i*EAF$$
$$D=c_i\ (E)d_i$$

Project	a_i	b_i	c_i	d_i
Organic	2.4	1.05	2.5	0.38
Semidetached	3.0	1.12	2.5	0.35
Embedded	3.6	1.20	2.5	0.32

Coefficients for intermediate COCOMO

3. Detailed COCOMO Model: Detailed COCOMO incorporates all qualities of the standard version with an assessment of the cost driver?s effect on each method of the software engineering process. The detailed model uses various effort multipliers for each cost driver property. In detailed cocomo, the whole software is differentiated into multiple modules, and then we apply COCOMO in various modules to estimate effort and then sum the effort.

The Six phases of detailed COCOMO are:

1. Planning and requirements
2. System structure
3. Complete structure
4. Module code and test
5. Integration and test
6. Cost Constructive model

The effort is determined as a function of program estimate, and a set of cost drivers are given according to every phase of the software lifecycle.

What is Project?

A project is a group of tasks that need to complete to reach a clear result. A project also defines as a set of inputs and outputs which are required to achieve a goal. Projects can vary from simple to difficult and can be operated by one person or a hundred.

Projects usually described and approved by a project manager or team executive. They go beyond their expectations and objects, and it's up to the team to handle logistics and complete the project on time. For good project development, some teams split the project into specific tasks so they can manage responsibility and utilize team strengths.

What is software project management?

Software project management is an art and discipline of planning and supervising software projects. It is a sub-discipline of software project management in which software projects planned, implemented, monitored and controlled.

It is a procedure of managing, allocating and timing resources to develop computer software that fulfills requirements.

In software Project Management, the client and the developers need to know the length, period and cost of the project.

Prerequisite of software project management?

There are three needs for software project management. These are:

1. Time
2. Cost
3. Quality

It is an essential part of the software organization to deliver a quality product, keeping the cost within the client?s budget and deliver the project as per schedule. There are various factors, both external and internal, which may impact this triple factor. Any of three-factor can severely affect the other two.

Project Manager

A project manager is a character who has the overall responsibility for the planning, design, execution, monitoring, controlling and closure of a project. A project manager represents an essential role in the achievement of the projects.

A project manager is a character who is responsible for giving decisions, both large and small projects. The project manager is used to manage the risk and minimize uncertainty. Every decision the project manager makes must directly profit their project.

Role of a Project Manager:

1. Leader

A project manager must lead his team and should provide them direction to make them understand what is expected from all of them.

2. Medium:

The Project manager is a medium between his clients and his team. He must coordinate and transfer all the appropriate information from the clients to his team and report to the senior management.

3. Mentor:

He should be there to guide his team at each step and make sure that the team has an attachment. He provides a recommendation to his team and points them in the right direction.

Responsibilities of a Project Manager:

1. Managing risks and issues.
2. Create the project team and assigns tasks to several team members.
3. Activity planning and sequencing.
4. Monitoring and reporting progress.
5. Modifies the project plan to deal with the situation.

Activities

Software Project Management consists of many activities, that includes planning of the project, deciding the scope of product, estimation of cost in different terms, scheduling of tasks, etc.

The list of activities are as follows:

1. Project planning and Tracking
2. Project Resource Management
3. Scope Management
4. Estimation Management
5. Project Risk Management
6. Scheduling Management
7. Project Communication Management
8. Configuration Management

Now we will discuss all these activities -

1. Project Planning: It is a set of multiple processes, or we can say that it a task that performed before the construction of the product starts.

2. Scope Management: It describes the scope of the project. Scope management is important because it clearly defines what would do and what would not. Scope Management create the project to contain restricted and quantitative tasks, which may merely be documented and successively avoids price and time overrun.

3. Estimation management: This is not only about cost estimation because whenever we start to develop software, but we also figure out their size(line of code), efforts, time as well as cost.

If we talk about the size, then Line of code depends upon user or software requirement.

If we talk about effort, we should know about the size of the software, because based on the size we can quickly estimate how big team required to produce the software.

If we talk about time, when size and efforts are estimated, the time required to develop the software can easily determine.

And if we talk about cost, it includes all the elements such as:

- Size of software
- Quality
- Hardware
- Communication
- Training
- Additional Software and tools
- Skilled manpower

4. Scheduling Management: Scheduling Management in software refers to all the activities to complete in the specified order and within time slotted to each activity. Project managers define multiple tasks and arrange them keeping various factors in mind.

For scheduling, it is compulsory -

- Find out multiple tasks and correlate them.
- Divide time into units.
- Assign the respective number of work-units for every job.
- Calculate the total time from start to finish.
- Break down the project into modules.

5. Project Resource Management: In software Development, all the elements are referred to as resources for the project. It can be a human resource, productive tools, and libraries.

Resource management includes:

- Create a project team and assign responsibilities to every team member
- Developing a resource plan is derived from the project plan.
- Adjustment of resources.

6. Project Risk Management: Risk management consists of all the activities like identification, analyzing and preparing the plan for predictable and unpredictable risk in the project.

Several points show the risks in the project:

- The Experienced team leaves the project, and the new team joins it.
- Changes in requirement.
- Change in technologies and the environment.
- Market competition.

7. Project Communication Management: Communication is an essential factor in the success of the project. It is a bridge between client, organization, team members and as well as other stakeholders of the project such as hardware suppliers.

From the planning to closure, communication plays a vital role. In all the phases, communication must be clear and understood. Miscommunication can create a big blunder in the project.

8. Project Configuration Management: Configuration management is about to control the changes in software like requirements, design, and development of the product.

The Primary goal is to increase productivity with fewer errors.

Some reasons show the need for configuration management:

- Several people work on software that is continually update.
- Help to build coordination among suppliers.
- Changes in requirement, budget, schedule need to accommodate.
- Software should run on multiple systems.

Tasks perform in Configuration management:

- Identification
- Baseline
- Change Control
- Configuration Status Accounting
- Configuration Audits and Reviews

People involved in Configuration Management:

Project Management Tools

To manage the Project management system adequately and efficiently, we use Project management tools. **Here are some standard tools:**

Gantt chart

Henry Gantt developed the Gantt Chart in 1917. Gantt chart is a handy tool when you want to see the whole landscape of either one or multiple projects. It helps you to view which functions are dependent on one another and which event is coming up.

Gantt charts are primarily used to allocate resources to activities. The funds allocated to activities include staff, hardware, and software. Gantt charts are useful for resource planning. A Gantt chart is a particular type of bar chart where each bar represents an activity. The bars are drawn along a timeline. The distance of each bar is proportional to the duration of time planned for the corresponding event. Gantt charts are used in software project management are an enlarged version of the regular Gantt charts. In the Gantt charts used for software project management, each bar subsists of a white part and a shaded section. The shaded part of the bar displays the length of time every task is estimated to take. The white part displays the slack time, that is, the current time by which a method must be completed. A Gantt chart representation for the MIS problem, as shown in fig:

PERT chart

PERT (Project Evaluation and Review Technique) charts contain a network of boxes and arrows. The boxes show activities, and the arrows represent function dependencies. PERT chart represents the numerical variations in the plan estimates assuming a normal distribution. Thus, in a PERT chart consist of making a single estimate for each function, pessimistic, likely, and optimistic size is made. The boxes of PERT charts are generally annotated with the pessimistic, likely, and optimistic estimates for each method. Since all possible completion times between the minimum and maximum period for each process has to be treated, there are not one but many critical ways, depending on the permutations of the estimates for each purpose. This makes the analytical path method in PERT charts very complicated. A critical way in a PERT chart is shown by utilizing thicker arrows. The PERT chart representation of the MIS problem as shown in fig. PERT charts are a more sophisticated method of activity chart. Inactivity diagrams only the estimated method durations are represented. Since the actual time might vary from the estimated time, the utility of the activity diagrams is finite.

Gantt chart representation of a project record is useful in planning the usage of resources, while PERT chart is useful for monitoring the proper progress of activities. Also, it is easier to identify parallel operations in a project utilizing a PERT chart. Project managers use to determine the parallel activities in a project for assignment to various engineers.

Advantage of PERT

It forces the manager to plan.

It shows the interrelationships among the tasks in the project and, in particular, clearly identifies the critical path of the project, thus helping to focus on it.

It exposes all possible parallelism in the activities and thus helps in allocating resources.

It allows scheduling and stimulation of alternative schedules.

It enables the manager to monitor and control the project.

Logic Network

The Logic Network shows the order of activities over time. It shows the sequence in which activities are to do. Distinguishing events and pinning down the project are the two primary uses. Moreover, it will help with understanding task dependencies, a timescale, and overall project workflow.

Product Breakdown Structure

Product Breakdown Structure (BBS) is a management tool and necessary a part of the project designing. It's a task-oriented system for subdividing a project into product parts. The product breakdown structure describes subtasks or work packages and represents the connection between work packages. Within the product breakdown Structure, the project work has diagrammatically pictured with various types of lists. The product breakdown structure is just like the work breakdown structure (WBS).

Work Breakdown Structure

Work Breakdown Structure (WBS) is used to decompose a given function set recursively into small activities. WBS provides a notation for representing the significant tasks that need to be carried out to solve a problem. The problem name labels the root of the tree. Each node of the tree is destroyed down into smaller activities that are building the children of the node. Each activity is recursively decomposed into smaller sub-activities until at the leaf level; the activities require approximately two weeks to develop, as shown in fig represents the WBS of MIS (Management Information System) software. While breaking down a function into smaller tasks, the manager has to make some hard decisions. If a task is broken down into a large number of minimal activities, these can be carried out independently. Thus, it becomes feasible to develop the product faster (with the help of additional workforce). Therefore, to be able to do a project in the least amount of time, the manager needs to break the vital function into smaller ones, expecting to find more parallelism. However, it is not useful to subdivide tasks into units which take less than a week or two to execute. Excellent subdivision measure that a disproportionate amount of time must be spent on preparing and revising various charts.

Resource Histogram

The resource histogram is precisely a bar chart that used for displaying the amounts of time that a resource is scheduled to be worked on over a prearranged and specific period. Resource histograms can also contain the related feature of resource availability, used for comparison on purposes of contrast.

Critical Path Analysis

Task	ES	EF	LS	LF	ST

Specification	0	15	0	15	0
Design database	15	60	15	60	0
Design GUI part	15	45	90	120	75
Code database	60	165	60	165	0
Code GUI part	45	90	120	165	75
Integrate and test	165	285	165	285	0
Write user manual	15	75	225	285	210

Critical path analysis is a technique that is used to categorize the activities which are required to complete a task, as well as classifying the time which is needed to finish each activity and the relationships between the activities. It is also called a critical path method. CPA helps in predicting whether a project will expire on time. From the activity network representation following method can be made. The minimum time (MT) to complete the project is the maximum of all way from start to finish. The earliest start (ES) time of a method is the maximum of all paths from the start to the task. The current start time is the difference between MT and the maximum of all paths from this method to the finish. The earliest finish time (EF) of a function is the sum of the earliest start time of the function and the duration of the function. The latest finish (LF) time of a function can be obtained by subtracting maximum of all paths from this method to complete from MT. The slack time (ST) is LS - EF and equally can be indicated as LF - EF. The slack time (or float time) is the total time that a function may be delayed before it will affect the last time of the project. The slack time means the "flexibility" in the starting and completion of tasks. A critical method is one with a zero slack time. A path from the begin node to the last node containing only critical tasks is called a critical path. These parameters for various methods for the MIS problem are shown in the following table.

Software Design

Software design is a mechanism to transform user requirements into some suitable form, which helps the programmer in software coding and implementation. It deals with representing the client's requirement, as described in SRS (Software Requirement Specification) document, into a form, i.e., easily implementable using programming language.

The software design phase is the first step in **SDLC (Software Design Life Cycle)**, which moves the concentration from the problem domain to the solution domain. In software design, we consider the system to be a set of components or modules with clearly defined behaviors & boundaries.

Objectives of Software Design

Following are the purposes of Software design:

1. **Correctness:**Software design should be correct as per requirement.
2. **Completeness:**The design should have all components like data structures, modules, and external interfaces, etc.

3. **Efficiency:**Resources should be used efficiently by the program.
4. **Flexibility:**Able to modify on changing needs.
5. **Consistency:**There should not be any inconsistency in the design.
6. **Maintainability:** The design should be so simple so that it can be easily maintainable by other designers.

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

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*

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**.

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.

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.

Coupling and Cohesion

Module Coupling

In software engineering, the coupling is the degree of interdependence between software modules. Two modules that are tightly coupled are strongly dependent on each other. However, two modules that are loosely coupled are not dependent on each other. **Uncoupled modules** have no interdependence at all within them.

The various types of coupling techniques are shown in fig:

A good design is the one that has low coupling. Coupling is measured by the number of relations between the modules. That is, the coupling increases as the number of calls between modules increase or the amount of shared data is large. Thus, it can be said that a design with high coupling will have more errors.

Types of Module Coupling

1. No Direct Coupling: There is no direct coupling between M1 and M2.

In this case, modules are subordinates to different modules. Therefore, no direct coupling.

2. Data Coupling: When data of one module is passed to another module, this is called data coupling.

3. Stamp Coupling: Two modules are stamp coupled if they communicate using composite data items such as structure, objects, etc. When the module passes non-global data structure or entire structure to another module, they are said to be stamp coupled. For example, passing structure variable in C or object in C++ language to a module.

4. Control Coupling: Control Coupling exists among two modules if data from one module is used to direct the structure of instruction execution in another.

5. External Coupling: External Coupling arises when two modules share an externally imposed data format, communication protocols, or device interface. This is related to communication to external tools and devices.

6. Common Coupling: Two modules are common coupled if they share information through some global data items.

7. Content Coupling: Content Coupling exists among two modules if they share code, e.g., a branch from one module into another module.

Module Cohesion

In computer programming, cohesion defines to the degree to which the elements of a module belong together. Thus, cohesion measures the strength of relationships between pieces of functionality within a given module. For example, in highly cohesive systems, functionality is strongly related.

Cohesion is an **ordinal** type of measurement and is generally described as "high cohesion" or "low cohesion."

Types of Modules Cohesion

1. **Functional Cohesion:** Functional Cohesion is said to exist if the different elements of a module, cooperate to achieve a single function.
2. **Sequential Cohesion:** A module is said to possess sequential cohesion if the element of a module form the components of the sequence, where the output from one component of the sequence is input to the next.
3. **Communicational Cohesion:** A module is said to have communicational cohesion, if all tasks of the module refer to or update the same data structure, e.g., the set of functions defined on an array or a stack.
4. **Procedural Cohesion:** A module is said to be procedural cohesion if the set of purpose of the module are all parts of a procedure in which particular sequence of steps has to be carried out for achieving a goal, e.g., the algorithm for decoding a message.
5. **Temporal Cohesion:** When a module includes functions that are associated by the fact that all the methods must be executed in the same time, the module is said to exhibit temporal cohesion.
6. **Logical Cohesion:** A module is said to be logically cohesive if all the elements of the module perform a similar operation. For example Error handling, data input and data output, etc.
7. **Coincidental Cohesion:** A module is said to have coincidental cohesion if it performs a set of tasks that are associated with each other very loosely, if at all.

Coupling	Cohesion
Coupling is also called Inter-Module Binding.	Cohesion is also called Intra-Module Binding.
Coupling shows the relationships between modules.	Cohesion shows the relationship within the module.

Differentiate between Coupling and Cohesion

Coupling shows the relative **independence** between the modules.	Cohesion shows the module's relative **functional** strength.
While creating, you should aim for low coupling, i.e., dependency among modules should be less.	While creating you should aim for high cohesion, i.e., a cohesive component/ module focuses on a single function (i.e., single-mindedness) with little interaction with other modules of the system.
In coupling, modules are linked to the other modules.	In cohesion, the module focuses on a single thing.

Function Oriented Design

Function Oriented design is a method to software design where the model is decomposed into a set of interacting units or modules where each unit or module has a clearly defined function. Thus, the system is designed from a functional viewpoint.

Design Notations

Design Notations are primarily meant to be used during the process of design and are used to represent design or design decisions. For a function-oriented design, the design can be represented graphically or mathematically by the following:

Data Flow Diagram

Data-flow design is concerned with designing a series of functional transformations that convert system inputs into the required outputs. The design is described as data-flow diagrams. These diagrams show how data flows through a system and how the output is derived from the input through a series of functional transformations.

Data-flow diagrams are a useful and intuitive way of describing a system. They are generally understandable without specialized training, notably if control information is excluded. They show end-to-end processing. That is the flow of processing from when data enters the system to where it leaves the system can be traced.

Data-flow design is an integral part of several design methods, and most CASE tools support data-flow diagram creation. Different ways may use different icons to represent data-flow diagram entities, but their meanings are similar.

The notation which is used is based on the following symbols:

The report generator produces a report which describes all of the named entities in a data-flow diagram. The user inputs the name of the design represented by the diagram. The report generator

then finds all the names used in the data-flow diagram. It looks up a data dictionary and retrieves information about each name. This is then collated into a report which is output by the system.

Data Dictionaries

A data dictionary lists all data elements appearing in the DFD model of a system. The data items listed contain all data flows and the contents of all data stores looking on the DFDs in the DFD model of a system.

A data dictionary lists the objective of all data items and the definition of all composite data elements in terms of their component data items. For example, a data dictionary entry may contain that the data *grossPay* consists of the parts *regularPay* and *overtimePay*.

grossPay = regularPay + overtimePay

For the smallest units of data elements, the data dictionary lists their name and their type.

A data dictionary plays a significant role in any software development process because of the following reasons:

- A Data dictionary provides a standard language for all relevant information for use by engineers working in a project. A consistent vocabulary for data items is essential since, in large projects, different engineers of the project tend to use different terms to refer to the same data, which unnecessarily causes confusion.
- The data dictionary provides the analyst with a means to determine the definition of various data structures in terms of their component elements.

Structured Charts

It partitions a system into block boxes. A Black box system that functionality is known to the user without the knowledge of internal design.

Structured Chart is a graphical representation which shows:

- System partitions into modules
- Hierarchy of component modules
- The relation between processing modules
- Interaction between modules
- Information passed between modules

The following notations are used in structured chart:

Pseudo-code

Pseudo-code notations can be used in both the preliminary and detailed design phases. Using pseudo-code, the designer describes system characteristics using short, concise, English Language phases that are structured by keywords such as If-Then-Else, While-Do, and End.

Object-Oriented Design

In the object-oriented design method, the system is viewed as a collection of objects (i.e., entities). The state is distributed among the objects, and each object handles its state data. For example, in a Library Automation Software, each library representative may be a separate object with its data and functions to operate on these data. The tasks defined for one purpose cannot refer or change data of other objects. Objects have their internal data which represent their state. Similar objects create a class. In other words, each object is a member of some class. Classes may inherit features from the superclass.

The different terms related to object design are:

1. **Objects:** All entities involved in the solution design are known as objects. For example, person, banks, company, and users are considered as objects. Every entity has some attributes associated with it and has some methods to perform on the attributes.
2. **Classes:** A class is a generalized description of an object. An object is an instance of a class. A class defines all the attributes, which an object can have and methods, which represents the functionality of the object.
3. **Messages:** Objects communicate by message passing. Messages consist of the integrity of the target object, the name of the requested operation, and any other action needed to perform the function. Messages are often implemented as procedure or function calls.
4. **Abstraction** In object-oriented design, complexity is handled using abstraction. Abstraction is the removal of the irrelevant and the amplification of the essentials.
5. **Encapsulation:** Encapsulation is also called an information hiding concept. The data and operations are linked to a single unit. Encapsulation not only bundles essential information of an object together but also restricts access to the data and methods from the outside world.
6. **Inheritance:** OOD allows similar classes to stack up in a hierarchical manner where the lower or sub-classes can import, implement, and re-use allowed variables and functions from their immediate superclasses.This property of OOD is called an inheritance. This makes it easier to define a specific class and to create generalized classes from specific ones.
7. **Polymorphism:** OOD languages provide a mechanism where methods performing similar tasks but vary in arguments, can be assigned the same name. This is known as polymorphism, which allows a single interface is performing functions for different types. Depending upon how the service is invoked, the respective portion of the code gets executed.

Software Reliability

Software Reliability means **Operational reliability**. It is described as the ability of a system or component to perform its required functions under static conditions for a specific period.

Software reliability is also defined as the probability that a software system fulfills its assigned task in a given environment for a predefined number of input cases, assuming that the hardware and the input are free of error.

Software Reliability is an essential connect of software quality, composed with functionality, usability, performance, serviceability, capability, installability, maintainability, and documentation. Software Reliability is hard to achieve because the complexity of software turn to be high. While any system with a high degree of complexity, containing software, will be hard to reach a certain level of reliability, system developers tend to push complexity into the software layer, with the speedy growth of system size and ease of doing so by upgrading the software.

For example, large next-generation aircraft will have over 1 million source lines of software on-board; next-generation air traffic control systems will contain between one and two million lines; the upcoming International Space Station will have over two million lines on-board and over 10 million lines of ground support software; several significant life-critical defense systems will have over 5 million source lines of software. While the complexity of software is inversely associated with software reliability, it is directly related to other vital factors in software quality, especially functionality, capability, etc.

Software Failure Mechanisms

The software failure can be classified as:

Transient failure: These failures only occur with specific inputs.

Permanent failure: This failure appears on all inputs.

Recoverable failure: System can recover without operator help.

Unrecoverable failure: System can recover with operator help only.

Non-corruption failure: Failure does not corrupt system state or data.

Corrupting failure: It damages the system state or data.

Software failures may be due to bugs, ambiguities, oversights or misinterpretation of the specification that the software is supposed to satisfy, carelessness or incompetence in writing code, inadequate testing, incorrect or unexpected usage of the software or other unforeseen problems.

Hardware vs. Software Reliability

Hardware Reliability	Software Reliability
Hardware faults are mostly physical faults.	Software faults are design faults, which are tough to visualize, classify, detect, and correct.
Hardware components generally fail due to wear and tear.	Software component fails due to bugs.

In hardware, design faults may also exist, but physical faults generally dominate.	In software, we can simply find a strict corresponding counterpart for "**manufacturing**" as the hardware manufacturing process, if the simple action of uploading software modules into place does not count. Therefore, the quality of the software will not change once it is uploaded into the storage and start running
Hardware exhibits the failure features shown in the following figure: It is called the **bathtub curve**. Period A, B, and C stand for burn-in phase, useful life phase, and end-of-life phase respectively.	Software reliability does not show the same features similar as hardware. A possible curve is shown in the following figure: If we projected software reliability on the same axes.

There are two significant differences between hardware and software curves are:

One difference is that in the last stage, the software does not have an **increasing failure rate** as hardware does. In this phase, the software is approaching **obsolescence**; there are no motivations for any upgrades or changes to the software. Therefore, the failure rate will not change.

The second difference is that in the **useful-life** phase, the software will experience a radical increase in failure rate each time an upgrade is made. The failure rate levels off gradually, partly because of the defects create and fixed after the updates.

The upgrades in above figure signify **feature upgrades**, not upgrades for reliability. For feature upgrades, the complexity of software is possible to be increased, since the functionality of the software is enhanced. Even error fixes may be a reason for more software failures if the bug fix induces other defects into the software. For **reliability upgrades**, it is likely to incur a drop in software failure rate, if the objective of the upgrade is enhancing software reliability, such as a redesign or reimplementation of some modules using better engineering approaches, such as clean-room method.

A partial list of the distinct features of software compared to hardware is listed below:

Failure cause: Software defects are primarily designed defects.

Wear-out: Software does not have an energy-related wear-out phase. Bugs can arise without warning.

Repairable system: Periodic restarts can help fix software queries.

Time dependency and life cycle: Software reliability is not a purpose of operational time.

Environmental factors: Do not affect Software reliability, except it may affect program inputs.

Reliability prediction: Software reliability cannot be predicted from any physical basis since it depends entirely on human factors in design.

Redundancy: It cannot improve Software reliability if identical software elements are used.

Interfaces: Software interfaces are merely conceptual other than visual.

Failure rate motivators: It is generally not predictable from analyses of separate statements.

Built with standard components: Well-understood and extensively tested standard element will help improve maintainability and reliability. But in the software industry, we have not observed this trend. Code reuse has been around for some time but to a minimal extent. There are no standard elements for software, except for some standardized logic structures.

Software Reliability Measurement Techniques

Reliability metrics are used to quantitatively expressed the reliability of the software product. The option of which parameter is to be used depends upon the type of system to which it applies & the requirements of the application domain.

Measuring software reliability is a severe problem because we don't have a good understanding of the nature of software. It is difficult to find a suitable method to measure software reliability and most of the aspects connected to software reliability. Even the software estimates have no uniform definition. If we cannot measure the reliability directly, something can be measured that reflects the features related to reliability.

The current methods of software reliability measurement can be divided into four categories:

1. Product Metrics

Product metrics are those which are used to build the artifacts, i.e., requirement specification documents, system design documents, etc. These metrics help in the assessment if the product is right sufficient through records on attributes like usability, reliability, maintainability & portability. In these measurements are taken from the actual body of the source code.

i. Software size is thought to be reflective of complexity, development effort, and reliability. Lines of Code **(LOC)**, or LOC in thousands **(KLOC)**, is an initial intuitive approach to measuring software size. The basis of **LOC** is that program length can be used as a predictor of program characteristics such as effort &ease of maintenance. It is a measure of the functional complexity of the program and is independent of the programming language.

ii. Function point metric is a technique to measure the functionality of proposed software development based on the count of inputs, outputs, master files, inquires, and interfaces.

iii. Test coverage metric size fault and reliability by performing tests on software products, assuming that software reliability is a function of the portion of software that is successfully verified or tested.

iv. Complexity is directly linked to software reliability, so representing complexity is essential. Complexity-oriented metrics is a way of determining the complexity of a program's control structure by simplifying the code into a graphical representation. The representative metric is McCabe's Complexity Metric.

v. Quality metrics measure the quality at various steps of software product development. An vital quality metric is **Defect Removal Efficiency (DRE)**. DRE provides a measure of quality because of different quality assurance and control activities applied throughout the development process.

2. Project Management Metrics

Project metrics define project characteristics and execution. If there is proper management of the project by the programmer, then this helps us to achieve better products. A relationship exists between the development process and the ability to complete projects on time and within the desired quality objectives. Cost increase when developers use inadequate methods. Higher reliability can be achieved by using a better development process, risk management process, configuration management process.

These metrics are:

- Number of software developers
- Staffing pattern over the life-cycle of the software
- Cost and schedule
- Productivity

3. Process Metrics

Process metrics quantify useful attributes of the software development process & its environment. They tell if the process is functioning optimally as they report on characteristics like cycle time & rework time. The goal of process metric is to do the right job on the first time through the process. The quality of the product is a direct function of the process. So process metrics can be used to estimate, monitor, and improve the reliability and quality of software. Process metrics describe the effectiveness and quality of the processes that produce the software product.

Examples are:

- The effort required in the process
- Time to produce the product
- Effectiveness of defect removal during development
- Number of defects found during testing
- Maturity of the process

4. Fault and Failure Metrics

A fault is a defect in a program which appears when the programmer makes an error and causes failure when executed under particular conditions. These metrics are used to determine the failure-free execution software.

To achieve this objective, a number of faults found during testing and the failures or other problems which are reported by the user after delivery are collected, summarized, and analyzed. Failure metrics are based upon customer information regarding faults found after release of the software. The failure data collected is therefore used to calculate failure density, **Mean Time between Failures (MTBF)**, or other parameters to measure or predict software reliability.

Reliability Metrics

Reliability metrics are used to quantitatively expressed the reliability of the software product. The option of which metric is to be used depends upon the type of system to which it applies & the requirements of the application domain.

Some reliability metrics which can be used to quantify the reliability of the software product are as follows:

1. Mean Time to Failure (MTTF)

MTTF is described as the time interval between the two successive failures. An **MTTF** of 200 mean that one failure can be expected each 200-time units. The time units are entirely dependent on the system & it can even be stated in the number of transactions. **MTTF** is consistent for systems with large transactions.

For example, It is suitable for computer-aided design systems where a designer will work on a design for several hours as well as for Word-processor systems.

To measure **MTTF**, we can evidence the failure data for n failures. Let the failures appear at the time instants $\mathbf{t_1,t_2.....t_n}$.

MTTF can be calculated as

2. Mean Time to Repair (MTTR)

Once failure occurs, some-time is required to fix the error. **MTTR** measures the average time it takes to track the errors causing the failure and to fix them.

3. Mean Time Between Failure (MTBR)

We can merge **MTTF** & **MTTR** metrics to get the MTBF metric.

MTBF = MTTF + MTTR

Thus, an **MTBF** of 300 denoted that once the failure appears, the next failure is expected to appear only after 300 hours. In this method, the time measurements are real-time & not the execution time as in **MTTF**.

4. Rate of occurrence of failure (ROCOF)

It is the number of failures appearing in a unit time interval. The number of unexpected events over a specific time of operation. **ROCOF** is the frequency of occurrence with which unexpected role is likely to appear. A **ROCOF** of 0.02 mean that two failures are likely to occur in each 100 operational time unit steps. It is also called the failure intensity metric.

5. Probability of Failure on Demand (POFOD)

POFOD is described as the probability that the system will fail when a service is requested. It is the number of system deficiency given several systems inputs.

POFOD is the possibility that the system will fail when a service request is made.

A **POFOD** of 0.1 means that one out of ten service requests may fail.**POFOD** is an essential measure for safety-critical systems. POFOD is relevant for protection systems where services are demanded occasionally.

6. Availability (AVAIL)

Availability is the probability that the system is applicable for use at a given time. It takes into account the repair time & the restart time for the system. An availability of 0.995 means that in every 1000 time units, the system is feasible to be available for **995** of these. The percentage of time that a system is applicable for use, taking into account planned and unplanned downtime. If a system is down an average of four hours out of 100 hours of operation, its **AVAIL** is 96%.

Software Metrics for Reliability

The Metrics are used to improve the reliability of the system by identifying the areas of requirements.

Different Types of Software Metrics are:-

Requirements Reliability Metrics

Requirements denote what features the software must include. It specifies the functionality that must be contained in the software. The requirements must be written such that is no misconception between the developer & the client. The requirements must include valid structure to avoid the loss of valuable data.

The requirements should be thorough and in a detailed manner so that it is simple for the design stage. The requirements should not include inadequate data. Requirement Reliability metrics calculates the above-said quality factors of the required document.

Design and Code Reliability Metrics

The quality methods that exists in design and coding plan are complexity, size, and modularity. Complex modules are tough to understand & there is a high probability of occurring bugs. The reliability will reduce if modules have a combination of high complexity and large size or high complexity and small size. These metrics are also available to object-oriented code, but in this, additional metrics are required to evaluate the quality.

Testing Reliability Metrics

These metrics use two methods to calculate reliability.

First, it provides that the system is equipped with the tasks that are specified in the requirements. Because of this, the bugs due to the lack of functionality reduces.

The **second** method is calculating the code, finding the bugs & fixing them. To ensure that the system includes the functionality specified, test plans are written that include multiple test cases. Each test method is based on one system state and tests some tasks that are based on an associated set of requirements. The goals of an effective verification program is to ensure that each elements is tested, the implication being that if the system passes the test, the requirement?s functionality is contained in the delivered system.

Software Fault Tolerance

Software fault tolerance is the ability for software to detect and recover from a fault that is happening or has already happened in either the software or hardware in the system in which the software is running to provide service by the specification.

Software fault tolerance is a necessary component to construct the next generation of highly available and reliable computing systems from embedded systems to data warehouse systems.

To adequately understand software fault tolerance it is important to understand the nature of the problem that software fault tolerance is supposed to solve.

Software faults are all design faults. Software manufacturing, the reproduction of software, is considered to be perfect. The source of the problem being solely designed faults is very different than almost any other system in which fault tolerance is the desired property.

Software Fault Tolerance Techniques

1. Recovery Block

The **recovery block method** is a simple technique developed by Randel. The recovery block operates with an adjudicator, which confirms the results of various implementations of the same algorithm. In a system with recovery blocks, the system view is broken down into fault recoverable blocks.

The entire system is constructed of these fault-tolerant blocks. Each block contains at least a primary, secondary, and exceptional case code along with an adjudicator. The adjudicator is the component, which determines the correctness of the various blocks to try.

The adjudicator should be kept somewhat simple to maintain execution speed and aide in correctness. Upon first entering a unit, the adjudicator first executes the primary alternate. (There may be N alternates in a unit which the adjudicator may try.) If the adjudicator determines that the fundamental block failed, it then tries to roll back the state of the system and tries the secondary alternate.

If the adjudicator does not accept the results of any of the alternates, it then invokes the exception handler, which then indicates the fact that the software could not perform the requested operation.

The **recovery block technique** increases the pressure on the specification to be specific enough to create various multiple alternatives that are functionally the same. This problem is further discussed in the context of the **N-version software method**.

2. N-Version Software

The **N-version software** methods attempt to parallel the traditional hardware fault tolerance concept of N-way redundant hardware. In an **N-version software system**, every module is done with up to N different methods. Each variant accomplishes the same function, but hopefully in a various way. Each version then submits its answer to voter or decider, which decides the correct answer, and returns that as the result of the module.

This system can hopefully overcome the design faults present in most software by relying upon the design diversity concept. An essential distinction in **N-version software** is the fact that the system could include multiple types of hardware using numerous versions of the software.

N-version software can only be successful and successfully tolerate faults if the required design diversity is met. The dependence on appropriate specifications in N-version software, (and recovery blocks,) cannot be stressed enough.

3. N-Version Software and Recovery Blocks

The differences between the **recovery block technique** and the **N-version technique** are not too numerous, but they are essential. In traditional recovery blocks, each alternative would be executed serially until an acceptable solution is found as determined by the adjudicator. The recovery block method has been extended to contain concurrent execution of the various alternatives.

The **N-version techniques** have always been designed to be implemented using N-way hardware concurrently. In a serial retry system, the cost in time of trying multiple methods may be too expensive, especially for a real-time system. Conversely, concurrent systems need the expense of N-way hardware and a communications network to connect them.

The recovery block technique requires that each module build a specific adjudicator; in the N-version method, a single decider may be used. The recovery block technique, assuming that the programmer can create a sufficiently simple adjudicator, will create a system, which is challenging to enter into an incorrect state.

Software Testing Tutorial

Software testing tutorial provides basic and advanced concepts of software testing. Our software testing tutorial is designed for beginners and professionals.

Software testing is widely used technology because it is compulsory to test each and every software before deployment.

Our Software testing tutorial includes all topics of Software testing such as Methods such as BlackBox Testing, WhiteBox Testing, VisualBox Testing and GrayBox Testing. Levels such as Unit Testing, Integration Testing, Regression Testing, Functional Testing. System Testing, Acceptance Testing, Alpha Testing, Beta Testing, Non-Functional testing, Security Testing, Portability Testing.

What is Software Testing

Software testing is a process of identifying the correctness of a software by considering its all attributes (Reliability, Scalability, Portability, Re-usability, Usability) and evaluating the execution of software components to find the software bugs or errors or defects.

Software testing provides an independent view and objective of the software and gives surety of fitness of the software. It involves testing of all components under the required services to confirm that whether it is satisfying the specified requirements or not. The process is also providing the client with information about the quality of the software.

Testing is mandatory because it will be a dangerous situation if the software fails any of time due to lack of testing. So, without testing software cannot be deployed to the end user.

What is Testing

Testing is a group of techniques to determine the correctness of the application under the predefined script but, testing cannot find all the defect of application. The main intent of testing is to detect failures of the application so that failures can be discovered and corrected. It does not demonstrate that a product functions properly under all conditions but only that it is not working in some specific conditions.

Testing furnishes comparison that compares the behavior and state of software against mechanisms because the problem can be recognized by the mechanism. The mechanism may include past versions of the same specified product, comparable products, and interfaces of expected purpose, relevant standards, or other criteria but not limited up to these.

Testing includes an examination of code and also the execution of code in various environments, conditions as well as all the examining aspects of the code. In the current scenario of software

development, a testing team may be separate from the development team so that Information derived from testing can be used to correct the process of software development.

The success of software depends upon acceptance of its targeted audience, easy graphical user interface, strong functionality load test, etc. For example, the audience of banking is totally different from the audience of a video game. Therefore, when an organization develops a software product, it can assess whether the software product will be beneficial to its purchasers and other audience.

Manual Testing

Manual testing is a software testing process in which test cases are executed manually without using any automated tool. All test cases executed by the tester manually according to the end user's perspective. It ensures whether the application is working as mentioned in the requirement document or not. Test cases are planned and implemented to complete almost 100 percent of the software application. Test case reports are also generated manually.

Manual Testing is one of the most fundamental testing processes as it can find both visible and hidden defects of the software. The difference between expected output and output, given by the software is defined as a defect. The developer fixed the defects and handed it to the tester for retesting.

Manual testing is mandatory for every newly developed software before automated testing. This testing requires great efforts and time, but it gives the surety of bug-free software. Manual Testing requires knowledge of manual testing techniques but not of any automated testing tool.

Manual testing is essential because one of the software testing fundamentals is "100% automation is not possible."

There are various methods used for manual testing. Each method is used according to its testing criteria. Types of manual testing are given below:

Types of Manual Testing:

1. Black Box Testing
2. White Box Testing
3. Unit Testing
4. System Testing
5. Integration Testing
6. Acceptance Testing

How to perform Manual Testing

- First, tester examines all documents related to software, to select testing areas.
- Tester analyses requirement document to cover all requirements stated by the customer.
- Tester develops the test cases according to the requirement document.
- All test cases are executed manually by using Black box testing and white box testing.
- If bugs occurred then the testing team informs to the development team.
- Development team fixes bugs and handed software to the testing team for retesting.

Advantages of Manual Testing

- It does not require programming knowledge while using the Black box method.
- It is used to test dynamically changing GUI designs.
- Tester interacts with software as a real user so that they are able to discover usability and user interface issues.
- It ensures that the software is a hundred percent bug-free.
- It is cost effective.
- Easy to learn for new testers.

Disadvantages of Manual Testing

- It requires a large number of human resources.
- It is very time-consuming.
- Tester develops test cases based on their skills and experience. There is no evidence that they have covered all functions or not.
- Test cases cannot be used again. Need to develop separate test cases for each new software.
- It does not provide testing on all aspects of testing.
- Since two teams work together, sometimes it is difficult to understand each other's motives, it can mislead the process.

Manual testing tools

Selenium

Selenium is used to test the Web Application.

Appium

Appium is used to test the mobile application.

TestLink

TestLink is used for test management.

Postman

Postman is used for API testing.

Firebug

Firebug is an online debugger.

JMeter

JMeter is used for load testing of any application.

Mantis

Mantis is used for bug tracking.

Automation Testing

When the testing case suites are performed by using automated testing tools is known as Automation Testing. The testing process is done by using special automation tools to control the execution of test cases and compare the actual result with the expected result. Automation testing requires a pretty huge investment of resources and money.

Generally, repetitive actions are tested in automated testing such as regression tests. The testing tools used in automation testing are used not only for regression testing but also for automated GUI interaction, data set up generation, defect logging, and product installation.

The goal of automation testing is to reduce manual test cases but not to eliminate any of them. Test suits can be recorded by using the automation tools, and tester can play these suits again as per the requirement. Automated testing suites do not require any human intervention.

The life cycle of Automation Testing

The life cycle of automation testing is a systematic approach to organize and execute testing activities in a manner that provides maximum test coverage with limited resources. The structure of the test involves a multi-step process that supports the required, detailed and inter-related activities to perform the task.

The life cycle of automation testing consists of the following components:

Decision to Automation Testing

It is the first phase of **Automation Test Life-cycle Methodology (ATLM)**. At this phase, the main focus of the testing team is to manage expectations from the test and find out the potential benefits if applying the automated testing correctly.

On adopting an automated testing suit, organizations have to face many issues, some are listed below:

- Testing tool experts are required for automation testing, so the first issue, to appoint a testing equipment specialist.
- The second issue is, choose the exact tool for the testing of a particular function.
- The issue of design and development standards in the implementation of an automated testing process.
- Analysis of various automated testing tools to choose the best tool for automation testing.
- The issue of money and time occurs as the consumption of money and time is high in the beginning of the testing.

Test Tool Selection

Test Tool Selection represents the second phase of the **Automation Test Life-cycle Methodology (ATLM)**. This phase guides the tester in the evaluation and selection of the testing tool.

Since the testing tool supports almost all testing requirements, the tester still needs to review the system engineering environment and other organizational needs and then make a list of evaluation parameters of the tools. Test engineers evaluate the equipment based on the provided sample criteria.

Scope Introduction

This phase represents the third phase of **Automation Test Life-cycle Methodology (ATLM)**. The scope of automation includes the testing area of the application. The determination of scope is based on the following points:

- Common functionalities of the software application that are held by every software application.
- Automation test sets the reusable range of business components.
- Automation Testing decides the extent of reusability of the business components.
- An application should have business-specific features and must be technically feasible.
- Automation testing provides the repetition of test cases in the case of cross-browser testing.

This phase ensures the overall testing strategy that should be well managed and modified if required. In order to ensure the availability of skills, testing skills of a particular member and whole team are analyzed against the required specific skills for a particular software application.

Test Planning and Development

Test planning and development is the fourth and most important phase of Automation Test Life -cycle Methodology (ATLM) because all the testing strategies are defined here. Planning of long -lead test activities, the creation of standards and guidelines,an arrangement of the required combination of hardware, software and network to create a test environment, defect tracking procedure, guidelines to control test configuration and environment all are identified in this phase. Tester determines estimated effort and cost for the entire project. Test strategy and effort estimation documents are the deliverables provided by this phase. Test case execution can be started after the successful completion of test planning.

Test Case Execution

Test case Execution is the sixth phase of Automation Test Life -cycle Methodology (ATLM). It takes place after the successful completion of test planning. At this stage, the testing team defines test design and development. Now, test cases can be executed under product testing.In this phase, the testing team starts case development and execution activity by using automated tools. The prepared test cases are reviewed by peer members of the testing team or quality assurance leaders.

During the execution of test procedures, the testing team directed to comply with the execution schedule. Execution phase implements the strategies such as integration, acceptance and unit testing that have defined in the test plan previously.

Review and Assessment

Review and assessment is the sixth and final stage of the automated testing life cycle but the activities of this phase are conducted throughout the whole life cycle to maintain continuous quality improvement. The improvement process is done via the evaluation of matrices, review and assessment of the activities.

During the review, the examiner concentrates whether the particular metric satisfies the acceptance criteria or not, if yes, then it is ready to use in software production.It is comprehensive as test cases cover each feature of the application.

The test team performs its own survey to inquire about the potential value of the process; if the potential benefit is less than sufficient, the testing team can change the testing tool. The team also provides a sample survey form to ask for feedback from the end user about the attributes and management of the software product.

Advantages of Automation Testing

- Automation testing takes less time than manual testing.
- A tester can test the response of the software if the execution of the same operation is repeated several times.
- Automation Testing provides re-usability of test cases on testing of different versions of the same software.
- Automation testing is reliable as it eliminates hidden errors by executing test cases again in the same way.
- Automation Testing is comprehensive as test cases cover each and every feature of the application.
- It does not require many human resources, instead of writing test cases and testing them manually, they need an automation testing engineer to run them.
- The cost of automation testing is less than manual testing because it requires a few human resources.

Disadvantages of Automation Testing

- Automation Testing requires high-level skilled testers.
- It requires high-quality testing tools.
- When it encounters an unsuccessful test case, the analysis of the whole event is complicated.
- Test maintenance is expensive because high fee license testing equipment is necessary.
- Debugging is mandatory if a less effective error has not been solved, it can lead to fatal results.

White Box Testing

The box testing approach of software testing consists of black box testing and white box testing. We are discussing here white box testing which also known as glass box is **testing, structural testing, clear box testing, open box testing and transparent box testing**. It tests internal coding and infrastructure of a software focus on checking of predefined inputs against expected and desired outputs. It is based on inner workings of an application and revolves around internal structure testing. In this type of testing programming skills are required to design test cases. The primary goal of white box testing is to focus on the flow of inputs and outputs through the software and strengthening the security of the software.

The term 'white box' is used because of the internal perspective of the system. The clear box or white box or transparent box name denote the ability to see through the software's outer shell into its inner workings.

Test cases for white box testing are derived from the design phase of the software development lifecycle. Data flow testing, control flow testing, path testing, branch testing, statement and decision coverage all these techniques used by white box testing as a guideline to create an error-free software.

White box testing follows some working steps to make testing manageable and easy to understand what the next task to do. There are some basic steps to perform white box testing.

Generic steps of white box testing

- Design all test scenarios, test cases and prioritize them according to high priority number.
- This step involves the study of code at runtime to examine the resource utilization, not accessed areas of the code, time taken by various methods and operations and so on.
- In this step testing of internal subroutines takes place. Internal subroutines such as nonpublic methods, interfaces are able to handle all types of data appropriately or not.
- This step focuses on testing of control statements like loops and conditional statements to check the efficiency and accuracy for different data inputs.
- In the last step white box testing includes security testing to check all possible security loopholes by looking at how the code handles security.

Reasons for white box testing

- It identifies internal security holes.
- To check the way of input inside the code.
- Check the functionality of conditional loops.
- To test function, object, and statement at an individual level.

Advantages of White box testing

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

Disadvantages of White box testing

- White box testing is too much time consuming when it comes to large-scale programming applications.

- White box testing is much expensive and complex.
- It can lead to production error because it is not detailed by the developers.
- White box testing needs professional programmers who have a detailed knowledge and understanding of programming language and implementation.

Techniques Used in White Box Testing

Data Flow Testing	Data flow testing is a group of testing strategies that examines the control flow of programs in order to explore the sequence of variables according to the sequence of events.
Control Flow Testing	Control flow testing determines the execution order of statements or instructions of the program through a control structure. The control structure of a program is used to develop a test case for the program. In this technique, a particular part of a large program is selected by the tester to set the testing path. Test cases represented by the control graph of the program.
Branch Testing	Branch coverage technique is used to cover all branches of the control flow graph. It covers all the possible outcomes (true and false) of each condition of decision point at least once.
Statement Testing	Statement coverage technique is used to design white box test cases. This technique involves execution of all statements of the source code at least once. It is used to calculate the total number of executed statements in the source code, out of total statements present in the source code.
Decision Testing	This technique reports true and false outcomes of Boolean expressions. Whenever there is a possibility of two or more outcomes from the statements like do while statement, if statement and case statement (Control flow statements), it is considered as decision point because there are two outcomes either true or false.

Black box testing

Black box testing is a technique of software testing which examines the functionality of software without peering into its internal structure or coding. The primary source of black box testing is a specification of requirements that is stated by the customer.

In this method, tester selects a function and gives input value to examine its functionality, and checks whether the function is giving expected output or not. If the function produces correct output, then it is passed in testing, otherwise failed. The test team reports the result to the development team and then tests the next function. After completing testing of all functions if there are severe problems, then it is given back to the development team for correction.

Generic steps of black box testing

- The black box test is based on the specification of requirements, so it is examined in the beginning.
- In the second step, the tester creates a positive test scenario and an adverse test scenario by selecting valid and invalid input values to check that the software is processing them correctly or incorrectly.
- In the third step, the tester develops various test cases such as decision table, all pairs test, equivalent division, error estimation, cause-effect graph, etc.
- The fourth phase includes the execution of all test cases.
- In the fifth step, the tester compares the expected output against the actual output.
- In the sixth and final step, if there is any flaw in the software, then it is cured and tested again.

Test procedure

The test procedure of black box testing is a kind of process in which the tester has specific knowledge about the software's work, and it develops test cases to check the accuracy of the software's functionality.

It does not require programming knowledge of the software. All test cases are designed by considering the input and output of a particular function.A tester knows about the definite output of a particular input, but not about how the result is arising. There are various techniques used in black box testing for testing like decision table technique, boundary value analysis technique, state transition, All-pair testing, cause-effect graph technique, equivalence partitioning technique, error guessing technique, use case technique and user story technique. All these techniques have been explained in detail within the tutorial.

Test cases

Test cases are created considering the specification of the requirements. These test cases are generally created from working descriptions of the software including requirements, design parameters, and other specifications. For the testing, the test designer selects both positive test scenario by taking valid input values and adverse test scenario by taking invalid input values to determine the correct output. Test cases are mainly designed for functional testing but can also be used for non-functional testing. Test cases are designed by the testing team, there is not any involvement of the development team of software.

Techniques Used in Black Box Testing

Decision Table Technique	Decision Table Technique is a systematic approach where various input combinations and their respective system behavior are captured in a tabular form. It is appropriate for the functions that have a logical relationship between two and more than two inputs.

Boundary Value Technique	Boundary Value Technique is used to test boundary values, boundary values are those that contain the upper and lower limit of a variable. It tests, while entering boundary value whether the software is producing correct output or not.
State Transition Technique	State Transition Technique is used to capture the behavior of the software application when different input values are given to the same function. This applies to those types of applications that provide the specific number of attempts to access the application.
All-pair Testing Technique	All-pair testing Technique is used to test all the possible discrete combinations of values. This combinational method is used for testing the application that uses checkbox input, radio button input, list box, text box, etc.
Cause-Effect Technique	Cause-Effect Technique underlines the relationship between a given result and all the factors affecting the result.It is based on a collection of requirements.
Equivalence Partitioning Technique	Equivalence partitioning is a technique of software testing in which input data divided into partitions of valid and invalid values, and it is mandatory that all partitions must exhibit the same behavior.
Error Guessing Technique	Error guessing is a technique in which there is no specific method for identifying the error. It is based on the experience of the test analyst, where the tester uses the experience to guess the problematic areas of the software.
Use Case Technique	Use case Technique used to identify the test cases from the beginning to the end of the system as per the usage of the system. By using this technique, the test team creates a test scenario that can exercise the entire software based on the functionality of each function from start to end.

GreyBox Testing

Greybox testing is a software testing method to test the software application with partial knowledge of the internal working structure. It is a **combination of black box and white box testing** because it involves access to internal coding to design test cases as white box testing and testing practices are done at functionality level as black box testing.

No.	Alpha Testing	Beta Testing

1)	It is always done by developers at the software development site.	It is always performed by customers at their site.
2)	It is also performed by Independent testing team	It is not be performed by Independent testing team
3)	It is not open to the market and public.	It is open to the market and public.
4)	It is always performed in a virtual environment.	It is always performed in a real-time environment.
5)	It is used for software applications and projects.	It is used for software products.
6)	It follows the category of both white box testing and Black Box Testing.	It is only the kind of Black Box Testing.
7)	It is not known by any other name.	It is also known as field testing.

GreyBox testing commonly identifies context-specific errors that belong to web systems. For example; then test it again in real time. It concentrates on all the layers of any complex software system to increase testing coverage. It gives the ability to test both presentation layer as well as internal coding structure. It is primarily used in integration testing and penetration testing.

25) What is alpha and beta testing?

These are the key differences between alpha and beta testing:

while testing, if tester encounters any defect then he makes changes in code to resolve the defect and

Software Quality

Software quality product is defined in term of its fitness of purpose. That is, a quality product does precisely what the users want it to do. For software products, the fitness of use is generally explained in terms of satisfaction of the requirements laid down in the SRS document. Although "fitness of purpose" is a satisfactory interpretation of quality for many devices such as a car, a table fan, a grinding machine, etc.for software products, "fitness of purpose" is not a wholly satisfactory definition of quality.

Example: Consider a functionally correct software product. That is, it performs all tasks as specified in the SRS document. But, has an almost unusable user interface. Even though it may be functionally right, we cannot consider it to be a quality product.

The modern view of a quality associated with a software product several quality methods such as the following:

Portability: A software device is said to be portable, if it can be freely made to work in various operating system environments, in multiple machines, with other software products, etc.

Usability: A software product has better usability if various categories of users can easily invoke the functions of the product.

Reusability: A software product has excellent reusability if different modules of the product can quickly be reused to develop new products.

Correctness: A software product is correct if various requirements as specified in the SRS document have been correctly implemented.

Maintainability: A software product is maintainable if bugs can be easily corrected as and when they show up, new tasks can be easily added to the product, and the functionalities of the product can be easily modified, etc.

Software Quality Management System

A quality management system is the principal methods used by organizations to provide that the products they develop have the desired quality.

A quality system subsists of the following:

Managerial Structure and Individual Responsibilities: A quality system is the responsibility of the organization as a whole. However, every organization has a sever quality department to perform various quality system activities. The quality system of an arrangement should have the support of the top management. Without help for the quality system at a high level in a company, some members of staff will take the quality system seriously.

Quality System Activities: The quality system activities encompass the following:

Auditing of projects

Review of the quality system

Development of standards, methods, and guidelines, etc.

Production of documents for the top management summarizing the effectiveness of the quality system in the organization.

Evolution of Quality Management System

Quality systems have increasingly evolved over the last five decades. Before World War II, the usual function to produce quality products was to inspect the finished products to remove defective devices. Since that time, quality systems of organizations have undergone through four steps of evolution, as shown in the fig. The first product inspection task gave method to quality control (QC).

Quality control target not only on detecting the defective devices and removes them but also on determining the causes behind the defects. Thus, quality control aims at correcting the reasons for bugs and not just rejecting the products. The next breakthrough in quality methods was the development of quality assurance methods.

The primary premise of modern quality assurance is that if an organization's processes are proper and are followed rigorously, then the products are obligated to be of good quality. The new quality functions include guidance for recognizing, defining, analyzing, and improving the production process.

Total quality management (TQM) advocates that the procedure followed by an organization must be continuously improved through process measurements. TQM goes stages further than quality assurance and aims at frequently process improvement. TQM goes beyond documenting steps to optimizing them through a redesign. A term linked to TQM is Business Process Reengineering (BPR).

BPR aims at reengineering the method business is carried out in an organization. From the above conversation, it can be stated that over the years, the quality paradigm has changed from product assurance to process assurance, as shown in fig.

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

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. **Continued Inspection:** The registrar continued to monitor the organization time by time.

- An introduction to object-oriented analysis and design.
- System modeling.

- Conceptual model.
- Structural, Interaction, and Behavioral models.
- Design principles and Design patterns.

The Concept Of Object–Orientation

Object-orientation is what's referred to as a programming paradigm. It's not a language itself but a set of concepts that is supported by many languages.

If you aren't familiar with the concepts of object-orientation, you may take a look at The Story of Object-Oriented Programming.

If everything we do in these languages is object-oriented, it means, we are oriented or focused around **objects**.

Now in an object-oriented language, this one large program will instead be split apart into self contained objects, almost like having several mini-programs, each object representing a different part of the application.

And each object contains its own data and its own logic, and they communicate between themselves.

These objects aren't random. They represent the way you talk and think about the problem you are trying to solve in your real life.

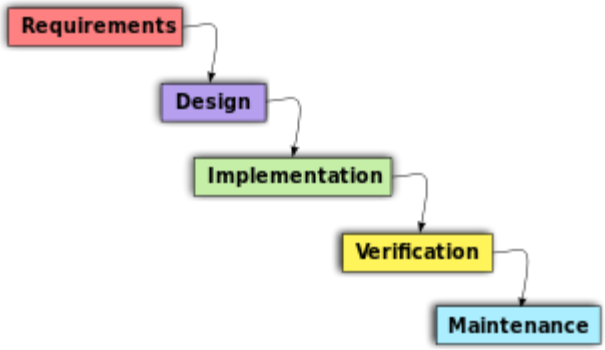
They represent things like employees, images, bank accounts, spaceships, asteroids, video segment, audio files, or whatever exists in your program.

Object–Oriented Analysis And Design (OOAD)

It's a structured method for analyzing, designing a system by applying the object-orientated concepts, and develop a set of graphical system models during the development life cycle of the software.

OOAD In The SDLC

The software life cycle is typically divided up into stages going from abstract descriptions of the problem to designs then to code and testing and finally to deployment.



The earliest stages of this process are analysis (requirements) and design.

The distinction between analysis and design is often described as “what Vs how”.

In analysis developers work with users and domain experts to define what the system is supposed to do. Implementation details are supposed to be mostly or totally ignored at this phase.

The goal of the analysis phase is to create a model of the system regardless of constraints such as appropriate technology. This is typically done via use cases and *abstract* definition of the most important objects using conceptual model.

The design phase refines the analysis model and applies the needed technology and other implementation constrains.

It focuses on describing the objects, their attributes, behavior, and interactions. The design model should have all the details required so that programmers can implement the design in code.

They’re best conducted in an iterative and incremental software methodologies. So, the activities of OOAD and the developed models aren’t done once, we will revisit and refine these steps continually.

Object-Oriented Analysis

In the object-oriented analysis, we …

1. **Elicit** *requirements:* Define what does the software need to do, and what’s the problem the software trying to solve.
2. **Specify** *requirements*: Describe the requirements, usually, using use cases (and scenarios) or user stories.
3. **Conceptual** *model*: Identify the important objects, refine them, and define their relationships and behavior and draw them in a simple diagram.

We're not going to cover the first two activities, just the last one. These are already explained in detail in Requirements Engineering.

Object–Oriented Design

The analysis phase identifies the objects, their relationship, and behavior using the conceptual model (an **abstract** definition for the objects).

While in design phase, we describe these objects (by creating class diagram from conceptual diagram — usually mapping conceptual model to class diagram), their attributes, behavior, and interactions.

In addition to applying the software design principles and patterns which will be covered in later tutorials.

The input for object-oriented design is provided by the output of object-oriented analysis. But, analysis and design may occur in parallel, and the results of one activity can be used by the other.

In the object-oriented design, we …

1. **Describe the classes** and their relationships using class diagram.
2. **Describe the interaction** between the objects using sequence diagram.
3. **Apply** software design principles and design patterns.

A class diagram gives a visual representation of the classes you need. And here is where you get to be really specific about object-oriented principles like inheritance and polymorphism.

Describing the interactions between those objects lets you better understand the responsibilities of the different objects, the behaviors they need to have.

— Other diagrams

There are many other diagrams we can use to model the system from different perspectives; interactions between objects, structure of the system, or the behavior of the system and how it responds to events.

It's always about selecting the right diagram for the right need. You should realize which diagrams will be useful when thinking about or discussing a situation that isn't clear.

System modeling and the different models we can use will be discussed next.

System Modeling

System modeling is the process of developing models of the system, with each model representing a different perspectives of that system.

The most important aspect about a system model is that it leaves out detail; It's an abstract representation of the system.

The models are usually based on graphical notation, which is almost always based on the notations in the Unified Modeling Language (UML). Other models of the system like mathematical model; a detailed system description.

Models are used during the analysis process to help to elicit the requirements, during the design process to describe the system to engineers, and after implementation to document the system structure and operation.

Different Perspectives

We may develop a model to represent the system from different perspectives.

1. **External**, where you model the context or the environment of the system.
2. **Interaction**, where you model the interaction between components of a system, or between a system and other systems.
3. **Structural**, where you model the organization of the system, or the structure of the data being processed by the system.
4. **Behavioral**, where you model the dynamic behavior of the system and how it respond to events.

Unified Modeling Language (UML)

The unified modeling language become the standard modeling language for object-oriented modeling. It has many diagrams, however, the most diagrams that are commonly used are:

- **Use case diagram**: It shows the interaction between a system and it's environment (users or systems) within a particular situation.
- **Class diagram**: It shows the different objects, their relationship, their behaviors, and attributes.
- **Sequence diagram**: It shows the interactions between the different objects in the system, and between actors and the objects in a system.
- **State machine diagram**: It shows how the system respond to external and internal events.
- **Activity diagram**: It shows the flow of the data between the processes in the system.

You can do diagramming work on paper or on a whiteboard, at least in the initial stages of a project. But there are some diagramming tools that will help you to draw these UML diagrams.

Organizational project-enabling processes

Detailed here are life cycle model management, infrastructure management, portfolio management, human resource management, quality management, and knowledge management processes. These processes help a business or organization enable, control, and support the system life cycle and related projects. Life cycle mode management helps ensure acquisition and supply efforts are supported, while infrastructure and portfolio management supports business and project-specific initiatives during the entire system life cycle. The rest ensure the necessary resources and quality controls are in place to support the business' project and system endeavors. If an organization does not have an appropriate set of organizational processes, a project executed by the organization may apply those processes directly to the project instead.[1]

Technical management processes

ISO/IEC/IEEE 12207:2017 places eight different processes .

- Task planning
- Project assessment and control
- Decision management
- Risk management
- Configuration management
- Information management
- Measurement
- Quality assurance

These processes deal with planning, assessment, and control of software and other projects during the life cycle, ensuring quality along the way.

Technical processes

The technical processes of ISO/IEC/IEEE 12207:2017 encompass 14 different processes, some of which came from the old software-specific processes that were phased out from the 2008 version.

The full list includes.

- Business or mission analysis
- Stakeholder needs and requirements definition
- Systems/Software requirements definition
- Architecture definition
- Design definition
- System analysis
- Implementation
- Integration
- Verification
- Transition
- Validation
- Operation
- Maintenance
- Disposal

These processes involve technical activities and personnel (information technology, troubleshooters, software specialists, etc.) during pre-, post- and during operation. The analysis and definition processes early on set the stage for how software and projects are implemented. Additional processes of integration, verification, transition, and validation help ensure quality and readiness. The operation and maintenance phases occur simultaneously, with the operation phase consisting of activities like assisting users in working with the implemented software product, and the maintenance phase consisting of maintenance tasks to keep the product up and running. The disposal process describes how the system/project will be retired and cleaned up, if necessary.[1]