Software Engineering

BCA II Year

THE WATERFALL MODEL

The Waterfall Model was first Process Model to be introduced. It is also referred to as a linear-sequential life cycle model. It is very simple to understand and use. In a waterfall model, each phase must be completed before the next phase can begin and there is no overlapping in the phases.

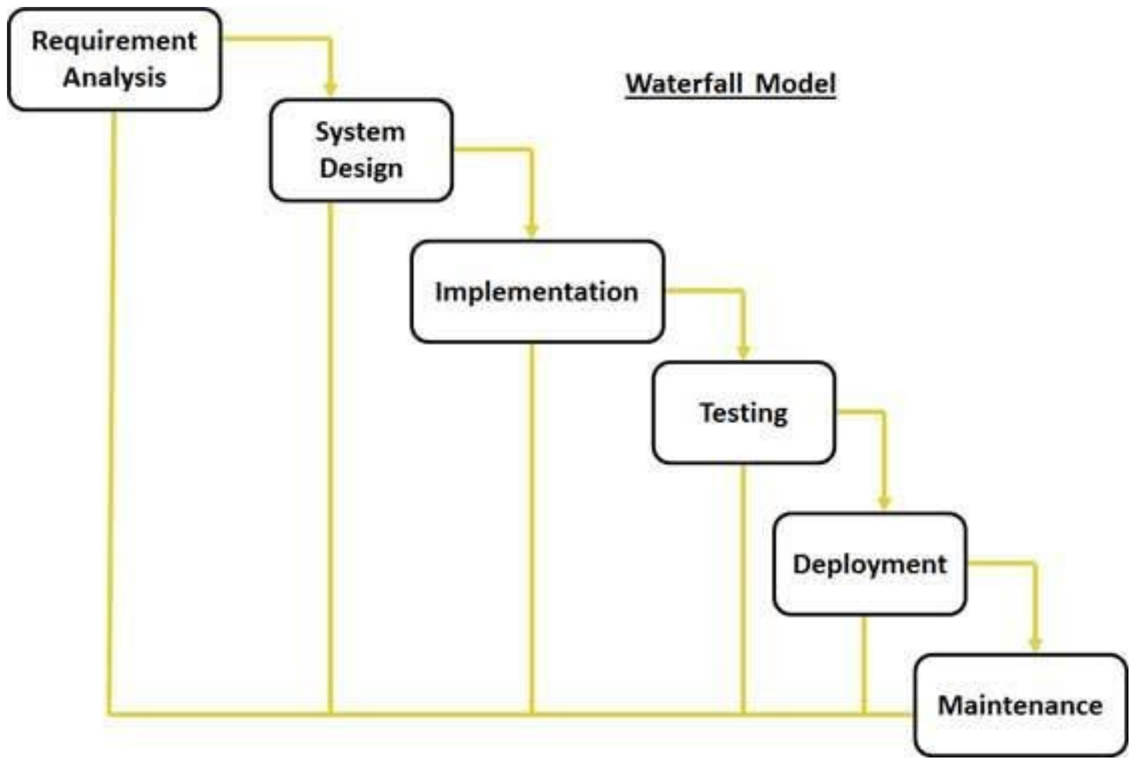
Waterfall model is the earliest SDLC approach that was used for software development.

The waterfall Model illustrates the software development process in a linear sequential flow; hence it is also referred to as a linear-sequential life cycle model. This means that any phase in the development process begins only if the previous phase is complete. In waterfall model phases do not overlap.

Waterfall Model design

Waterfall approach was first SDLC Model to be used widely in Software Engineering to ensure success of the project. In "The Waterfall" approach, the whole process of software development is divided into separate phases. In Waterfall model, typically, the outcome of one phase acts as the input for the next phase sequentially.

Following is a diagrammatic representation of different phases of waterfall model.



The sequential phases in Waterfall model are:

- **Requirement Gathering and analysis:** All possible requirements of the system to be developed are captured in this phase and documented in a requirement specification doc.
- **System Design:** The requirement specifications from first phase are studied in this phase and system design is prepared. System Design helps in specifying hardware and system requirements and also helps in defining overall system architecture.
- **Implementation:** With inputs from system design, the system is first developed in small programs called units, which are integrated in the next phase. Each unit is developed and tested for its functionality which is referred to as Unit Testing.
- **Integration and Testing:** All the units developed in the implementation phase are integrated into a system after testing of each unit. Post integration the entire system is tested for any faults and failures.
- **Deployment of system:** Once the functional and non-functional testing is done, the product is deployed in the customer environment or released into the market.

- **Maintenance:** There are some issues which come up in the client environment. To fix those issues patches are released. Also to enhance the product some better versions are released. Maintenance is done to deliver these changes in the customer environment.

All these phases are cascaded to each other in which progress is seen as flowing steadily downwards (like a waterfall) through the phases. The next phase is started only after the defined set of goals are achieved for previous phase and it is signed off, so the name "Waterfall Model". In this model phases do not overlap.

Waterfall Model Application

Every software developed is different and requires a suitable SDLC approach to be followed based on the internal and external factors. Some situations where the use of Waterfall model is most appropriate are:

- Requirements are very well documented, clear and fixed.
- Product definition is stable.
- Technology is understood and is not dynamic.
- There are no ambiguous requirements.
- Ample resources with required expertise are available to support the product.
- The project is short.

Waterfall Model Pros & Cons

Advantage

The advantage of waterfall development is that it allows for departmentalization and control. A schedule can be set with deadlines for each stage of development and a product can proceed through the development process model phases one by one.

Development moves from concept, through design, implementation, testing, installation, troubleshooting, and ends up at operation and maintenance. Each phase of development proceeds in strict order.

Disadvantage

The disadvantage of waterfall development is that it does not allow for much reflection or revision. Once an application is in the testing stage, it is very difficult to go back and change something that was not well-documented or thought upon in the concept stage.

The following table lists out the pros and cons of Waterfall model:

Pros	**Cons**
• Simple and easy to understand and use • Easy to manage due to the rigidity of the model . each phase has specific deliverables and a review process. • Phases are processed and completed one at a time. • Works well for smaller projects where requirements are very well understood. • Clearly defined stages. • Well understood milestones. • Easy to arrange tasks. • Process and results are well documented.	• No working software is produced until late during the life cycle. • High amounts of risk and uncertainty. • Not a good model for complex and object-oriented projects. • Poor model for long and ongoing projects. • Not suitable for the projects where requirements are at a moderate to high risk of changing. So risk and uncertainty is high with this process model. • It is difficult to measure progress within stages. • Cannot accommodate changing requirements. • Integration is done as a "big-bang. at the very end, which doesn't allow identifying any technological or business bottleneck or challenges early.

Limitation of waterfall model

1. It assumes that the requirements of a system can be frozen before the design begins. This is possible for system designed to automate an existing manual system. But for new systems, determining the requirements is difficult as the user does not even know the requirements.
2. It follows the “Big Bang” approach the entire software is delivered in one shot at the end. This entails heavy risks as the user does not know until the very end what they are getting. Furthermore, if the project runs out of money in the middle, there will be no software.
3. It is a document driven process that requires formal documents at the end of each phase.

Despite these limitations the waterfall model has been the most widely used process model.

It is well suited types of projects where the requirements are well understood

ITERATIVE MODEL

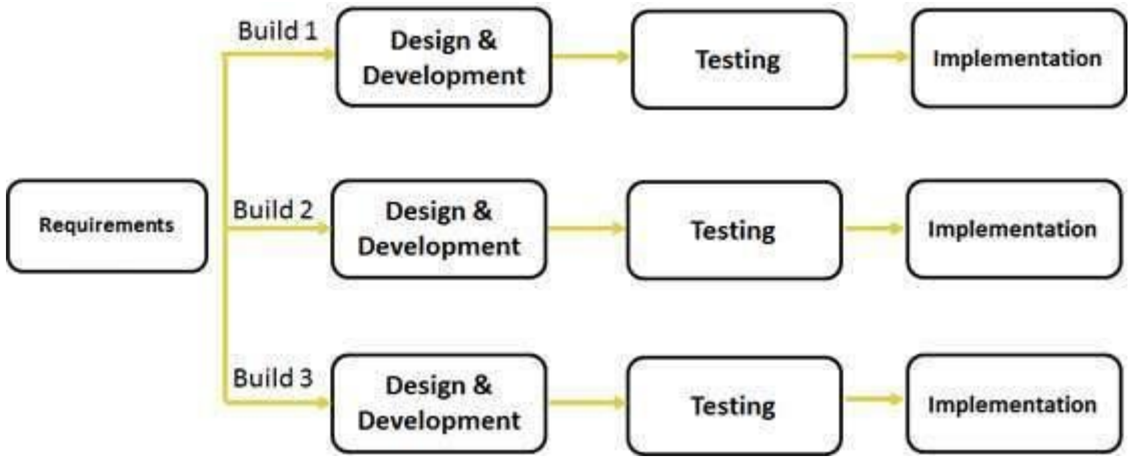
In Iterative model, iterative process starts with a simple implementation of a small set of the software requirements and iteratively enhances the evolving versions until the complete system is implemented and ready to be deployed.

An iterative life cycle model does not attempt to start with a full specification of requirements. Instead, development begins by specifying and implementing just part of the software, which is then reviewed in order to identify further requirements. This process is then repeated, producing a new version of the software at the end of each iteration of the model.

Iterative Model design

Iterative process starts with a simple implementation of a subset of the software requirements and iteratively enhances the evolving versions until the full system is implemented. At each iteration, design modifications are made and new functional capabilities are added. The basic idea behind this method is to develop a system through repeated cycles (iterative) and in smaller portions at a time (incremental).

Following is the pictorial representation of Iterative and Incremental model:



Iterative and Incremental development is a combination of both iterative design or iterative method and incremental build model for development. "During software development, more than one iteration of the software development cycle may be in progress at the same time." and "This process may be described as an "evolutionary acquisition" or "incremental build" approach."

In incremental model the whole requirement is divided into various builds. During each iteration, the development module goes through the requirements, design, implementation and testing phases. Each subsequent release of the module adds function to the previous release. The process continues till the complete system is ready as per the requirement.

The key to successful use of an iterative software development lifecycle is rigorous validation of requirements, and verification & testing of each version of the software against those requirements within each cycle of the model. As the software evolves through successive cycles, tests have to be repeated and extended to verify each version of the software.

Iterative Model Application

Like other SDLC models, Iterative and incremental development has some specific applications in the software industry. This model is most often used in the following scenarios:

- Requirements of the complete system are clearly defined and understood.
- Major requirements must be defined; however, some functionalities or requested enhancements may evolve with time.
- There is a time to the market constraint.

- A new technology is being used and is being learnt by the development team while working on the project.
- Resources with needed skill set are not available and are planned to be used on contract basis for specific iterations.
- There are some high risk features and goals which may change in the future.

Iterative Model Pros and Cons

The advantage of this model is that there is a working model of the system at a very early stage of development which makes it easier to find functional or design flaws. Finding issues at an early stage of development enables to take corrective measures in a limited budget.

The disadvantage with this SDLC model is that it is applicable only to large and bulky software development projects. This is because it is hard to break a small software system into further small serviceable increments/modules.

The following table lists out the pros and cons of Iterative and Incremental SDLC Model:

Pros	Cons
• Some working functionality can be developed quickly and early in the life cycle. • Results are obtained early and periodically. • Parallel development can be planned. • Progress can be measured. • Less costly to change the scope/requirements. • Testing and debugging during smaller iteration is easy.	• More resources may be required. • Although cost of change is lesser but it is not very suitable for changing requirements. • More management attention is required. • System architecture or design issues may arise because not all requirements are gathered in the beginning of the entire life cycle.

• Risks are identified and resolved during iteration; and each iteration is an easily managed milestone. • Easier to manage risk - High risk part is done first. • With every increment operational product is delivered. • Issues, challenges & risks identified from each increment can be utilized/applied to the next increment. • Risk analysis is better. • It supports changing requirements. • Initial Operating time is less. • Better suited for large and mission-critical projects. • During life cycle software is produced early which facilitates customer evaluation and feedback.	• Defining increments may require definition of the complete system. • Not suitable for smaller projects. • Management complexity is more. • End of project may not be known which is a risk. • Highly skilled resources are required for risk analysis. • Project. progress is highly dependent upon the risk analysis phase.

Software Design

Software Design:- The process of design involves "planning out in the mind and making a drawing, pattern or sketch off"

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

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

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

Software Design Levels

Software design yields three levels of results:

- **Architectural Design -** The architectural design is the highest abstract version of the system. It identifies the software as a system with many components interacting with each other. At this level, the designers get the idea of proposed solution domain.

- **High-level Design-** The high-level design breaks the ‘single entity-multiple component’ concept of architectural design into less-abstracted view of sub-systems and modules and depicts their interaction with each other. High-level design focuses on how the system along with all of its components can be implemented in forms of modules. It recognizes modular structure of each sub-system and their relation and interaction among each other. This is also called the system design or top level design.
- **Detailed Design-** Detailed design deals with the implementation part of what is seen as a system and its sub-systems in the previous two designs. It is more detailed towards modules and their implementations. It defines logical structure of each module and their interfaces to communicate with other modules.

  Detailed design essentially expands the system design to contain a more detailed description of the processing ,logic and data structure so that the design sufficiently complete for coding.

Design and Quality

  - The design must implement all of the explicit requirements contained in analysis model, and it must accommodate all of the implicit requirements desired by the customer.
  - The design must be a readable, understandable guide for those who generate code and for those who test and subsequently support the software.
  - The design should provide a complete picture of the software, addressing the data, functional, and behavioral domains from an implementation perspective.

Design Principles

The design process is a sequences of steps that helps the designer to describe all aspects of the software to be built.

The critical factors necessary for good design are creative skill, Past Experience, a sense of what makes good software and on overall commitment to quality so the software design is both a process and a model.

Design Principles

1. The design process should not suffer from tunnel vision.
2. The design should be traceable to the analysis model.
3. The design should minimize the intellectual distance between the SW and the problem as it exists in the real world
4. The design should exhibit uniformity & integration.
5. The design should be structured to accommodate change clear distinction between design and coding.

Design Concept

Abstraction:

Concentrate on the essential features and ignore details that are not relevant.

Procedural Abstraction:-A named sequence of instructions that has a specific & limited function.

Data Abstraction :A named collection of data that describes a data object.

Functional Abstraction: In this, a module is specified by the function it performs .for ex:- A module to compute the log of a value can be abstractly by the log. Similarly a module to sort an input array can be represented by the specification of sorting.

Control abstraction: implies a program control mechanisms without specify internal details.

Modularity

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

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

Advantage of modularization:

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

Concurrency

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

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

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

Top Down and Bottom Up Strategies

A top down approach starts with identifying the major components of the system .decomposing them into their lower level components and iterating until the desired level of detail is achieved.

Top down design methods often result in some form of stepwise refinement .Starting from an abstract design in each step the design is refined to a more concrete level. Until we reach a level no more refinement is needed and the design can be implemented directly .

The top down approach has been promulgate by many researches and has been found to be extremely useful for design. Most design methodologies are based on the top down approach.

A top down approach is suitable only if the specification of the system are clearly known and the system development is from scratch.

Bottom Up:- A bottom up design approach starts with designing the most basic or primitive components and proceeds to higher level components that use these lower level components .

Bottom up methods work with layers of abstraction starting from the very bottom operations that provide a layer of abstraction are implemented.

The operation of this layer are then used to implement more powerful operations and a still higher layer of abstraction until the stage is reached where the operations supported by the layer are those desired by the system.

If a system is to be built from an existing system a bottom up approach is more suitable as it starts from some existing components.

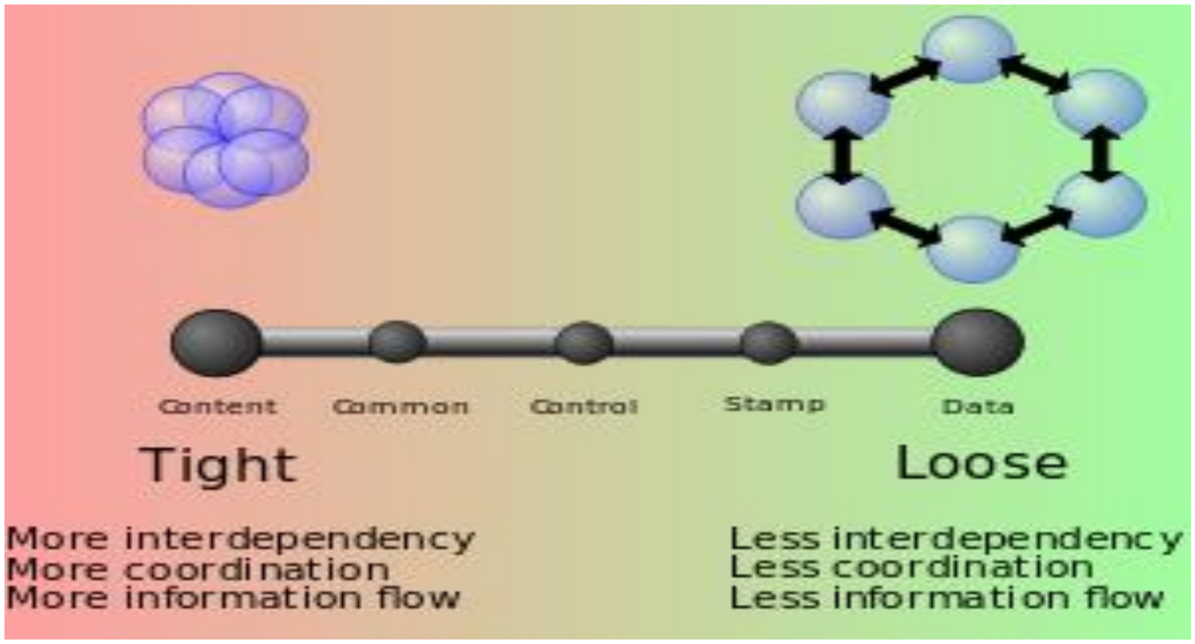
Module Level Concept

Coupling and Cohesion

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

Cohesion

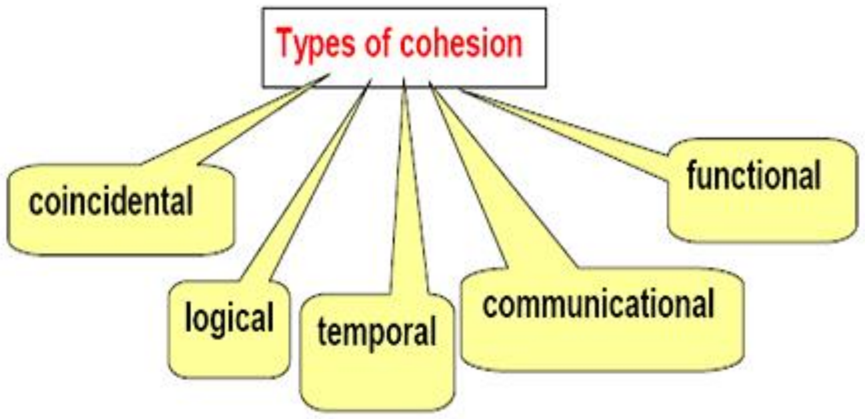
To maximize the relationship between elements of same modules. Cohesion **is the concept come in sight.** The cohesion **of a module represents how tightly internal element of a module are bound with one another. Greater the cohesion of each module, lower will be the coupling between the modules.**



Types of cohesion:-

1. Coincidental cohesion
2. Logical cohesion
3. Temporal cohesion
4. Procedural cohesion
5. Communicational cohesion

6. Sequential cohesion
7. Functional cohesion



What is cohesion? Explain different types of cohesion?

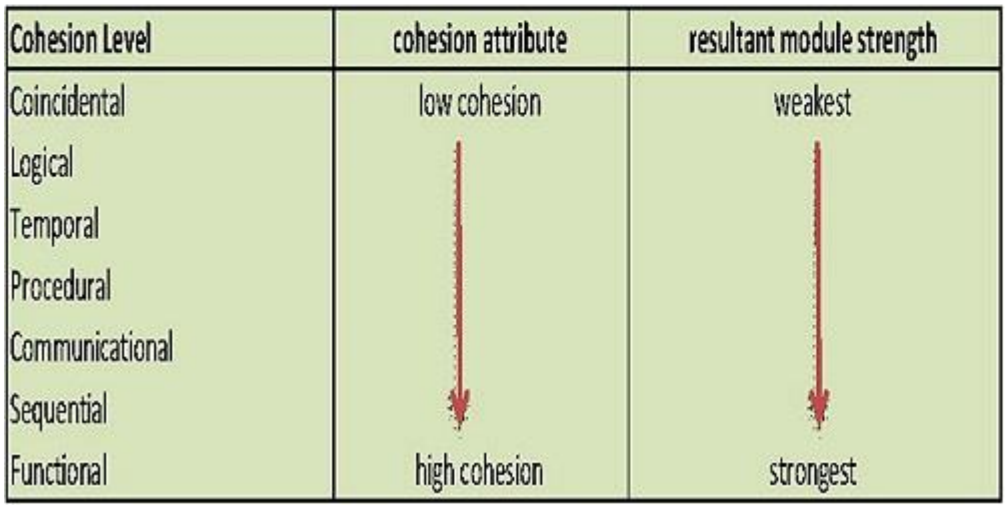
Coincidental cohesion:-

This occurs when there is no relationship among elements of a module. **-** It is unplanned and random cohesion, which might be the result of breaking the program into smaller modules for the sake of modularization. Because it is unplanned, it may serve confusion to the programmers and is generally not-accepted.

Logical cohesion:-

A module having logical cohesion if there are some logical relationship between elements of a modules i.e. elements of a module performs the operation.

Example:- data handling, data input, data output etc. A set of print function generating an output report has been arranged in a single module.



Cohesion Level	cohesion attribute	resultant module strength
Coincidental	low cohesion	weakest
Logical		
Temporal		
Procedural		
Communicational		
Sequential		
Functional	high cohesion	strongest

What is cohesion? Explain different types of cohesion?

Temporal cohesion:-

It is same as logical cohesion except that the element must be executed in same time. Set of function responsible for initialization, startup, the shutdown of the same process. It is higher than logical cohesion since all elements are executed together. This avoids the problem of passing the flag.

Procedural cohesion:-

It contains that belongs to a procedural unit in which a certain sequence of steps has to be carried out in a certain order for achieving an objective.

Example:-

- A loop or a sequence of decision state in a module may be combined to form a separate module.
- The algorithm for decoding a module.

Communicational cohesion:-

A module is said to have Communicational cohesion if all function of module refers to an update the same data structure.

For example:-

The set of the defined an array or a stack.

Sequential cohesion:-

A module is said to have sequential cohesion if element module from different parts of the sequence. When the output from one element of the sequence is input to the next element of a sequence. A sequence bounded module may contain several functions or part of different functions.

Functional cohesion:-

It is the strongest cohesion in a functional bound module, all elements of the module are related to performing a single function. By function we not mean simply mathematical function but also these modules which have single goal function like computing square root and sort array are a clear example of functionality cohesion modules.

Coupling

Coupling is a measure of the relationship (i.e., dependency) between two modules. Coupling measures the likelihood of a change or fault in one module affecting another module.The lower the coupling, the better the program.

Designing Loosely Coupled Modules

There are different types of interfaces that can be used to communicate between modules. Coupling between modules can be achieved by passing parameters, using global data areas, or by referencing the internal contents of another module.

Coupling is a measure of the quality of a design. The objective is to minimize coupling between modules, i.e., make modules as independent as possible.

In structured design, loosely coupled modules are desirable.

Loose coupling between modules is an indication the system is well partitioned. Loose coupling is achieved by:

· eliminating unnecessary relationships,

· minimizing the number of necessary relationships,

· reducing the complexity of necessary relationships.

One way to design independent modules is to consider the re-usability of the module.

There are five levels of coupling, namely –

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

CODING

Once we start working on software development, Coding or Development is the third phase of SDLC. This phase is the third step of software development life cycle and comes after requirement analyses and Designing.
In designing phase we have already taken all the major decision regarding the system, now it's time to develop the physical system. We will consider designing phase output as input for coding phase. The basic role of this phase is to convert designing into code using the programming language decided in designing phase. The well-developed code in this phase can help to reduce the efforts required in testing and maintenance. But even we make any silly mistake; it may lead us to put extra efforts in testing and maintenance.

If we see it in a business perspective, the cost for testing efforts and maintenance is much higher than coding. So it always makes sense to spend time on coding phase. Here all developers write their own code and merged with other developer's code to make sure that all modules developed by different developers interact with each other as per expectations. This is one of the longest phases in software development life cycle.

Few points which we need to take care in this phase.

- Version control application required in this phase.
- Before begin the actual coding, you should spend some time on selecting development tool, which will be suitable for your debugging, coding, modification and designing needs.
- Before actual writing code, some standard should be defined, as multiple developers going to use the same file for coding.
- During development developer should write appropriate comments so that other developers will come to know the logic behind the code.
- Last but most important point. There should be a regular review meeting need to conduct in this stage. It helps to identify the prospective defects in an early stage. Helps to improve product and coding quality.

Design Metrics

The Basic purpose of metrics is to provide quantitative data to help monitor the project.

After size metrics of a product, another metric is Complexity.

A possible use of complexity metrics at design time is to improve the design by reducing the complexity of the modules that have been found to be most complex.

This will directly improve the testability and maintainability.

If the complexity cannot be reduced because it is inherent in the problem, complexity metrics can be used to highlight the more complex modules. As complex modules are often more error-prone, this feedback can be used by project management to ensure that strict quality assurance is performed on these modules as they evolve.

Overall complexity metrics are great interest at design time and they can be used to evaluate the quality of design, improve the design, and improve quality assurance of the project.

Some of the metrics that have been proposed to quantify the complexity of design.

Cyclomatic Complexity Measures

Every program encompasses statements to execute in order to perform some task and other decision-making statements that decide, what statements need to be executed. These decision-making constructs change the flow of the program.

If we compare two programs of same size, the one with more decision-making statements will be more complex as the control of program jumps frequently.

McCabe, in 1976, proposed Cyclomatic Complexity Measure to quantify complexity of a given software. It is graph driven model that is based on decision-making constructs of program such as if-else, do-while, repeat-until, switch-case and goto statements.

Process to make flow control graph:

- Break program in smaller blocks, delimited by decision-making constructs.
- Create nodes representing each of these nodes.
- Connect nodes as follows:
  - If control can branch from block i to block j

    Draw an arc
  - From exit node to entry node

    Draw an arc.

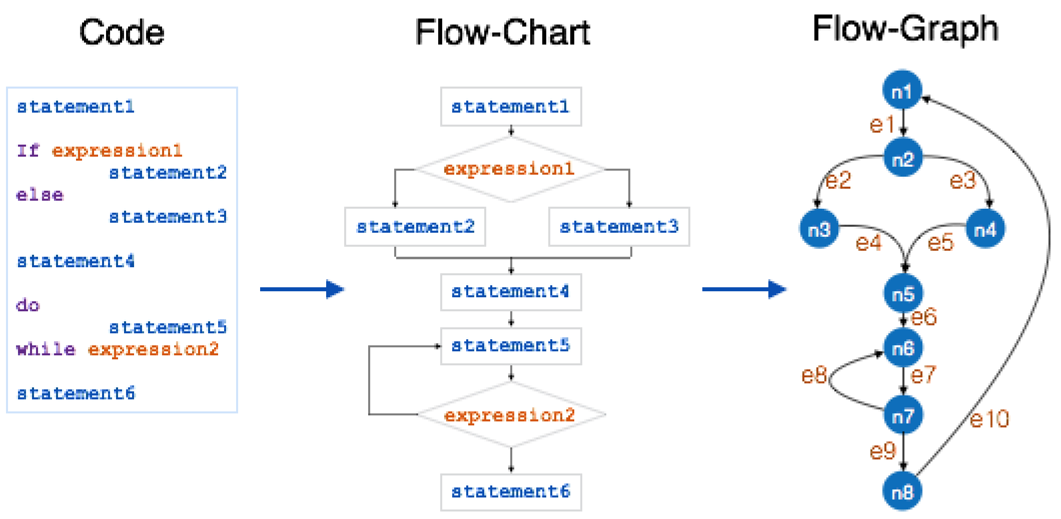
To calculate Cyclomatic complexity of a program module, we use the formula -

V(G) = e – n + 2

Where

e is total number of edges

n is total number of nodes



```
statement1

If expression1
        statement2
else
        statement3

statement4

do
        statement5
while expression2

statement6
```

The Cyclomatic complexity of the above module is

e = 10

n = 8

Cyclomatic Complexity = 10 - 8 + 2

= 4

According to P. Jorgensen, Cyclomatic Complexity of a module should not exceed 10.

Project Monitoring Plan

The software development can be seen split in two parts:

- Software Creation
- Software Project Management

A project is well-defined task, which is a collection of several operations done in order to achieve a goal (for example, software development and delivery). A Project can be characterized as:

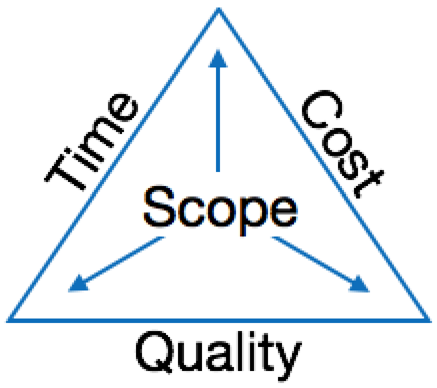
- Every project may has a unique and distinct goal.
- Project is not routine activity or day-to-day operations.
- Project comes with a start time and end time.
- Project ends when its goal is achieved hence it is a temporary phase in the lifetime of an organization.
- Project needs adequate resources in terms of time, manpower, finance, material and knowledge-bank.

Software Project

A Software Project is the complete procedure of software development from requirement gathering to testing and maintenance, carried out according to the execution methodologies, in a specified period of time to achieve intended software product.

Need of software project management

Software is said to be an intangible product. Software development is a kind of all new stream in world business and there's very little experience in building software products. Most software products are tailor made to fit client's requirements. The most important is that the underlying technology changes and advances so frequently and rapidly that experience of one product may not be applied to the other one. All such business and environmental constraints bring risk in software development hence it is essential to manage software projects efficiently.



The image above shows triple constraints for software projects. It is an essential part of software organization to deliver quality product, keeping the cost within client's budget constrain and deliver the project as per scheduled. There are several factors, both internal and external, which may impact this triple constrain triangle. Any of three factor can severely impact the other two.

Therefore, software project management is essential to incorporate user requirements along with budget and time constraints.

Software Project Manager

A software project manager is a person who undertakes the responsibility of executing the software project. Software project manager is thoroughly aware of all the phases of SDLC that the software would go through. Project manager may never directly involve in producing the end product but he controls and manages the activities involved in production.

A project manager closely monitors the development process, prepares and executes various plans, arranges necessary and adequate resources, maintains communication among all team members in order to address issues of cost, budget, resources, time, quality and customer satisfaction.

Let us see few responsibilities that a project manager shoulders -

Managing People

- Act as project leader
- Association with stakeholders
- Managing human resources
- Setting up reporting hierarchy etc.

Managing Project

- Defining and setting up project scope
- Managing project management activities
- Monitoring progress and performance
- Risk analysis at every phase
- Take necessary step to avoid or come out of problems
- Act as project spokesperson
-

Software Management Activities

Software project management comprises of a number of activities, which contains planning of project, deciding scope of software product, estimation of cost in various terms, scheduling of tasks and events, and resource management. Project management activities may include:

- **Project Planning**
- **Scope Management**
- **Project Estimation**
-

Project Planning

Software project planning is task, which is performed before the production of software actually starts. It is there for the software production but involves no concrete activity that has any direction connection with software production; rather it is a set of multiple processes, which facilitates software production. Project planning may include the following:

Scope Management

It defines the scope of project; this includes all the activities, process need to be done in order to make a deliverable software product. Scope management is essential because it creates boundaries of the project by clearly defining what would be done in the project and what would not be done. This makes project to contain limited and quantifiable tasks, which can easily be documented and in turn avoids cost and time overrun.

During Project Scope management, it is necessary to -

- Define the scope
- Decide its verification and control
- Divide the project into various smaller parts for ease of management.
- Verify the scope
- Control the scope by incorporating changes to the scope

Project Estimation

For an effective management accurate estimation of various measures is a must. With correct estimation managers can manage and control the project more efficiently and effectively.

Project estimation may involve the following:

- **Software size estimation**

  Software size may be estimated either in terms of KLOC (Kilo Line of Code) or by calculating number of function points in the software. Lines of code depend upon coding practices and Function points vary according to the user or software requirement.

- **Effort estimation**

  The managers estimate efforts in terms of personnel requirement and man-hour required to produce the software. For effort estimation software size should be known. This can either be derived by managers' experience, organization’s historical data or software size can be converted into efforts by using some standard formulae.

- **Time estimation**

  Once size and efforts are estimated, the time required to produce the software can be estimated. Efforts required is segregated into sub categories as per the requirement specifications and interdependency of various components of software. Software tasks are divided into smaller tasks, activities or events by Work Breakthrough Structure (WBS). The tasks are scheduled on day-to-day basis or in calendar months.

  The sum of time required to complete all tasks in hours or days is the total time invested to complete the project.

- **Cost estimation**

  This might be considered as the most difficult of all because it depends on more elements than any of the previous ones. For estimating project cost, it is required to consider -

  - Size of software
  - Software quality
  - Hardware
  - Additional software or tools, licenses etc.
  - Skilled personnel with task-specific skills
  - Travel involved
  - Communication
  - Training and support
  -

Project Estimation Techniques

We discussed various parameters involving project estimation such as size, effort, time and cost.

Project manager can estimate the listed factors using two broadly recognized techniques –

Decomposition Technique

This technique assumes the software as a product of various compositions.

There are two main models -

- **Line of Code** Estimation is done on behalf of number of line of codes in the software product.
- **Function Points** Estimation is done on behalf of number of function points in the software product.

Empirical Estimation Technique

This technique uses empirically derived formulae to make estimation.These formulae are based on LOC or FPs.

- **COCOMO**

  COCOMO stands for COnstructiveCOstMOdel, developed by Barry W. Boehm. It divides the software product into three categories of software: organic, semi-detached and embedded.

Project Scheduling

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

For scheduling a project, it is necessary to -

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

-

Resource management

All elements used to develop a software product may be assumed as resource for that project. This may include human resource, productive tools and software libraries.

The resources are available in limited quantity and stay in the organization as a pool of assets. The shortage of resources hampers the development of project and it can lag behind the schedule. Allocating extra resources increases development cost in the end. It is therefore necessary to estimate and allocate adequate resources for the project.

Resource management includes -

- Defining proper organization project by creating a project team and allocating responsibilities to each team member
- Determining resources required at a particular stage and their availability
- Manage Resources by generating resource request when they are required and de-allocating them when they are no more needed.
-

Project Risk Management

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

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

Risk Management Process

There are following activities involved in risk management process:

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

Project Execution & Monitoring

In this phase, the tasks described in project plans are executed according to their schedules.

Execution needs monitoring in order to check whether everything is going according to the plan. Monitoring is observing to check the probability of risk and taking measures to address the risk or report the status of various tasks.

These measures include -

- **Activity Monitoring -** All activities scheduled within some task can be monitored on day-to-day basis. When all activities in a task are completed, it is considered as complete.
- **Status Reports -** The reports contain status of activities and tasks completed within a given time frame, generally a week. Status can be marked as finished, pending or work-in-progress etc.
- **Milestones Checklist -** Every project is divided into multiple phases where major tasks are performed (milestones) based on the phases of SDLC. This milestone checklist is prepared once every few weeks and reports the status of milestones.
-

Project Communication Management

Effective communication plays vital role in the success of a project. It bridges gaps between client and the organization, among the team members as well as other stake holders in the project such as hardware suppliers.

Communication can be oral or written. Communication management process may have the following steps:

- **Planning** - This step includes the identifications of all the stakeholders in the project and the mode of communication among them. It also considers if any additional communication facilities are required.
- **Sharing** - After determining various aspects of planning, manager focuses on sharing correct information with the correct person on correct time. This keeps every one involved the project up to date with project progress and its status.
- **Feedback** - Project managers use various measures and feedback mechanism and create status and performance reports. This mechanism ensures that input from various stakeholders is coming to the project manager as their feedback.
- **Closure** - At the end of each major event, end of a phase of SDLC or end of the project itself, administrative closure is formally announced to update every stakeholder by sending email, by distributing a hardcopy of document or by other mean of effective communication.

After closure, the team moves to next phase or project.

SOFTWARE CONFIGURATION MANAGEMENT PLAN

SCM:- The primary focus of the software configuration Management (SCM) is to identify and control major software changes , ensure that changes is being properly implemented, and report changes to any other personnel or clients who may have an interest.

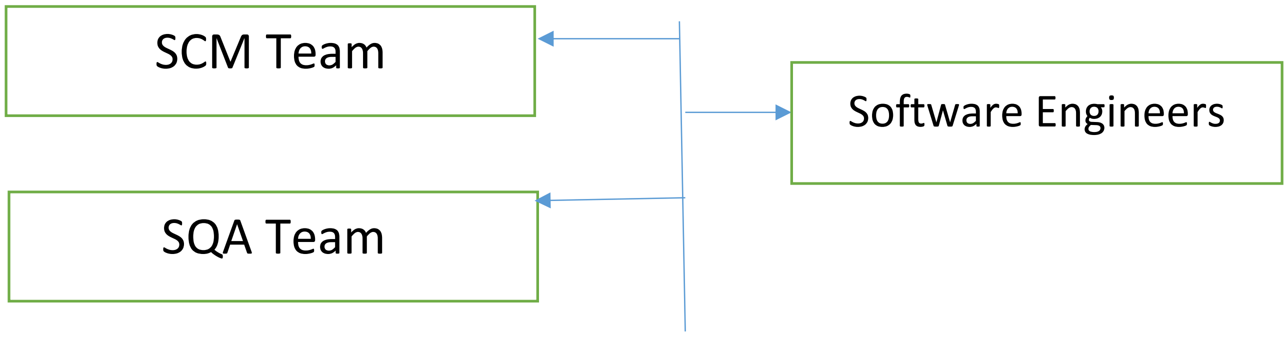
The objective of SCM is to limit the impact changes may have on the entire system. This will help to eliminate unnecessary changes, and to monitor and control any necessary changes .This allows software development to continue,

despite large and/or insignificant changes without significant backtracking, lessening development time and resulting in a higher quality product.

The SCM team will oversee these activities ,and any changes to existing code or architectural design must pass their inspection before they are carried out.

SCM Organizational Rule

The SCM team will work closely with the SQA (software quality Assurance) team, cross examining many of the submitted documents and software change requests. Software Engineers will submit change requests directly to the SCM team for their inspection and approval.



The SCM leader will be appointed to oversee all SCM activities. He will receive all change requests, and will make any final decisions regarding those changes, including which software engineer will carry out approved changes.

The SCM leader also keeps a library of all submitted requests, even those that have been denied.

SCM Tasks

Identification

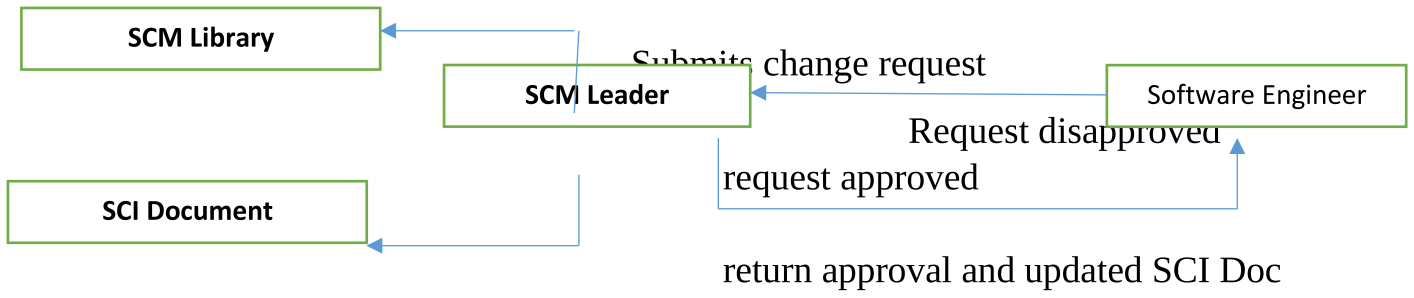
The SCM leader will analyze all current design specifications and break down the software into subsystems. All subsystems will consist of major software functions or interface components. Any submitted changes will be connected to its corresponding subsystem, which will be traced backwards though the system to determine its impact.

An SCI document will contain the breakdown of subsystems and how they are interrelated. Any approved changes will be returned to the software engineer with an change approval sheet, a listing of all possible affected subsystems, and any additional information the software engineer may need before he begins changing code.

Configuration Control

Software engineers will submit a change request to the SCM leader. The SCM leader will then analyze the request, using the SCI document, the project design document, and the current prototype of the software. He will base his decision on how severely the change will impact the entire system and, more importantly, on the corresponding subsystem.

Once his decision has been made, he must submit the change to the software engineer of his choice, as well as updating the SCI document to accommodate the change, and the SCM library to record the change request and decision.



Submitting Change Requests

Software engineers will submit a change request to the SCM leader. Because PA Software is a very small development house, all personnel work closely with each other, resulting in a more informal atmosphere.

Because of this, formal request documents are not required of the software engineer when he submits change requests. These requests may be done via email or by word of mouth (so long as the SCM leader documents the request). Formal requests will always be accepted, but PA Software does not retain a standard form. All requests, whether they are approved or not, are recorded in the SCM Request Library. Records include the requesting engineer's name, the date of the request, the subject of the request, and eventually whether or not it was approved.

Request Analysis

The SCM leader will then analyze the request, using the SCI document, the project design document, and the current prototype of the software.

The SCI document will be used to track the impact through the corresponding subsystem, and eventually the entire system. The original design document will be consulted to ensure that the requested change remains within requirement specifications and the over all spirit of the project. The SCM leader may consult the current prototype if he is unsure any change is required. This is imperative for cosmetic changes to the interface. He will base his decision on how severely the change will impact the entire system and, more importantly, on the corresponding subsystem.

Request Disapproval

If the SCM leader deems the change necessary, he will update the SCI document to reflect the change. This may include changes with the corresponding subsystem, and any impact it may have on the entire

system. Once the SCI document has been amended, it will be returned to the software engineer, and he will be notified of all possible affected subsystems for surveillance after the change has been introduced.

If these changes result in a drastic alteration of the software’s functionality or the way in which users will interact with the software, the client will be contacted for approval prior the delivery of the approval to the software engineer.

Quality Plan

To ensure that the final product is of high quality is of high quality , some quality control (QC) activities must be performed throughout the development .

A quality control task is one whose main purpose is to identify defects.

The purpose of a quality plan in a project is to specify the activities that need to be performed for identifying and removing defects, and the tools and methods that may be used for that purpose.

To ensure that the delivered software is of good quality , it is essential to ensure that all work products like the requirements specification , designed test plan are also of good quality.

For this reason a quality plan should contain quality activities throughout the project.

DEFECT INJECTION AND REMOVAL CYCLE:

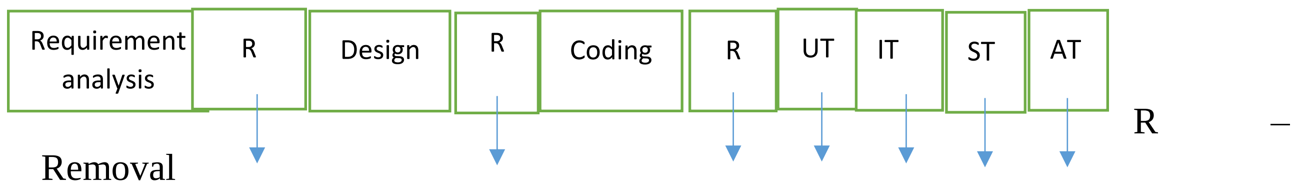
Software development is highly people oriented activity and hence it is error prone.

Defects can be injected in software at any stage during its evolution i.e during the transformation from user needs to software to satisfy those needs, defects can be injected in all the transformation activities undertaken.

These injection stages are requirement specification, the high level design, the detailed design and coding.

For high quality software, the final product should have as few defects as possible. Hence for delivery of high quality software active removal defects through quality control activities is necessary.

The QC activities for defect removal include requirement reviews, design reviews, code reviews, unit testing .integration testing, system testing and acceptance testing.



Removal

The task of quality management is to plan suitable quality control the terms verification and validation are often used.

Verification is the process of determining whether or not the products of a given phase of software development full fill the specifications established during the phases.

Validation is the process of evaluating software at the end of the software development to ensure compliance with software requirement.

The major V&V activities for software development are inspection and testing.

The quality plan identifies the different V&V tasks for the different phases and specifies how these takes contribute to the project V& V goals.

Reviews and testing are two most common QC activities .

Reviews are structured human oriented processes whereas testing is the process of executing software is an

attempt identify defects.

Testing

In a software development project, errors can be introduced at any stage during development.

Though errors are detected after each phase by techniques like inspections, reviews etc., still some errors remain undetected.

Ultimately these remaining errors will be reflected in the code . Testing is the activity where the errors remaining from all the previous phases must be detected . Hence testing performs a very critical role for ensuring quality.

Testing Fundamentals

There are some common term that used in testing :-

- **Errors -** These are actual coding mistakes made by developers. In addition, there is a difference in output of software and desired output, is considered as an error.
- **Fault -** When error exists fault occurs. A fault, also known as a bug, is a result of an error which can cause system to fail.
- **Failure -** failure is said to be the inability of the system to perform the desired task. Failure occurs when fault exists in the system.

Manual Vs Automated Testing

Testing can either be done manually or using an automated testing tool:

- **Manual -** This testing is performed without taking help of automated testing tools. The software tester prepares test cases for different sections and levels of the code, executes the tests and reports the result to the manager.

  Manual testing is time and resource consuming. The tester needs to confirm whether or not right test cases are used. Major portion of testing involves manual testing.
- **Automated** This testing is a testing procedure done with aid of automated testing tools. The limitations with manual testing can be overcome using automated test tools.

A test needs to check if a webpage can be opened in Internet Explorer. This can be easily done with manual testing. But to check if the web-server can take the load of 1 million users, it is quite impossible to test manually.

Testing Approaches

Tests can be conducted based on two approaches –

- Functionality testing
- Implementation testing

When functionality is being tested without taking the actual implementation in concern it is known as black-box testing. The other side is known as white-box testing where not only functionality is tested but the way it is implemented is also analyzed.

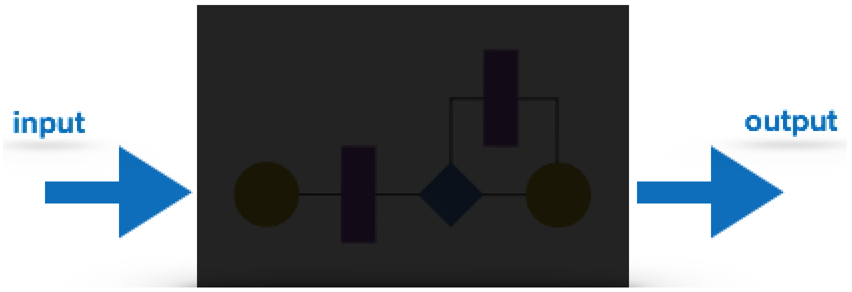
Black-box testing

Black box testing is a software testing techniques in which functionality of the software Under test (SUT) is tested without looking at the internal code structure (implementation details and knowledge of internal paths of the software).

This type of testing is based entirely on the software requirements and specifications.

In black Box testing we just focus on inputs and outputs of the software system without bothering about internal knowledge of the software program.

The tester in this case, has a set of input values and respective desired results. On providing input, if the output matches with the desired results, the program is tested 'ok', and problematic otherwise.



In this testing method, the design and structure of the code are not known to the tester, and testing engineers and end users conduct this test on the software.

For Example: An operating system like windows, a website like google, a database like oracle, Customer Application. Under Black Box Testing, you can test these applications by just focusing on the inputs and outputs without their internal code implementation.

Black Box testing –Steps

1. Initially requirements and specifications of the system are examined. Tester chooses valid inputs to check whether SUT processes them correctly. Also some invalid inputs are chosen to verify that the SUT is able to detect them.

2. Tester determines expected outputs for all those inputs.

3. Software tester constructs test cases with the selected inputs.

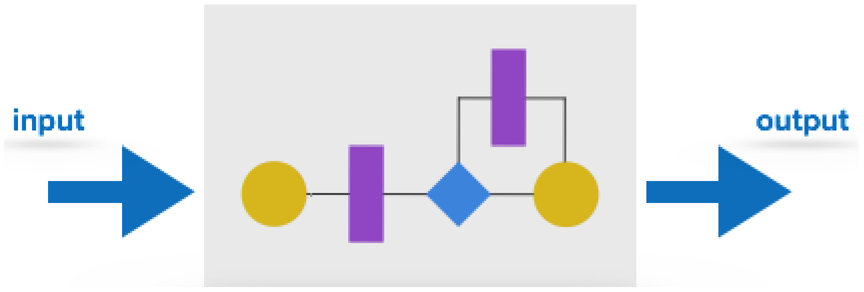
4. The test cases are executed.

5. Software tester compares the actual outputs with the expected outputs.

6. Defects if any are fixed and Re-tested.

White-box testing

It is conducted to test program and its implementation, in order to improve code efficiency or structure. It is also known as 'Structural' testing.



In this testing method, the design and structure of the code are known to the tester. Programmers of the code conduct this test on the code.

The below are some White-box testing techniques:

- **Control-flow testing -** The purpose of the control-flow testing to set up test cases which covers all statements and branch conditions. The branch conditions are tested for both being true and false, so that all statements can be covered.

- **Data-flow testing** - This testing technique emphasis to cover all the data variables included in the program. It tests where the variables were declared and defined and where they were used or changed.

Testing Process

The basic goal of the software development process is to produce software that has no errors. In an effort to detect errors soon after they are introduced, each phase ends with a verification activity such as review.

However most of the verification activities in early phases of software development are based on human evaluation and cannot detect all the errors.

This unreliability of the quality assurance activities in the early part of the development cycle places a very high responsibility on testing.

In other words as testing is the last activity before the final software is delivered. It has enormous responsibility of detecting any type of error that may be in the software.

Testing Levels

Testing itself may be defined at various levels of SDLC. The testing process runs parallel to software development. Before jumping on the next stage, a stage is tested, validated and verified.

Testing separately is done just to make sure that there are no hidden bugs or issues left in the software. Software is tested on various levels -

Unit Testing

While coding, the programmer performs some tests on that unit of program to know if it is error free. Testing is performed under white-box testing approach. Unit testing helps developers decide that individual units of the program are working as per requirement and are error free.

Integration Testing

Even if the units of software are working fine individually, there is a need to find out if the units if integrated together would also work without errors. For example, argument passing and data updation etc.

System Testing

The software is compiled as product and then it is tested as a whole. This can be accomplished using one or more of the following tests:

- **Functionality testing -** Tests all functionalities of the software against the requirement.
- **Performance testing -** This test proves how efficient the software is. It tests the effectiveness and average time taken by the software to do desired task. Performance testing is done by means of load testing and stress testing where the software is put under high user and data load under various environment conditions.
- **Security & Portability -** These tests are done when the software is meant to work on various platforms and accessed by number of persons.

Acceptance Testing

When the software is ready to hand over to the customer it has to go through last phase of testing where it is tested for user-interaction and response. This is important because even if the software matches all user requirements and if user does not like the way it appears or works, it may be rejected.

- **Alpha testing -** The team of developer themselves perform alpha testing by using the system as if it is being used in work environment. They try to find out how user would react to some action in software and how the system should respond to inputs.
- **Beta testing -** After the software is tested internally, it is handed over to the users to use it under their production environment only for testing purpose.

This is not as yet the delivered product. Developers expect that users at this stage will bring minute problems, which were skipped to attend.

Regression Testing

Whenever a software product is updated with new code, feature or functionality, it is tested thoroughly to detect if there is any negative impact of the added code. This is known as regression testing.