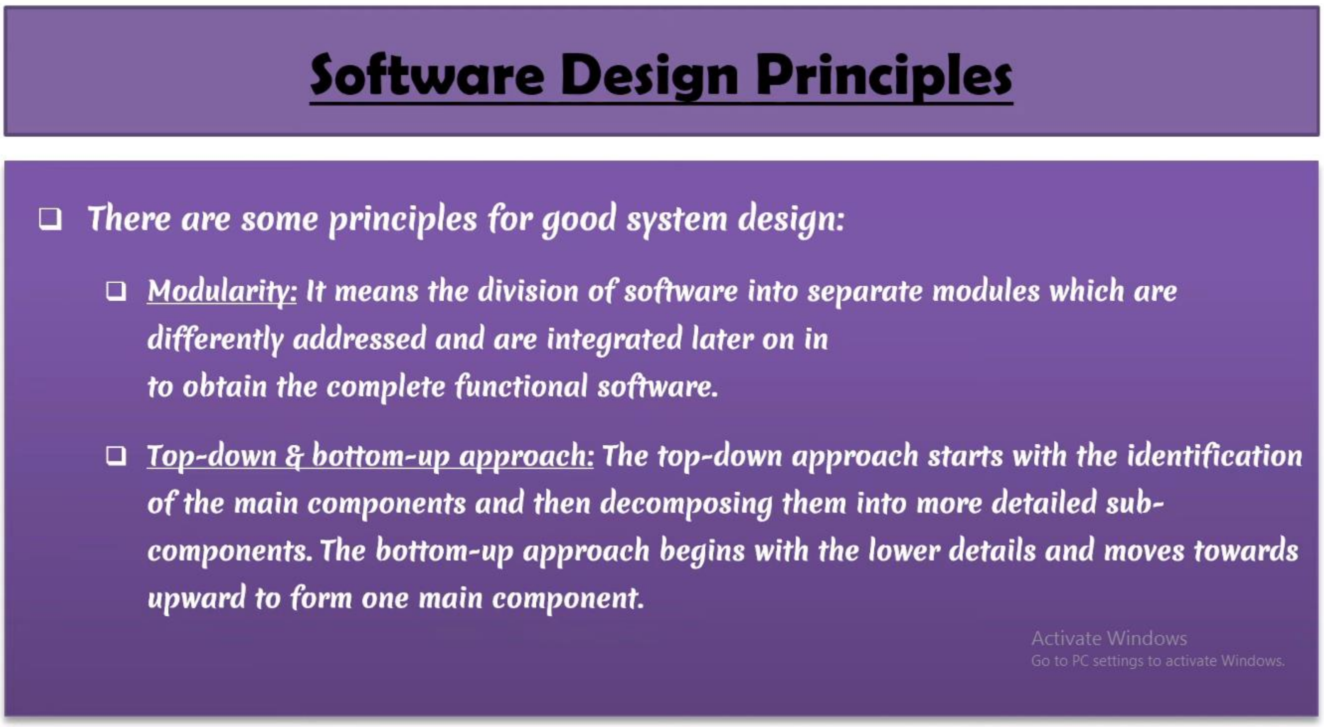
Software Design

- Software design is the process of designing the elements of a software such as the architecture, modules and components, different interfaces of those components and the data that goes into it.
- In Software designing we transform user requirements into some suitable form, which helps the programmer in software coding and implementation.
- In software design, the development is divided into several sub-activities, which coordinate with each other to achieve the main objective of software development.

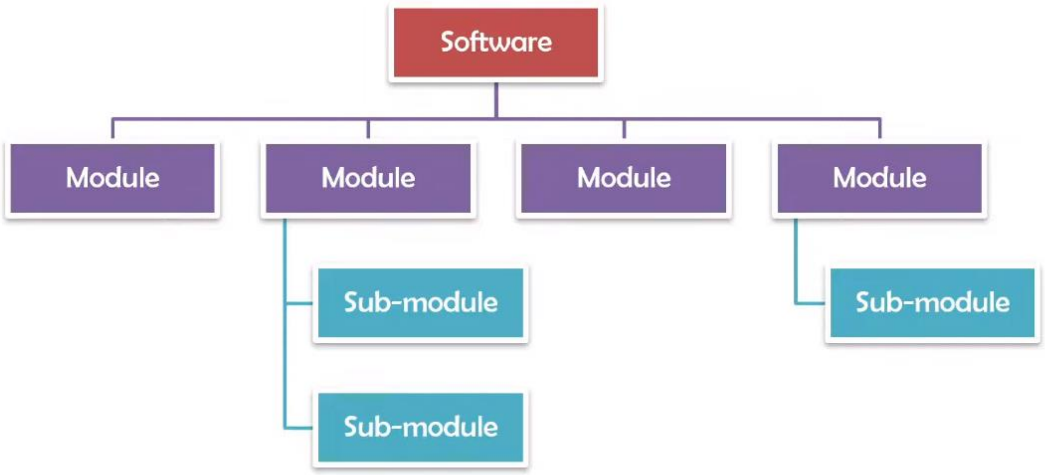
Software Design Principles

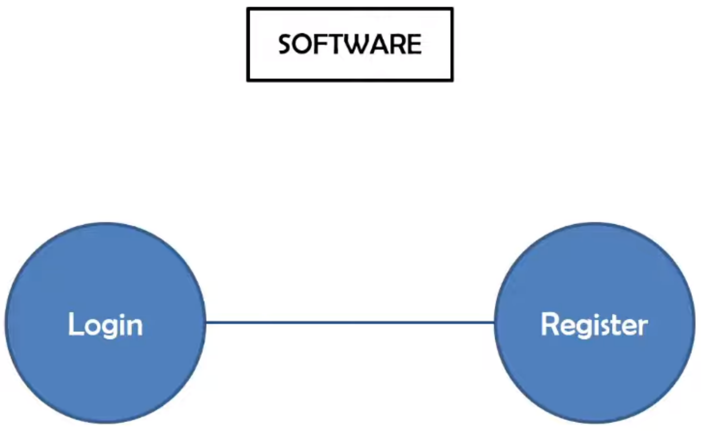
- There are some principles for good system design:
  - <u>Problem partitioning / Decomposition:</u> It refers to break down a complex system into components or smaller sub-system.
  - <u>Abstraction:</u> It means to describe the external behavior of that component without bothering with the internal details that produce the behavior.



Software Design Principles

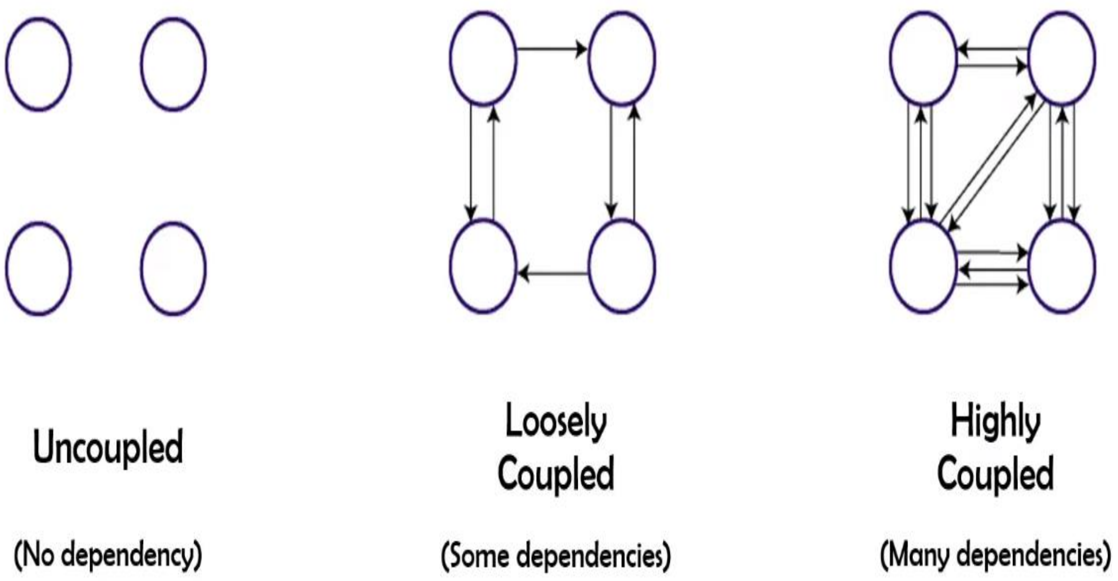
- There are some principles for good system design:
  - *__Modularity:__ It means the division of software into separate modules which are differently addressed and are integrated later on in to obtain the complete functional software.*
  - *__Top-down & bottom-up approach:__ The top-down approach starts with the identification of the main components and then decomposing them into more detailed sub-components. The bottom-up approach begins with the lower details and moves towards upward to form one main component.*

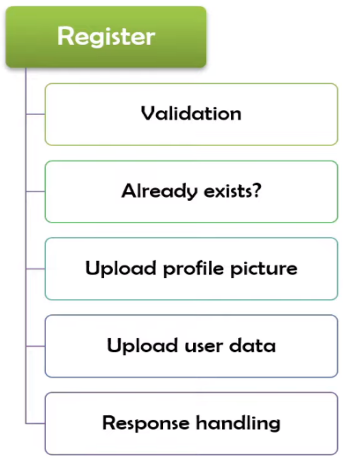


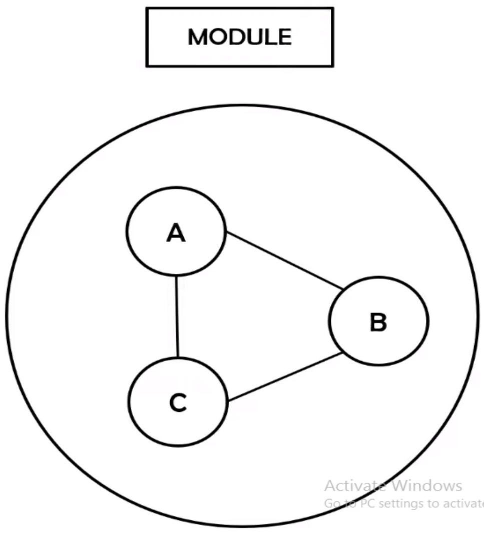


Coupling

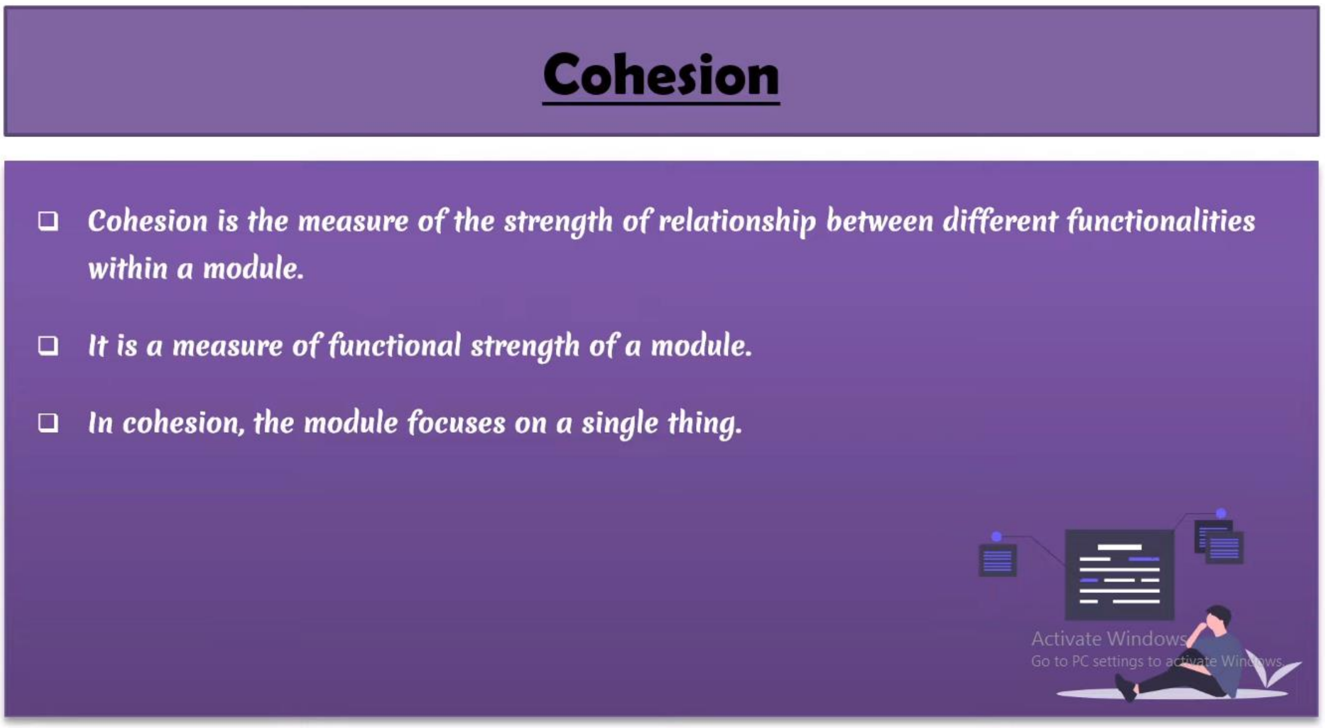
- ❑ *Modules that are tightly coupled are strongly dependent on each other.*
- ❑ *Modules that are loosely coupled are not dependent on each other.*
- ❑ *Uncoupled modules have no interdependence at all within them.*
- ❑ *While creating a software, you should aim for low coupling, i.e. , dependency among modules should be less.*


Uncoupled
(No dependency)
Loosely Coupled
(Some dependencies)
Highly Coupled
(Many dependencies)





COHESION

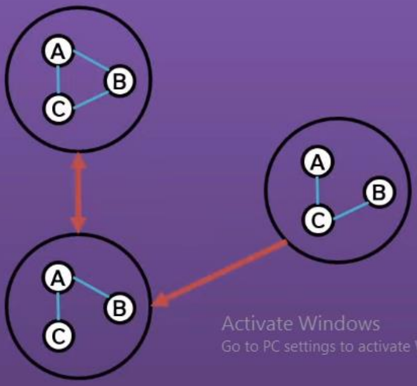


Cohesion

- ❑ *Cohesion is the measure of the strength of relationship between different functionalities within a module.*
- ❑ *It is a measure of functional strength of a module.*
- ❑ *In cohesion, the module focuses on a single thing.*

Important Point

- *A modules having low coupling and high cohesion are said to be functionally independent. It means that a cohesive module performs a single function or task and has very little interaction with other modules of the software.*

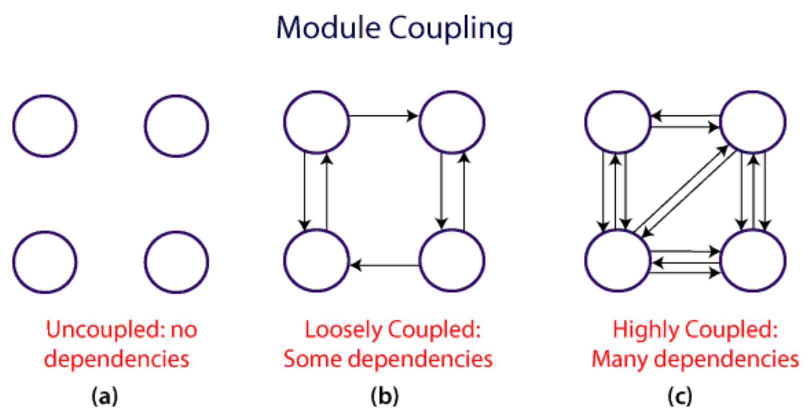


Coupling and Cohesion

Module Coupling

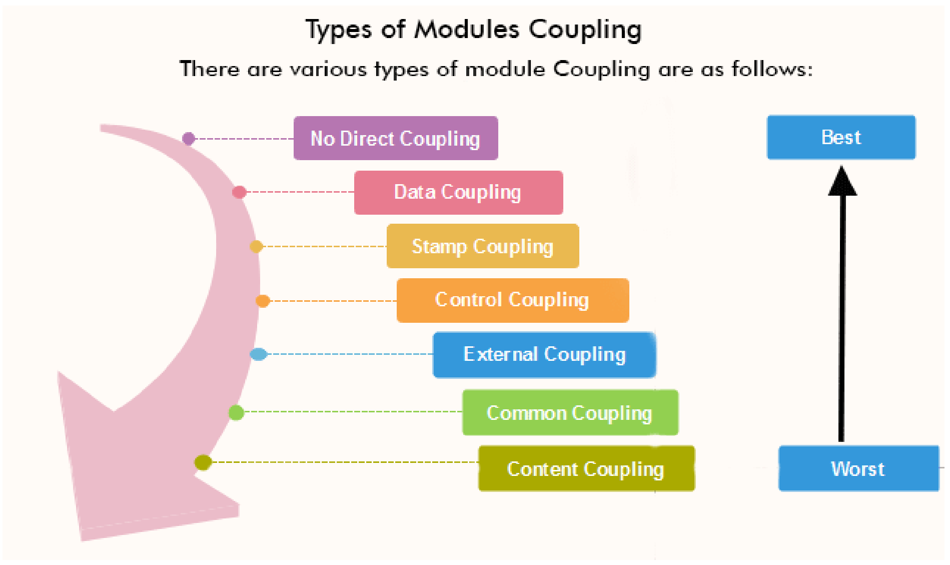
In software engineering, the coupling is the degree of interdependence between software modules. Two modules that are tightly coupled are strongly dependent on each other. However, two modules that are loosely coupled are not dependent on each other. **Uncoupled modules** have no interdependence at all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

The various types of coupling techniques are shown in fig:

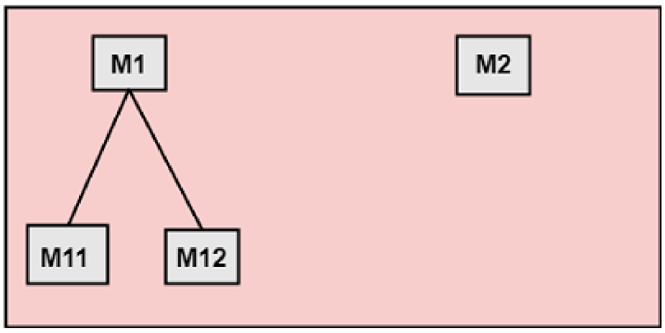

Module Coupling
Uncoupled: no dependencies
(a)
Loosely Coupled: Some dependencies
(b)
Highly Coupled: Many dependencies
(c)

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

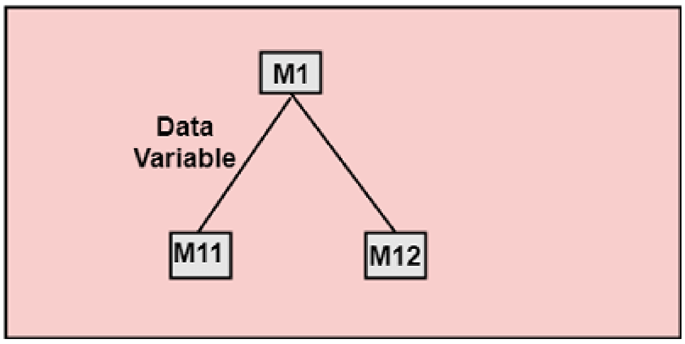


1. **No Direct Coupling:** there is a direct coupling between M1 and M2.



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.

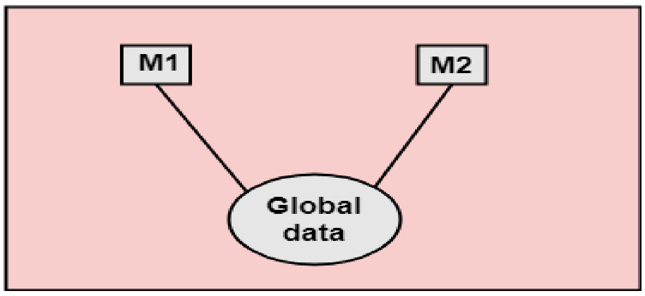


3. Stamp Coupling: Two modules are stamp coupled if they communicate using composite data items such as structure, objects, etc. When the module passes non-global data structure or entire structure to another module, they are said to be stamp coupled. For example, passing structure variable in C or object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

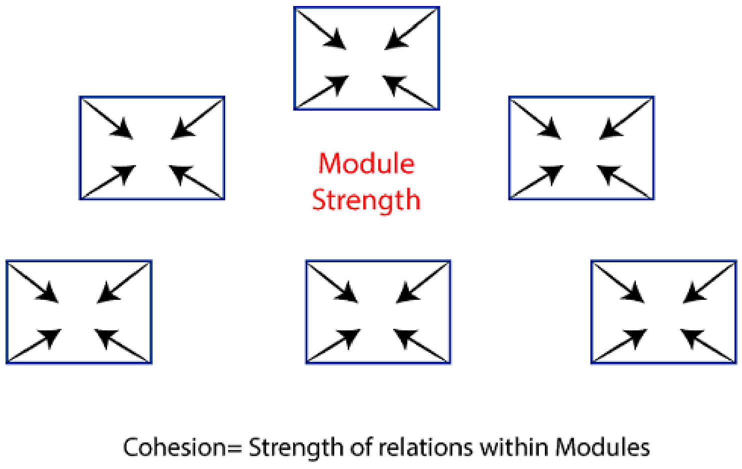


7. Content Coupling: Content Coupling exists among two modules if they share code, e.g., a branch from one module into another module.

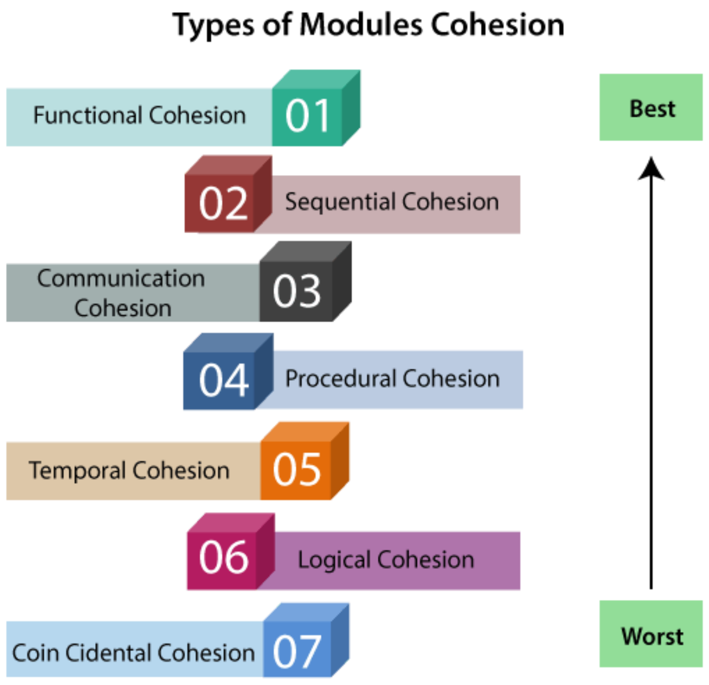
Module Cohesion

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.

Differentiate between Coupling and Cohesion

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.
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.

Introduction to Software Design Process

Software Design is the process to transform the user requirements into some suitable form, which helps the programmer in software coding and implementation. During the software design phase, the design document is produced, based on the customer requirements as documented in the SRS document. Hence the aim of this phase is to transform the SRS document into the design document.

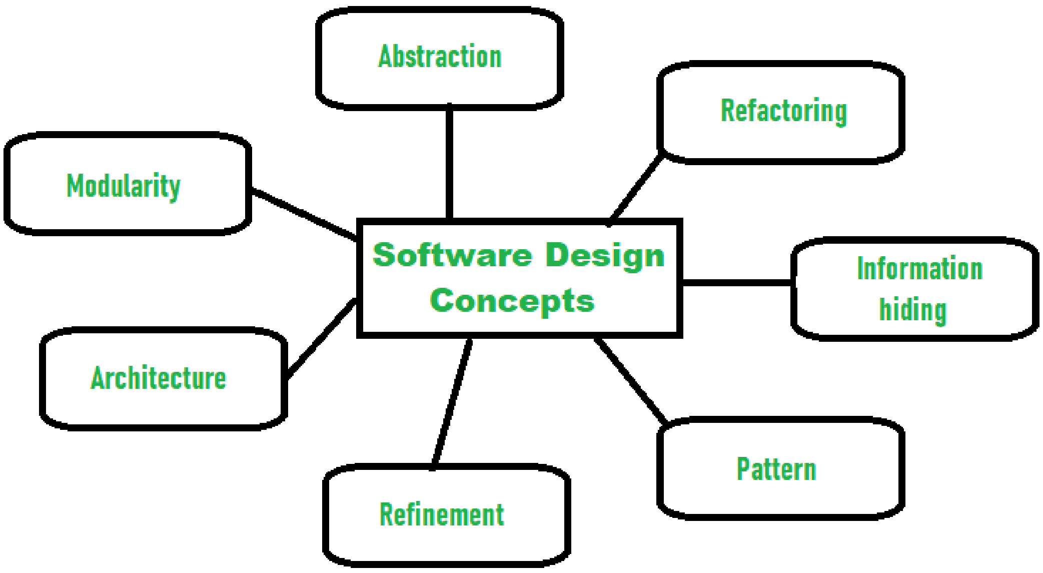
The following items are designed and documented during the design phase:

- Different modules required.
- Control relationships among modules.
- Interface among different modules.
- Data structure among the different modules.
- Algorithms required implementing among the individual modules.

Objectives of Software Design:

1. **Correctness:**
   A good design should be correct i.e. it should correctly implement all the functionalities of the system.
2. **Efficiency:**
   A good software design should address the resources, time, and cost optimization issues.
3. **Understandability:**
   A good design should be easily understandable, for which it should be modular and all the modules are arranged in layers.
4. **Completeness:**
   The design should have all the components like data structures, modules, and external interfaces, etc.
5. **Maintainability:**
   A good software design should be easily amenable to change whenever a change request is made from the customer side.

Software Design Concepts:
Concepts are defined as a principal idea or invention that comes into our mind or in thought to understand something. The **software design concept** simply means the idea or principle behind the design. It describes how you plan to solve the problem of designing software, the logic, or thinking behind how you will design software. It allows the software engineer to create the model of the system or software or product that is to be developed or built. The software design concept provides a supporting and essential structure or model for developing the right software. There are many concepts of software design and some of them are given below:



The following **points should be considered while designing Software:**

1. **Abstraction- hide irrelevant data**

   Abstraction simply means to hide the details to reduce complexity and increases efficiency or quality. Different levels of Abstraction are necessary and must be applied at each stage of the design process so that any error that is present can be removed to increase the efficiency of the software solution and to refine the software solution. The solution should be described in broad ways that cover a wide range of different things at a higher level of abstraction and a more detailed description of a solution of software should be given at the lower level of abstraction.

2. **Modularity- subdivide the system**

Modularity simply means dividing the system or project into smaller parts to reduce the complexity of the system or project. In the same way, modularity in design means subdividing a system into smaller parts so that these parts can be created independently and then use these parts in different systems to perform different functions. It is necessary to divide the software into components known as modules because nowadays there are different software available like Monolithic software that is hard to grasp for software engineers. So, modularity in design has now become a trend and is also important. If the system contains fewer components then it would mean the system is complex which requires a lot of effort (cost) but if we are able to divide the system into components then the cost would be small.

3. **Architecture- design a structure of something**

   Architecture simply means a technique to design a structure of something. Architecture in designing software is a concept that focuses on various elements and the data of the structure. These components interact with each other and use the data of the structure in architecture.

4. **Refinement- removes impurities**

   Refinement simply means to refine something to remove any impurities if present and increase the quality. The refinement concept of software design is actually a process of developing or presenting the software or system in a detailed manner that means to elaborate a system or software. Refinement is very necessary to find out any error if present and then to reduce it

   .

5. **Pattern- a repeated form**

   The pattern simply means a repeated form or design in which the same shape is repeated several times to form a pattern. The pattern in the design process means the repetition of a solution to a common recurring problem within a certain context.

6. **Information Hiding- hide the information**

   Information hiding simply means to hide the information so that it cannot be accessed by an unwanted party. In software design, information hiding is achieved by designing the modules in a manner that the information gathered or contained in one module is hidden and can’t be accessed by any other modules.

7. **Refactoring- reconstruct something**

   Refactoring simply means reconstructing something in such a way that it does not affect the behavior of any other features. Refactoring in software design means reconstructing the design to reduce complexity and simplify it without affecting the behavior or its functions. Fowler has defined refactoring as “the process of changing a software system in a way that it won’t affect the behavior of the design and improves the internal structure”.

Different levels of Software Design:

There are three different levels of software design. They are:

1. **Architectural Design:**

   The architecture of a system can be viewed as the overall structure of the system & the way in which structure provides conceptual integrity of the system. The architectural design identifies the software as a system with many components interacting with each other. At this level, the designers get the idea of the proposed solution domain.

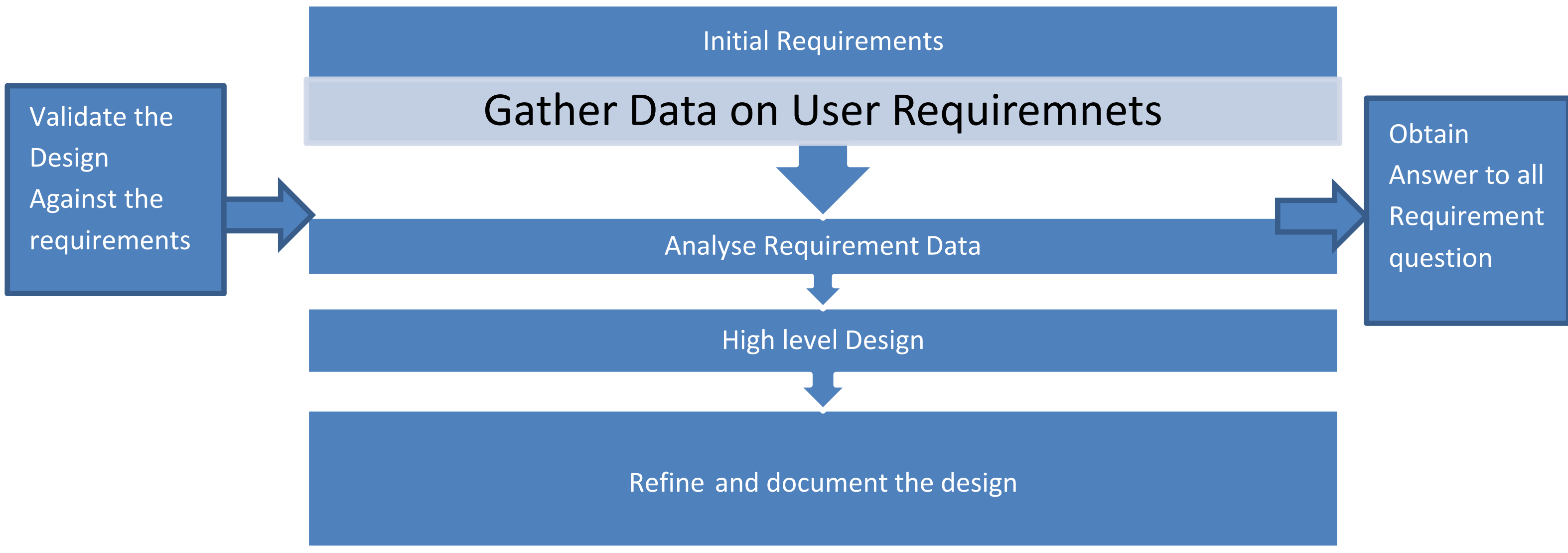
2. **Preliminary or high-level design:**

   Here the problem is decomposed into a set of modules, the control relationship among various modules identified, and also the interfaces among various modules are identified. The outcome of this stage is called the program architecture. Design representation techniques used in this stage are structure chart and UML.

3. **Detailed design:**

   Once the high-level design is complete, a detailed design is undertaken. In detailed design, each module is examined carefully to design the data structure and algorithms. The stage outcome is documented in the form of a module specification document.

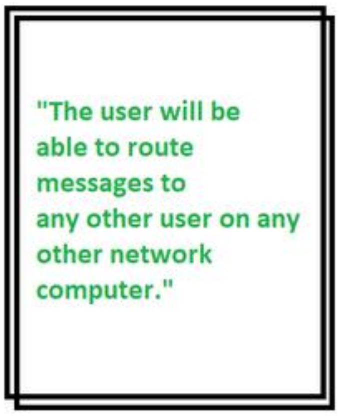
   - Software design is more creative than analysis.
   - It is helpful in problem solving activity.


Initial Requirements
Gather Data on User Requiremnets
Validate the Design Against the requirements
Analyse Requirement Data
Obtain Answer to all Requirement question
High level Design
Refine and document the design

What is Software Design?

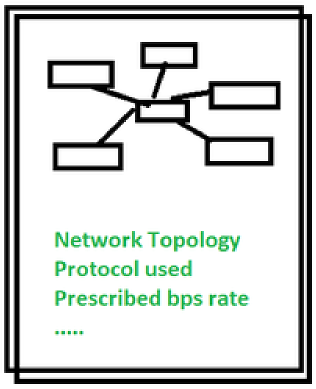
Software design is a process to convert the user requirements into client requirements user interface form, that helps the software developer in coding and implementation. The software design deals with portraying the client's requirement which is described in Software Requirement Specification (SRS) document into a user interface form.
To transform requirements into working system designers must satisfy both customers and therefore the system builders. The customers should understand what the system is to try to do. At an equivalent time, the system builders must understand how to try to. To accomplish the design is split into two parts as shown in the figure below and is named as the 2 parts iterative process. A software design has two parts:

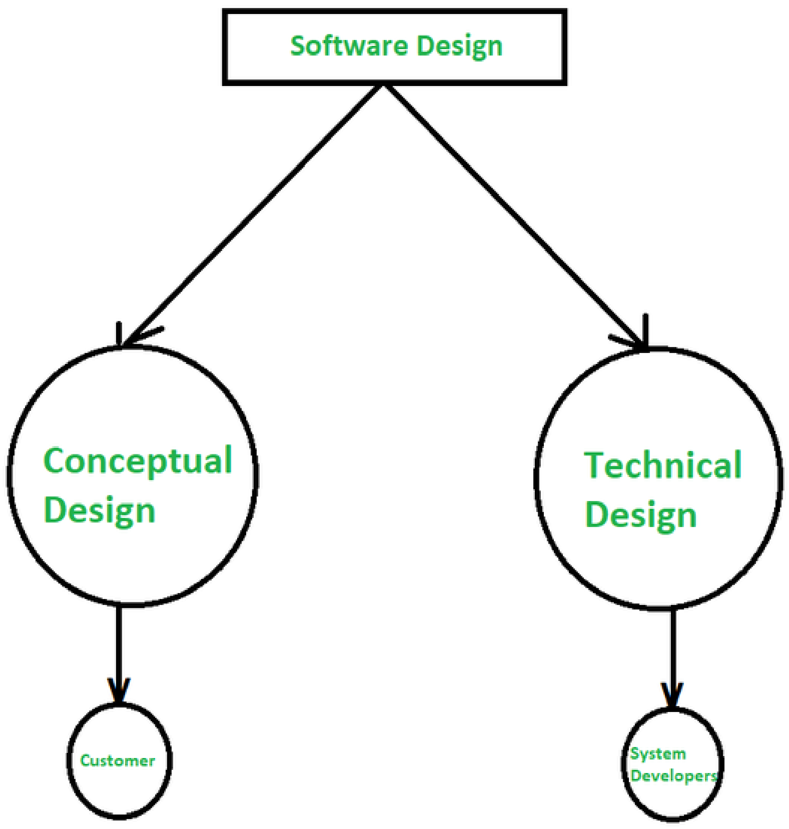
1. **Conceptual Design :**
   Conceptual design is an initial/starting phase in the process of planning, during which the broad outlines of function and sort of something are coupled. It tells the customers that what actually the system will do. An example of Conceptual design is given in the given figure:


"The user will be able to route messages to any other user on any other network computer."

2. Technical Design:

A Technical design is a phase in which the event team writes the code and describes the minute detail of either the whole design or some parts of it. It tells the designers that what actually the system will do. An example of Technical design is given in the given figure:



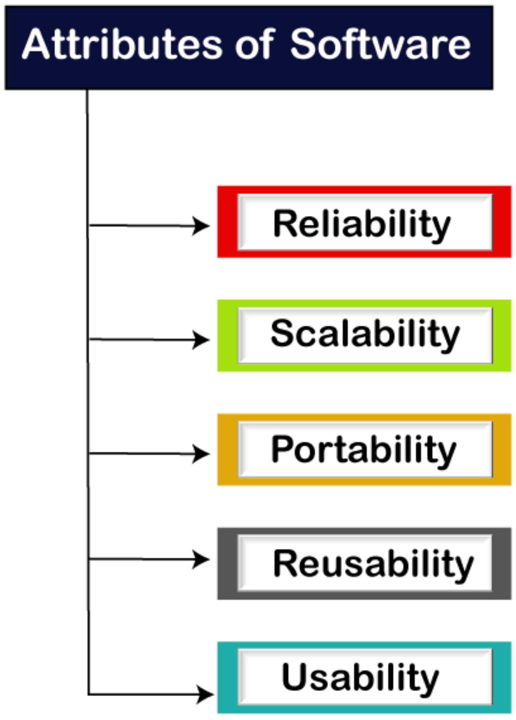

Software Design
Conceptual Design
Technical Design
Customer
System Developers

Difference between conceptual design and technical design :

Conceptual Design	Technical Design
Conceptual design is an initial/starting phase in the process of planning, during which the broad outlines of function and sort of something are coupled.	A Technical design is a phase in which the event team writes the code and describes the minute detail of either the whole design or some parts of it.
It is written in the customer’s language and designed according to the client’s requirements.	It describes any other thing that converts the requirements into a solution to the customer’s problem.
It describes that what will happen to the data in the system.	It describes the functions or methods of the system.
It shows the conceptual model i.e, what should a system look like.	It shows the data flow and the structure of the data.
It also includes the process and sub-processes besides the strategies.	It includes the functioning and working of the conceptual design.
It starts when a system requirement comes and the phase looks for a potential solution.	It starts after setting the system requirements.
At the end of this phase, the solutions to the problems are sent for review.	At the end of this phase, and after analysing the technical design, the specification is initiated.

What is Software Testing

Software testing is a process of identifying the correctness of 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.



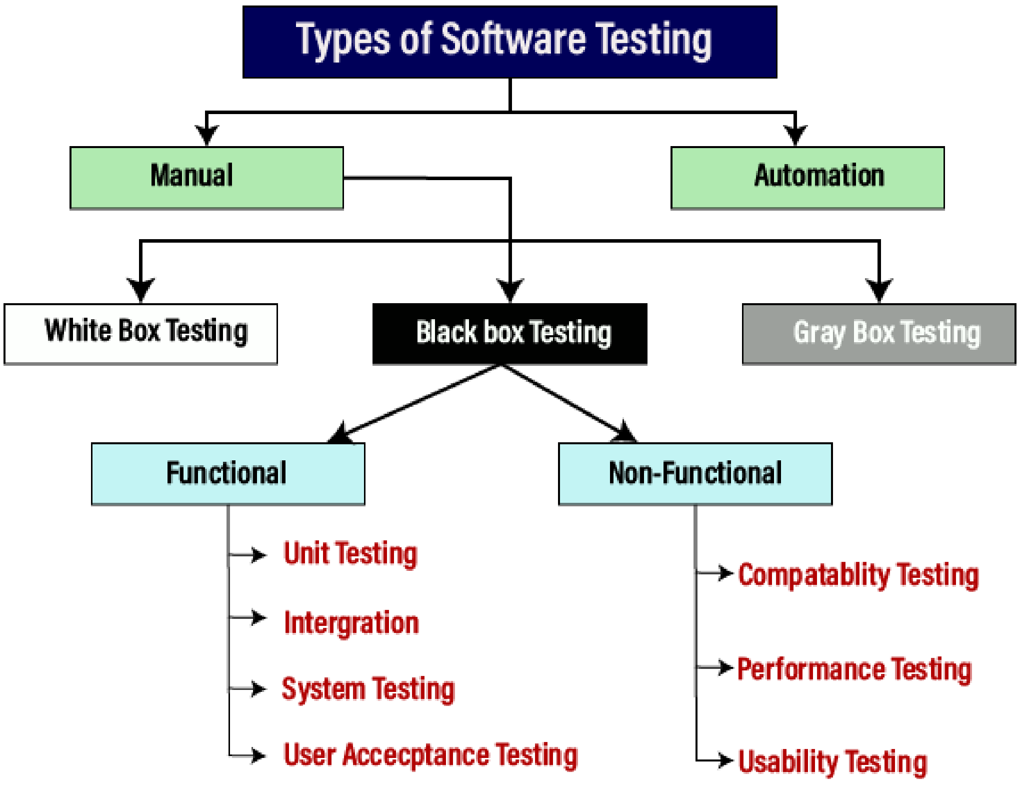
Software testing provides an independent view and objective of the software and gives surety of fitness of the software. It involves testing of all components under the required services to confirm that whether it is satisfying the specified requirements or not. The process is also providing the client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

Type of Software testing

We have various types of testing available in the market, which are used to test the application or the software.

With the help of below image, we can easily understand the type of software testing:



Manual testing

- The process of checking the functionality of an application as per the customer needs without taking any help of automation tools is known as manual testing. While performing the manual testing on any application, we do not need any specific knowledge of any testing tool, rather than have a proper understanding of the product so we can easily prepare the test document.

- 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 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.

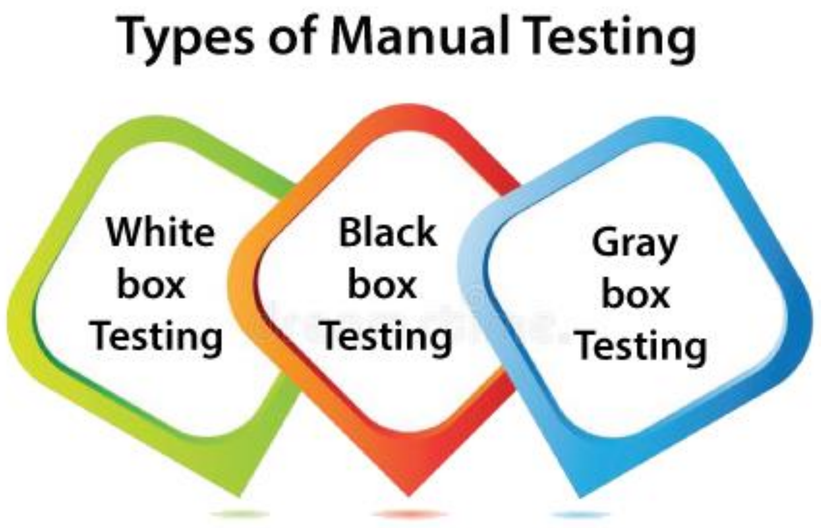
Why we need manual testing

- Whenever an application comes into the market, and it is unstable or having a bug or issues or creating a problem while end-users are using it.

- If we don't want to face these kinds of problems, we need to perform one round of testing to make the application bug free and stable and deliver a quality product to the client, because if the application is bug free, the end-user will use the application more conveniently.

- If the test engineer does manual testing, he/she can test the application as an end-user perspective and get more familiar with the product, which helps them to write the correct test cases of the application and give the quick feedback of the application.

  Manual testing can be further divided into three types of testing, which are as follows:



White-box testing

- The white box testing is done by Developer, where they check every line of a code before giving it to the Test Engineer. Since the code is visible for the Developer during the testing, that's why it is also known as White box testing.

Black box testing

- The black box testing is done by the Test Engineer, where they can check the functionality of an application or the software according to the customer /client's needs. In this, the code is not visible while performing the testing; that's why it is known as black-box testing.

Gray Box testing

- Gray box testing is a combination of white box and Black box testing. It can be performed by a person who knew both coding and testing. And if the single person performs white box, as well as black-box testing for the application, is known as Gray box testing.

How to perform Manual Testing

- First, tester observes all documents related to software, to select testing areas.
- Tester analyses requirement documents 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 the development team.
- The Development team fixes bugs and handed software to the testing team for a retest.

Advantages of Manual Testing

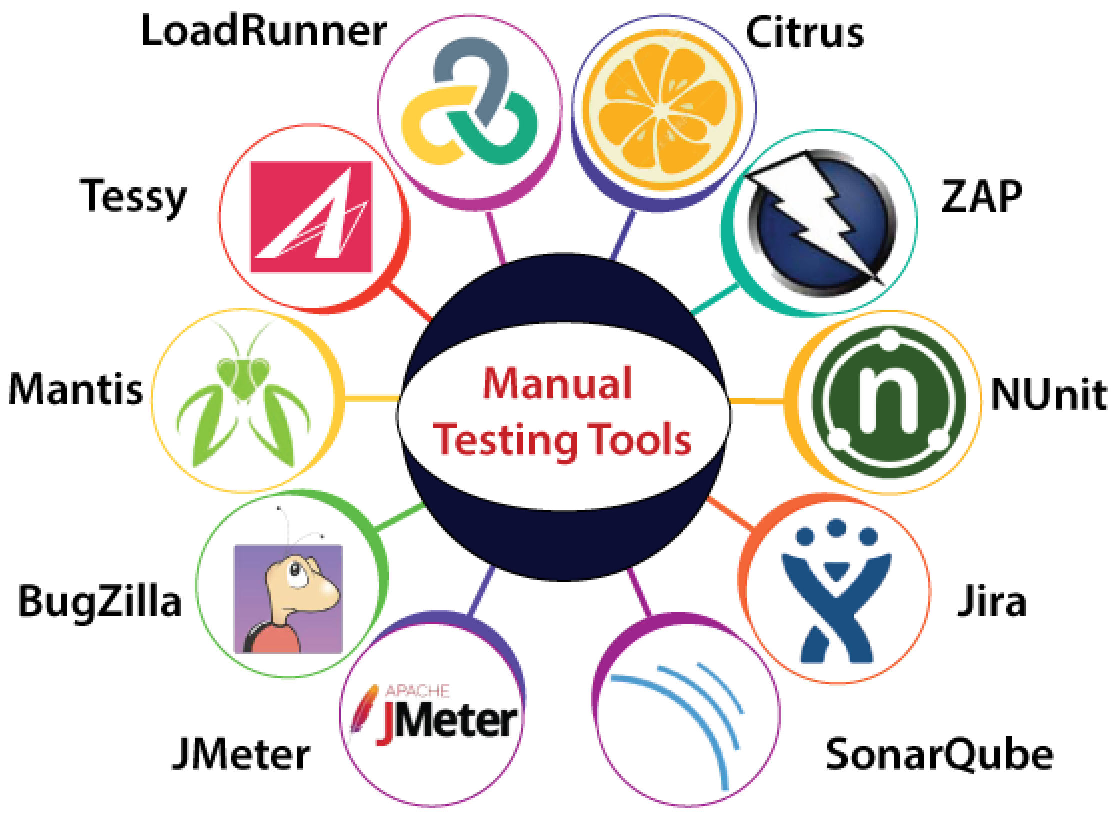
- Manual testing can be done on all kinds of applications
- It is preferable for short life cycle products
- Newly designed test cases should be executed manually.
- Application must be tested manually before it is automated
- It is preferred in the projects where the requirements change frequently and for the products where the GUI changes constantly
- It is cheaper in terms of initial investment compared to Automation testing
- It requires less time and expense to begin productive manual testing
- It allows tester to perform adhoc testing
- There is no necessity to the tester to have knowledge on Automation Tools

Disadvantages of Manual Testing

- Manual Testing is time-consuming mainly while doing regression testing.
- Manual testing is less reliable compared to automation testing because it is conducted by humans. So there will always be prone to errors and mistakes.
- Expense over automation testing in the long run.
  It is not possible to reuse because this process can’t be recorded

Manual testing tools

In manual testing, different types of testing like unit, integration, security, performance, and bug tracking, we have various tools such as Jira, Bugzilla, Mantis, Zap, NUnit, Tessy, LoadRunner, Citrus, SonarQube, etc. available in the market. Some of the tools are open-source, and some are commercial.



Automation Testing

"Automation testing refers to the automatic testing of the software in which developer or tester writes the test script once with the help of testing tools and framework and run it on the software. The test script automatically test the software without human intervention and shows the result (either error, bugs are present or software is free from them)."

- Automation testing needs manual effort when creating initial scripts, and further process is performed automatically to compare the actual testing result with expected results.

- In automation testing, the test automation engineer will write the test script or use the automation testing tools to execute the application. On the other hand, in manual testing, the test engineer will write the test cases and implement the software on the basis of written test cases.

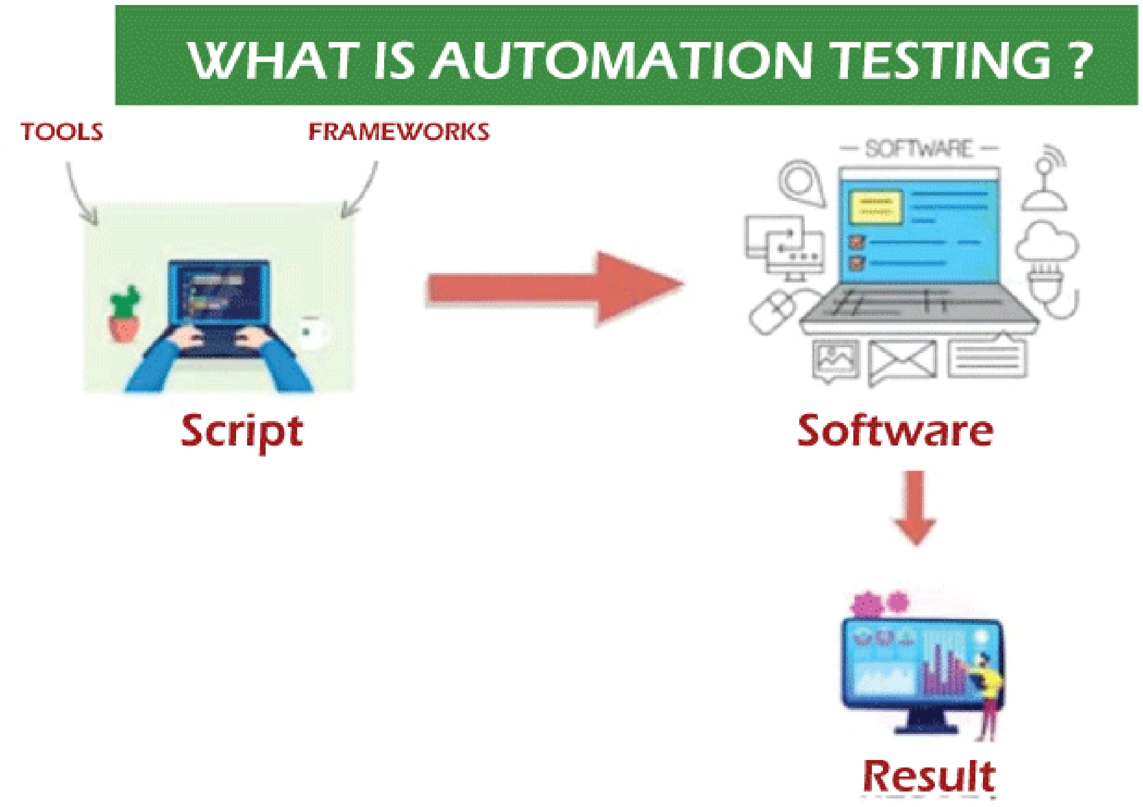
- In test automation, the test engineer can execute repetitive tasks and other related tasks. In manual testing, it is a tedious process to implement the repetitive take again and again.

- In other words, we can say that the main concentration of test Automation is to change the manual human activity with systems or devices.

- The automation testing process is a time-saving process as it spends less time in exploratory testing and more time in keeping the test scripts whereas enhancing the complete test coverage.

The execution of automation testing provides us various advantages, which are as discussed below:

- Reusability
- Consistency
- Running tests anytime (24/7)
- Early Bug detection
- Less Human Resources



1. Reusability

We can re-use the test scripts in automation testing, and we don't need to write the new test scripts again and again. And, we can also re-create the steps which are detailed as the earlier ones.

2. Consistency

As compared to manual testing, automation testing is more consistent and way faster than executing the regular monotonous tests that cannot be missed but may cause faults when tested manually.

3. Running Tests 24/7

In automation testing, we can start the testing process from anywhere in the world and anytime we want to. And even we can do that remotely if we don't have many approaches or the option to purchase them.

4. Early Bug Detection

We can easily detect the critical bugs in the software development process's initial phases by executing automation testing. It also helps us spend fewer working hours to fix these problems and reduce costs

5. Less Human Resources

To implement the automation test script, we need a test automation engineer who can write the test scripts to automate our tests, rather than having several people who are repeatedly performing the tedious manual tests.

Manual testing vs. Automation testing

Software testing is performed to discover bugs in software during its development. The key difference between automation and manual testing are as follows:

Manual testing	**Automation testing**
Testing in which a human tester executes test cases	In automation testing, automation tools are used to execute the test cases
In this testing, human resources are involved, that's why it is time-consuming	It is much faster than the manual testing
It is repetitive and error-prone	Here automated tools are used that make it interesting and accurate
BVT (build verification testing) is time-consuming and tough in manual testing	It's easy to build verification testing
Instead of frameworks, this testing use checklist, guidelines, and stringent process for drafting test cases.	Frameworks like keyword, hybrid, and data drive to accelerate the automation process.
The process turnaround time is higher than the automation testing process (one testing cycle takes lots of time)	It completes a single round of testing within record time; therefore, a process turnaround time is much lower than a manual testing process.
The main goal of manual testing is user-friendliness or improved customer experience.	Automation testing can only guarantee a positive customer experience and user-friendliness.
It is best for usability, exploratory and adhoc testing	It is widely used for performing testing, load testing and regression testing.
Low return on investment	The high return on investment

Automation Testing Methodologies

Automation testing contains the following three different methodologies and approaches, which will help the test engineer to enhance the software product's quality.

- GUI Testing
- Code-Driven
- Test Automation Framework

1. GUI (Graphical user interface) Testing

In this approach, we can implement that software or an application, which contains GUIs. So, that the automation test engineers can record user actions and evaluate them many times.

We know that the Test cases can be written in several programming languages like JAVA, C#, Python, Perl, etc.

2. Code-Driven

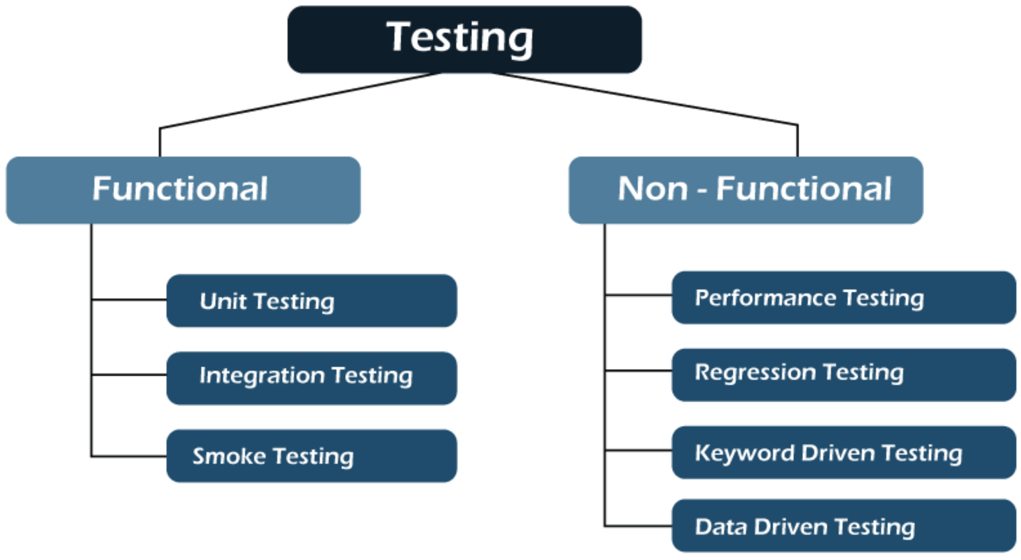
The code-driven technique is the subsequent methodology used in automation testing. In this method, the test engineer will mainly concentrate on test case execution in order to identify whether the several parts of code are performing according to the given requirement or not.

Hence, it is very a commonly used method in agile software development.

3. Test Automation Framework

Another approach in automation testing is test automation framework. The test automation framework is a set of rules used to generate valuable results of the automated testing activity.

Similarly, it brings together test data sources, function libraries, object details, and other reusable modules.



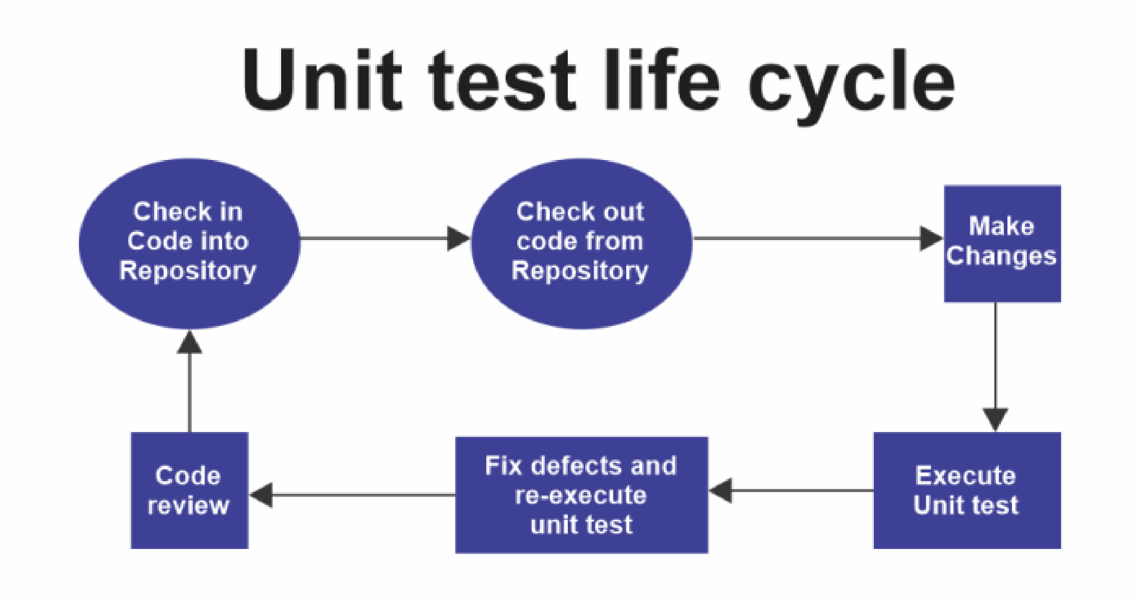
Functional Testing

The first test performed by tester on newly revised software is called functional testing, which verifies all the software functions' features per user requirement. This testing works on the real-world business application and obtaining the expected output from a given input. All application functions are tested and involve smoke, unit, and integration testing.

a) Unit testing

The unit is the smallest component of the software that functions individually. Unit testing simplifies the testing of the whole software, where each software element is fully tested before the final version is out. Unit testing depicts how the code performs at each part and has a faster execution time.

It's the favourite of developers because it consumes less time and assure the working of each part of the software. Before automation testing, the developers write the code for testing, but now there is no need. The unit testing technique is divided into three broad categories: White box testing, Black box testing and Grey box testing.

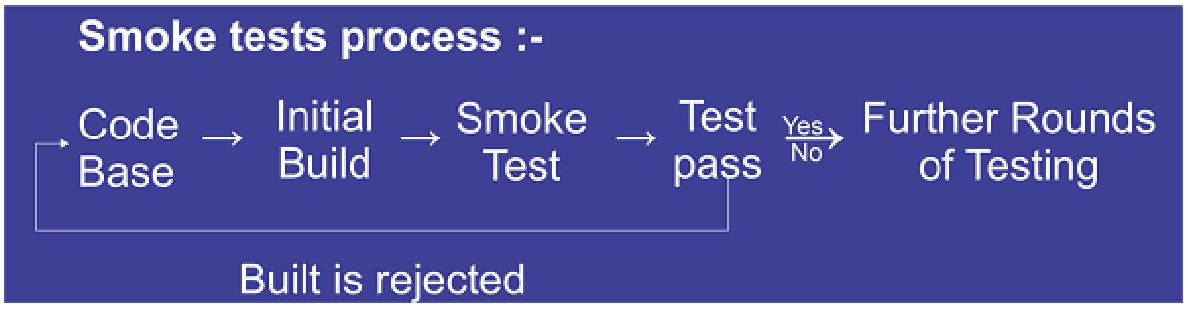

Unit test life cycle
Check in Code into Repository
Check out code from Repository
Make Changes
Execute Unit test
Fix defects and re-execute unit test
Code review

b) Integration testing

Integration testing is more complicated to set up compared with other tests. All the modules of the application communicate with each other to perform tasks. Therefore, testers group them for testing and exposing the flaws in maintaining the interaction between these modules. Another name for this testing is I&T or string testing, considered end-to-end.

c) Smoke testing

This testing checks and defines the product's stability (whether stable or not). If the product result is unstable, it is called an 'unstable build' and sent back to developers, where they run more test cases to find out the root cause of the problem. The smoke test works like this:-



Non-functional Testing

Non-functional testing focuses on how well application functions are doing, not on what the product does. It is the opposite of functional testing, where application elements like reliability, usability, performance, etc., are tested. Some types of non-functional testing are reliability testing, load testing, compatibility testing, performance testing, security testing etc.

a) Performance testing

This non-functional testing tests the software's stability, responsiveness and speed under the workload. It finds out the potential issues faced by critical software and medical programs used by the user, like slow operation of software under stressful circumstances. It finds hurdles in the performance of software and removes them to increase the ability of software to deliver the best results to the end user.

b) Regression testing

When some changes are made to the code of software or application, it needs to be tested to determine whether the software is working as before the change; for this purpose, testers use automation regression testing to automate scripts, applications of workflows, plans and other activities. It tests the system or software workflow after its updation and functional error.

c) Keyword driven testing

Keyword-driven testing tests the application using the data files consisting of the keywords related to the application, representing a set of actions needed to carry out the step. Here these specific keywords are identified and connected with the specific action. Therefore during testing, when these keywords are used, their related actions will automatically be done. This keyword testing is a popular choice for many businesses as it's flexible, concise, easy to maintain and reusable. Keyword-driven testing is compatible with all kinds of automation tools in the market. Instead of programming experts, functional testers can plan the testing before the application is fully developed.

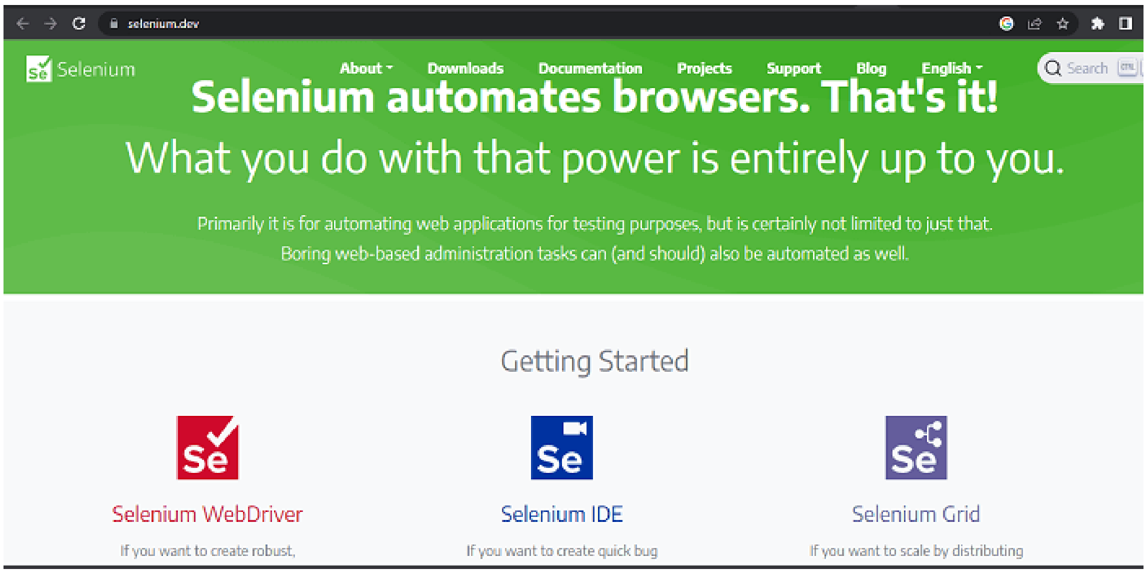
d) Data-driven testing

In data-driven testing, automation is inbuilt and very effective due to the few facilities provided, like the reusability of code, change in the script doesn't affect the test cases, and this testing can be carried out in the phase of the software development cycle. It provides consistency in results and reduces the investment of time and resources. Test cases use the data separately stored in the table or spreadsheet format, and testers have multiple data sets for testing.

Automation Testing Tools

Automation testing tools can describe in two categories, which are as follows:

- Functional Testing Tools
- Non-Functional Testing Tools



Selenium automates browsers. That's it!

What you do with that power is entirely up to you.

Primarily it is for automating web applications for testing purposes, but is certainly not limited to just that. Boring web-based administration tasks can (and should) also be automated as well.

Getting Started

Selenium WebDriver
If you want to create robust,

Selenium IDE
If you want to create quick bug

Selenium Grid
If you want to scale by distributing

Functional Automation Testing Tools

The automation test engineer uses the functional automation testing tool to implement the functional test cases. For example, the Repetitive Regression tests are automated under the function automation testing tools.

This type of tools can be further divided into two different parts, which are as below:

- **Commercial Tool**
- **Open-source Tool**

Commercial Functional Testing Tools

Commercial functional testing tools are those testing tools, which are not available freely in the market. These tools are also known as licensed tools. The licensed tools include various features and approaches as compared to the open-source tools.

Some of the most important commercial tools are as follows:

- QTP[Quick Test Professional]
- Rational Functional Tester [RFT]
- TestComplete
- SoapUI

Open-source Functional Testing Tools

Open-source functional testing tools are those tools, which are available freely in the market. These tools have less functionality and features than the commercial/licensed tools, but sometimes working on the commercial tool became an expensive process.

That's why some well-known organizations preferred to use open-source tools.

Following are the most commonly used open-source functional automation testing tools:

- Selenium
- Sikuli
- Autoit