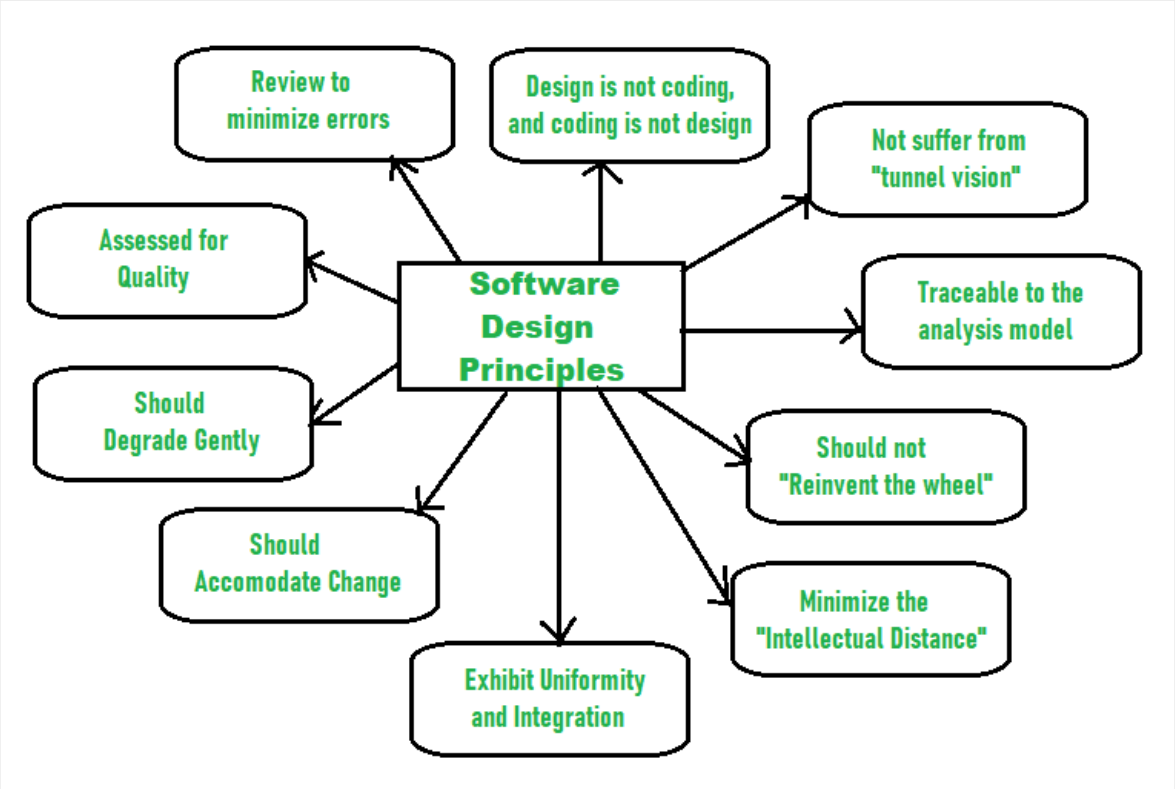
Unit 2

Principles of Software Design
Software Design is also a process to plan or convert the software requirements into a step that are needed to be carried out to develop a software system. There are several principles that are used to organize and arrange the structural components of Software design. Software Designs in which these principles are applied affect the content and the working process of the software from the beginning. These principles are stated below :



Principles Of Software Design :
1. **Should not suffer from "Tunnel Vision" –**
While designing the process, it should not suffer from "tunnel vision" which means that is should not only focus on completing or achieving the aim but on other effects also.

2. **Traceable to analysis model –**
The design process should be traceable to the analysis model which means it should satisfy all the requirements that software requires to develop a high-quality product.

3. **Should not "Reinvent The Wheel" –**
The design process should not reinvent the wheel that means it should not waste time or effort in creating things that already exist.

Due to this, the overall development will get increased.

4. **Minimize Intellectual distance –**
The design process should reduce the gap between real-world problems and software solutions for that problem meaning it should simply minimize intellectual distance.

5. **Exhibit uniformity and integration –**
The design should display uniformity which means it should be uniform throughout the process without any change. Integration means it should mix or combine all parts of software i.e. subsystems into one system.

6. **Accommodate change –**
The software should be designed in such a way that it accommodates the change implying that the software should adjust to the change that is required to be done as per the user's need.

7. **Degrade gently –**
The software should be designed in such a way that it degrades gracefully which means it should work properly even if an error occurs during the execution.

8. **Assessed or quality –**
The design should be assessed or evaluated for the quality meaning that during the evaluation, the quality of the design needs to be checked and focused on.

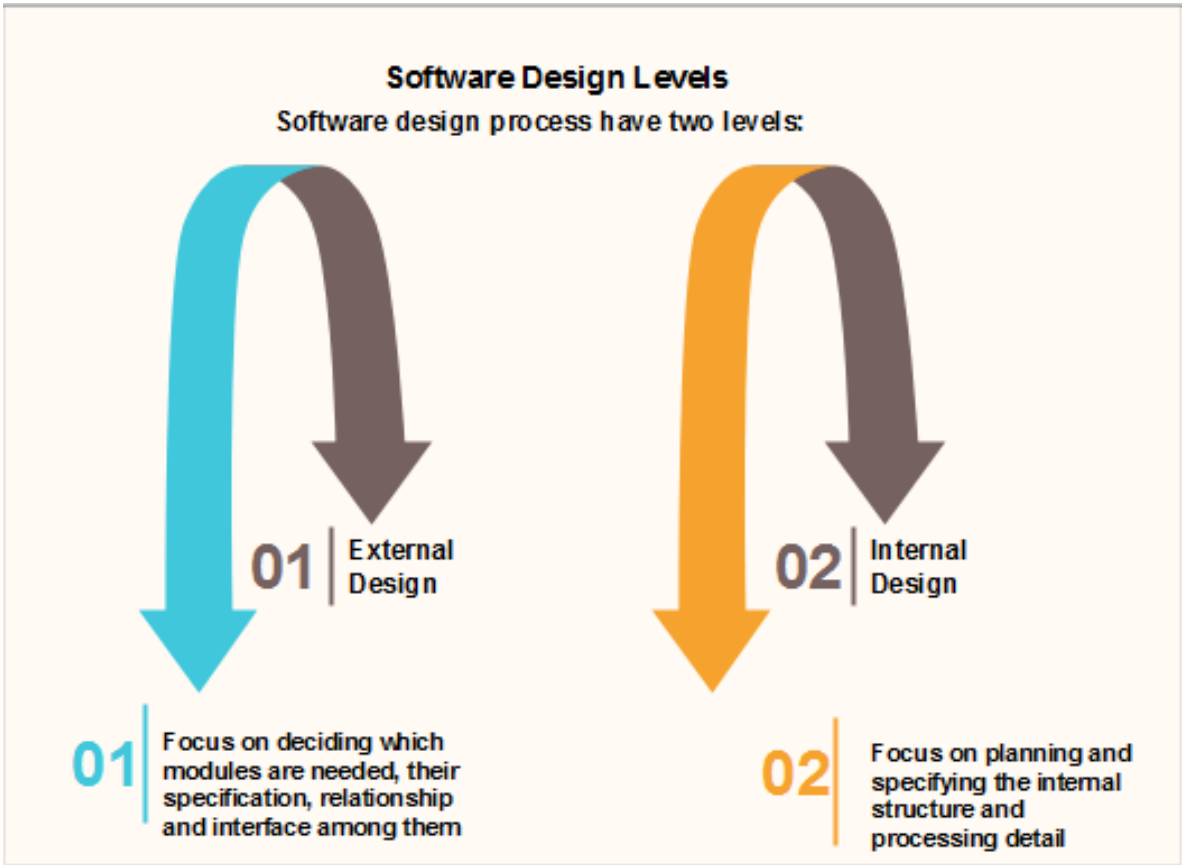
9. **Review to discover errors –**
The design should be reviewed which means that the overall evaluation should be done to check if there is any error present or if it can be minimized.

10. **Design is not coding and coding is not design –**
Design means describing the logic of the program to solve any problem and coding is a type of language that is used for the implementation of a design.

Software Design
Software design is a mechanism to transform user requirements into some suitable form, which helps the programmer in software coding and implementation. It deals with representing the client's

requirement, as described in SRS (Software Requirement Specification) document, into a form, i.e., easily implementable using programming language.

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



Objectives of Software Design

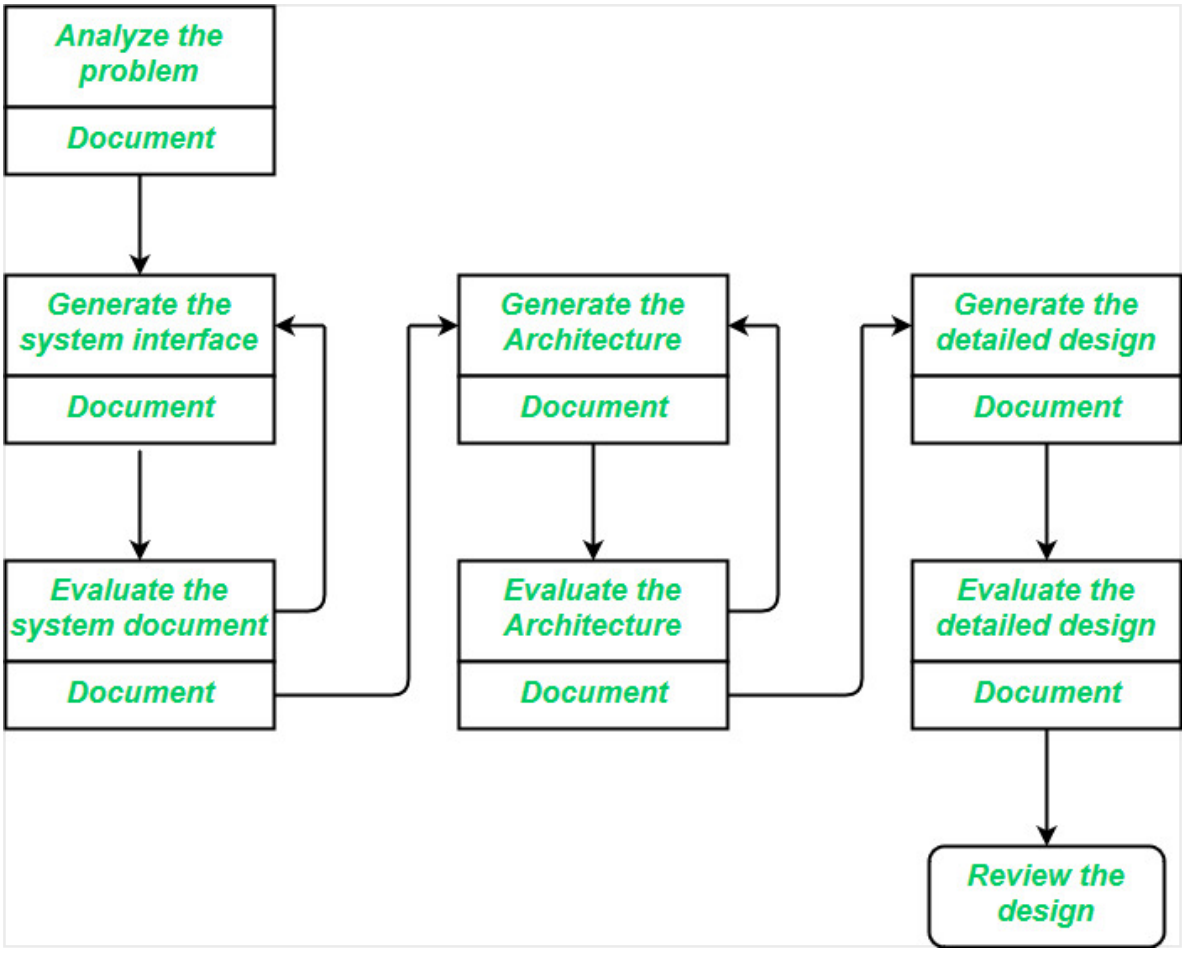
Following are the purposes of Software design:

1. **Correctness:**Software design should be correct as per requirement.
2. **Completeness:**The design should have all components like data structures, modules, and external interfaces, etc.
3. **Efficiency:**Resources should be used efficiently by the program.
4. **Flexibility:**Able to modify on changing needs.
5. **Consistency:**There should not be any inconsistency in the design.
6. **Maintainability:** The design should be so simple so that it can be easily maintainable by other designers.

The design phase of software development deals with transforming the customer requirements as described in the SRS documents into a form implementable using a programming language.

The software design process can be divided into the following three

levels of phases of design:
1. Interface Design
2. Architectural Design
3. Detailed Design



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

• Precise description of the events or messages that the system must produce.
• Specification on the data, and the formats of the data coming into and going out of the system.
• Specification of the ordering and timing relationships between incoming events or messages, and outgoing events or outputs.

User Interface Design

The visual part of a computer application or operating system through which a client interacts with a computer or software. It determines how commands are given to the computer or the program and how data is displayed on the screen.

Types of User Interface

There are two main types of User Interface:
o Text-Based User Interface or Command Line Interface
o Graphical User Interface (GUI)

Text-Based User Interface: This method relies primarily on the keyboard. A typical example of this is UNIX.

Advantages
o Many and easier to customizations options.
o Typically capable of more important tasks.

Disadvantages
o Relies heavily on recall rather than recognition.
o Navigation is often more difficult.

Graphical User Interface (GUI): GUI relies much more heavily on the mouse. A typical example of this type of interface is any versions of the Windows operating systems.

GUI Characteristics

Characteristics	Descriptions
Windows	Multiple windows allow different information to be displayed simultaneously on the user's screen.
Icons	Icons different types of information. On some systems, icons represent files. On other icons describes processes.
Menus	Commands are selected from a menu rather than typed in a command language.

Pointing	A pointing device such as a mouse is used for selecting choices from a menu or indicating items of interests in a window.
Graphics	Graphics elements can be mixed with text or the same display.

Advantages
o Less expert knowledge is required to use it.
o Easier to Navigate and can look through folders quickly in a guess and check manner.
o The user may switch quickly from one task to another and can interact with several different applications.
Disadvantages
o Typically decreased options.
o Usually less customizable. Not easy to use one button for tons of different variations.

UI Design Principles

Structure: Design should organize the user interface purposefully, in the meaningful and usual based on precise, consistent models that are apparent and recognizable to users, putting related things together and separating unrelated things, differentiating dissimilar things and making similar things resemble one another. The structure principle is concerned with overall user interface architecture.

Simplicity: The design should make the simple, common task easy, communicating clearly and directly in the user's language, and providing good shortcuts that are meaningfully related to longer procedures.

Visibility: The design should make all required options and materials for a given function visible without distracting the user with extraneous or redundant data.

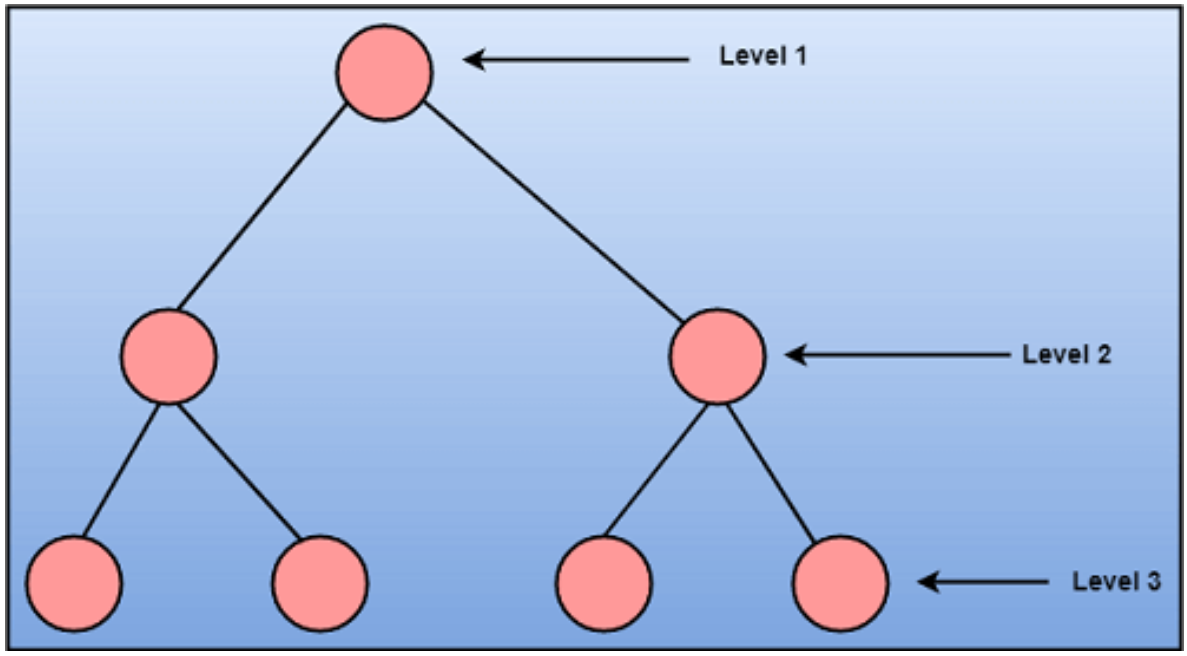
Feedback: The design should keep users informed of actions or interpretation, changes of state or condition, and bugs or exceptions that are relevant and of interest to the user through clear, concise, and unambiguous language familiar to users.

Tolerance: The design should be flexible and tolerant, decreasing the cost of errors and misuse by allowing undoing and redoing while also preventing bugs wherever possible by tolerating varied inputs and sequences and by interpreting all reasonable actions.

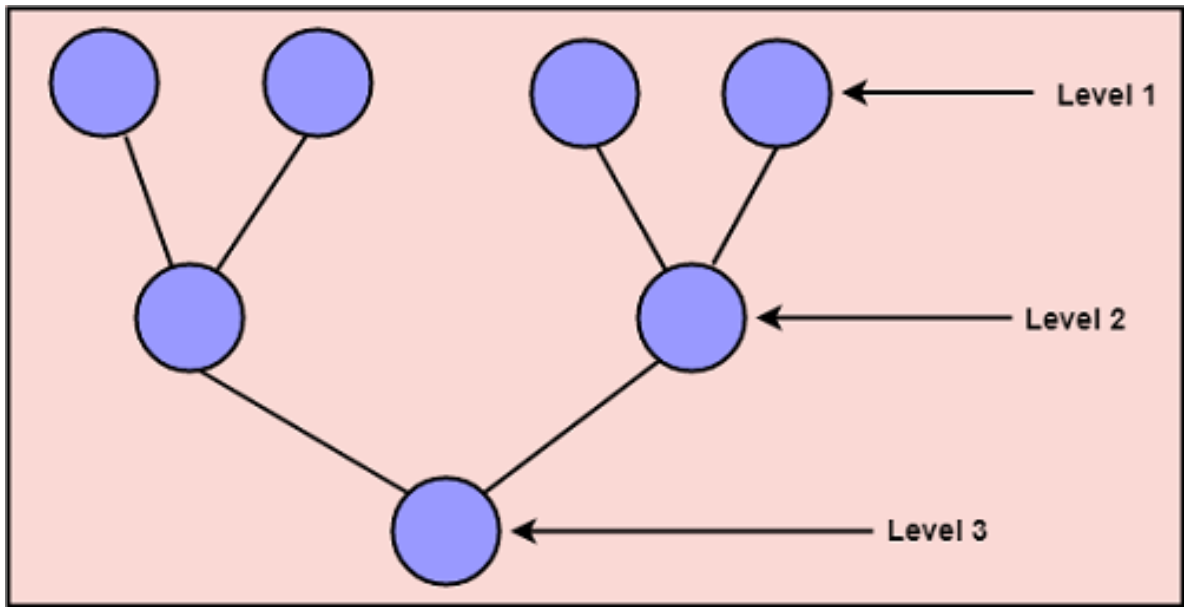
Architectural Design:
Architectural design is the specification of the major components of a system, their responsibilities, properties, interfaces, and the relationships and interactions between them. In architectural design, the overall structure of the system is chosen, but the internal details of major components are ignored.
Issues in architectural design includes:
• Gross decomposition of the systems into major components.
• Allocation of functional responsibilities to components.
• Component Interfaces
• Component scaling and performance properties, resource consumption properties, reliability properties, and so forth.
• Communication and interaction between components.
The architectural design adds important details ignored during the interface design. Design of the internals of the major components is ignored until the last phase of the design.
Detailed Design:
Design is the specification of the internal elements of all major system components, their properties, relationships, processing, and often their algorithms and the data structures.
The detailed design may include:
• Decomposition of major system components into program units.
• Allocation of functional responsibilities to units.
• User interfaces
• Unit states and state changes
• Data and control interaction between units
• Data packaging and implementation, including issues of scope and visibility of program elements
• Algorithms and data structures

Strategy of Design
A good system design strategy is to organize the program modules in such a method that are easy to develop and latter too, change. Structured design methods help developers to deal with the size and complexity of programs. Analysts generate instructions for the developers about how code should be composed and how pieces of code should fit together to form a program.
To design a system, there are two possible approaches:
1. Top-down Approach
2. Bottom-up Approach
1. Top-down Approach: This approach starts with the identification

of the main components and then decomposing them into their more detailed sub-components.



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



Coding

The coding is the process of transforming the design of a system into a computer language format. This coding phase of software development is concerned with software translating design specification into the source code. It is necessary to write source code & internal documentation so that conformance of the code to its specification can be easily verified.

Coding is done by the coder or programmers who are independent people than the designer. The goal is not to reduce the effort and

cost of the coding phase, but to cut to the cost of a later stage. The cost of testing and maintenance can be significantly reduced with efficient coding.

Goals of Coding

1. **To translate the design of system into a computer language format:** The coding is the process of transforming the design of a system into a computer language format, which can be executed by a computer and that perform tasks as specified by the design of operation during the design phase.
2. **To reduce the cost of later phases:** The cost of testing and maintenance can be significantly reduced with efficient coding.
3. **Making the program more readable:** Program should be easy to read and understand. It increases code understanding having readability and understandability as a clear objective of the coding activity can itself help in producing more maintainable software.

For implementing our design into code, we require a high-level functional language. A programming language should have the following characteristics:

Characteristics of Programming Language

Following are the characteristics of Programming Language:

Readability: A good high-level language will allow programs to be written in some methods that resemble a quite-English description of the underlying functions. The coding may be done in an essentially self-documenting way.

Portability: High-level languages, being virtually machine-independent, should be easy to develop portable software.

Generality: Most high-level languages allow the writing of a vast collection of programs, thus relieving the programmer of the need to develop into an expert in many diverse languages.

Brevity: Language should have the ability to implement the algorithm with less amount of code. Programs mean in high-level languages are often significantly shorter than their low-level equivalents.

Error checking: A programmer is likely to make many errors in the development of a computer program. Many high-level languages invoke a lot of bugs checking both at compile-time and run-time.

Cost: The ultimate cost of a programming language is a task of many of its characteristics.

Quick translation: It should permit quick translation.

Efficiency: It should authorize the creation of an efficient object code.

Modularity: It is desirable that programs can be developed in the language as several separately compiled modules, with the

appropriate structure for ensuring self-consistency among these modules.

Widely available: Language should be widely available, and it should be feasible to provide translators for all the major machines and all the primary operating systems.

A coding standard lists several rules to be followed during coding, such as the way variables are to be named, the way the code is to be laid out, error return conventions, etc.

Coding Guidelines

General coding guidelines provide the programmer with a set of the best methods which can be used to make programs more comfortable to read and maintain. Most of the examples use the C language syntax, but the guidelines can be tested to all languages.

The following are some representative coding guidelines recommended by many software development organizations.

1. Line Length: It is considered a good practice to keep the length of source code lines at or below 80 characters. Lines longer than this may not be visible properly on some terminals and tools. Some printers will truncate lines longer than 80 columns.

2. Spacing: The appropriate use of spaces within a line of code can improve readability.

Example:

Bad:
```
cost=price+(price*sales_tax)
fprintf(stdout ,"The total cost is %5.2f\n",cost);
```

Better:
```
cost = price + ( price * sales_tax )
fprintf (stdout,"The total cost is %5.2f\n",cost);
```

3. The code should be well-documented: As a rule of thumb, there must be at least one comment line on the average for every three-source line.

4. The length of any function should not exceed 10 source lines: A very lengthy function is generally very difficult to understand as it possibly carries out many various functions. For the same reason, lengthy functions are possible to have a disproportionately larger number of bugs.

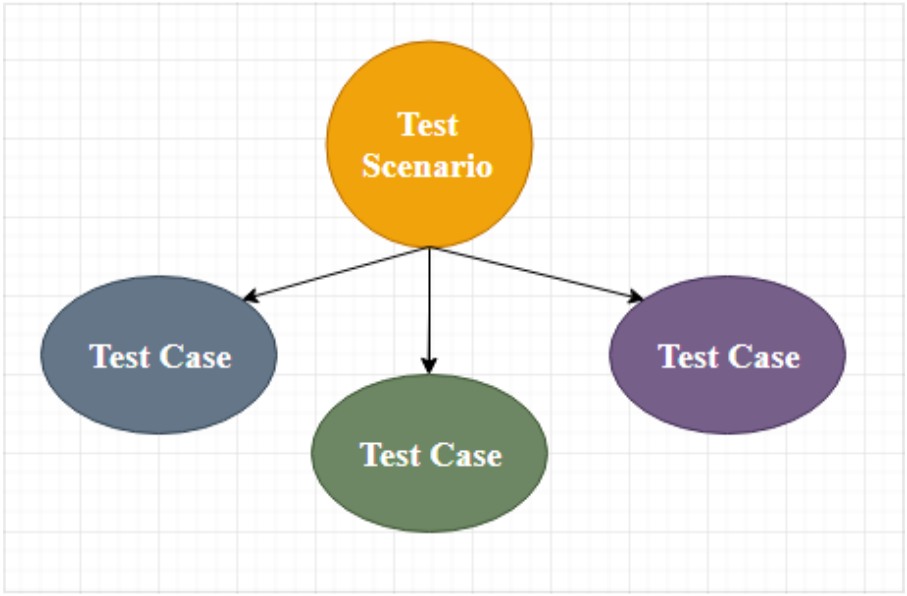
5. Do not use goto statements: Use of goto statements makes a program unstructured and very tough to understand.

6. Inline Comments: Inline comments promote readability.

7. Error Messages: Error handling is an essential aspect of computer programming. This does not only include adding the necessary logic to test for and handle errors but also involves making error messages meaningful.

Test Case

The test case is defined as a group of conditions under which a tester determines whether a software application is working as per the customer's requirements or not. Test case designing includes preconditions, case name, input conditions, and expected result. A test case is a first level action and derived from test scenarios.



It is an in-details document that contains all possible inputs (positive as well as negative) and the navigation steps, which are used for the test execution process. Writing of test cases is a one-time attempt that can be used in the future at the time of regression testing.

Test case gives detailed information about testing strategy, testing process, preconditions, and expected output. These are executed during the testing process to check whether the software application is performing the task for that it was developed or not.

Test case helps the tester in defect reporting by linking defect with test case ID. Detailed test case documentation works as a full proof guard for the testing team because if developer missed something, then it can be caught during execution of these full-proof test cases.

To write the test case, we must have the requirements to derive the inputs, and the test scenarios must be written so that we do not miss out on any features for testing. Then we should have the test case template to maintain the uniformity, or every test engineer follows the same approach to prepare the test document.

Generally, we will write the test case whenever the developer is busy in writing the code.

When do we write a test case?

We will write the test case when we get the following:

o When the customer gives the business needs then, the developer starts developing and says that they need 3.5 months to build this product.

o And In the meantime, the testing team will **start writing the test cases**.

o Once it is done, it will send it to the Test Lead for the review process.

o And when the developers finish developing the product, it is handed over to the testing team.

o The test engineers never look at the requirement while testing the product document because testing is constant and does not depends on the mood of the person rather than the quality of the test engineer.

Note: When writing the test case, the actual result should never be written as the product is still being in the development process. That? s why the actual result should be written only after the execution of the test cases.

Why we write the test cases?

We will write the test for the following reasons:

o **To require consistency in the test case execution**
o **To make sure a better test coverage**
o **It depends on the process rather than on a person**
o **To avoid training for every new test engineer on the product**

To require consistency in the test case execution: we will see the test case and start testing the application.

To make sure a better test coverage: for this, we should cover all possible scenarios and document it, so that we need not remember all the scenarios again and again.

It depends on the process rather than on a person: A test engineer has tested an application during the first release, second release, and left the company at the time of third release. As the test engineer understood a module and tested the application thoroughly by deriving many values. If the person is not there for the third release, it becomes difficult for the new person. Hence all the derived values are documented so that it can be used in the future.

To avoid giving training for every new test engineer on the product: When the test engineer leaves, he/she leaves with a lot of knowledge and scenarios. Those scenarios should be documented so that the new test engineer can test with the given scenarios and also can write the new scenarios.

Types of test cases

We have a different kind of test cases, which are as follows:

o **Function test cases**
o **Integration test cases**
o **System test cases**

The functional test cases

Firstly, we check for which field we will write test cases and then describe accordingly.

In functional testing or if the application is data-driven, we require the input column else; it is a bit time-consuming.

Rules to write functional test cases:

o In the expected results column, try to use **should be** or **must be**.
o Highlight the Object names.
o We have to describe only those steps which we required the most; otherwise, we do not need to define all the steps.
o To reduce the excess execution time, we will write steps correctly.
o Write a generic test case; do not try to hard code it.

Integration test case

In this, we should not write something which we already covered in the functional test cases, and something we have written in the integration test case should not be written in the system test case

again.

Rules to write integration test cases

o Firstly, understand the product
o Identify the possible scenarios
o Write the test case based on the priority

When the test engineer writing the test cases, they may need to consider the following aspects:

If the test cases are in details:

o They will try to achieve maximum test coverage.
o All test case values or scenarios are correctly described.
o They will try to think about the execution point of view.
o The template which is used to write the test case must be unique.

System test cases

We will write the system test cases for the end-to-end business flows. And we have the entire modules ready to write the system test cases.

The process to write test cases

The method of writing a test case can be completed into the following steps, which are as below:

System study

In this, we will understand the application by looking at the requirements or the SRS, which is given by the customer.

Identify all scenarios:

o When the product is launched, what are the possible ways the end-user may use the software to identify all the possible ways.
o I have documented all possible scenarios in a document, which is called test design/high-level design.
o The test design is a record having all the possible scenarios.

Write test cases

Convert all the identified scenarios to test claims and group the scenarios related to their features, prioritize the module, and write test cases by applying test case design techniques and use the standard test case template, which means that the one which is decided for the project.

Review the test cases

Review the test case by giving it to the head of the team and, after that, fix the review feedback given by the reviewer.

Test case approval

After fixing the test case based on the feedback, send it again for the approval.

Store in the test case repository

After the approval of the particular test case, store in the familiar

place that is known as the test case repository.

Reliability Testing is a testing technique that relates to test the ability of a software to function and given environmental conditions that helps in uncovering issues in the software design and functionality. It is defined as a type of software testing that determines whether the software can perform a failure free operation for a specific period of time in a specific environment. It ensures that the product is fault free and is reliable for its intended purpose.

Objective of Reliability Testing:

The objective of reliability testing is:

• To find the perpetual structure of repeating failures.

• To find the number of failures occurring is the specific period of time.

• To discover the main cause of failure.

• To conduct performance testing of various modules of software product after fixing defects.

Types of Reliability Testing:

There are three types of reliability testing:-

1. **Feature Testing:**

Following three steps are involved in this testing:

• Each function in the software should be executed at least once.

• Interaction between two or more functions should be reduced.

• Each function should be properly executed.

2. **Regression Testing:**

Regression testing is basically performed whenever any new functionality is added, old functionalities are removed or the bugs are fixed in an application to make sure with introduction of new functionality or with the fixing of previous bugs, no new bugs are introduced in the application.

3. **Load Testing:**

Load testing is carried out to determine whether the application is supporting the required load without getting breakdown. It is performed to check the performance of the software under maximum work load.

Debugging

In the development process of any software, the software program is religiously tested, troubleshot, and maintained for the sake of delivering bug-free products. There is nothing that is error-free in the first go.

So, it's an obvious thing to which everyone will relate that as when the software is created, it contains a lot of errors; the reason being

nobody is perfect and getting error in the code is not an issue, but avoiding it or not preventing it, is an issue!
All those errors and bugs are discarded regularly, so we can conclude that debugging is nothing but a process of eradicating or fixing the errors contained in a software program.
Debugging works stepwise, starting from identifying the errors, analyzing followed by removing the errors. Whenever a software fails to deliver the result, we need the software tester to test the application and solve it.
Since the errors are resolved at each step of debugging in the software testing, so we can conclude that it is a tiresome and complex task regardless of how efficient the result was.

Why do we need Debugging?

Debugging gets started when we start writing the code for the software program. It progressively starts continuing in the consecutive stages to deliver a software product because the code gets merged with several other programming units to form a software product.
Following are the benefits of Debugging:
o Debugging can immediately report an error condition whenever it occurs. It prevents hampering the result by detecting the bugs in the earlier stage, making software development stress-free and smooth.
o It offers relevant information related to the data structures that further helps in easier interpretation.
o Debugging assist the developer in reducing impractical and disrupting information.
o With debugging, the developer can easily avoid complex one-use testing code to save time and energy in software development.

Steps involved in Debugging

Following are the different steps that are involved in debugging:
1. **Identify the Error:** Identifying an error in a wrong may result in the wastage of time. It is very obvious that the production errors reported by users are hard to interpret, and sometimes the information we receive is misleading. Thus, it is mandatory to identify the actual error.
2. **Find the Error Location:** Once the error is correctly discovered, you will be required to thoroughly review the code repeatedly to locate the position of the error. In general, this step focuses on finding the error rather than perceiving it.
3. **Analyze the Error:** The third step comprises error analysis, a bottom-up approach that starts from the location of the error followed by analyzing the code. This step makes it easier to

comprehend the errors. Mainly error analysis has two significant goals, i.e., evaluation of errors all over again to find existing bugs and postulating the uncertainty of incoming collateral damage in a fix.
4. **Prove the Analysis:** After analyzing the primary bugs, it is necessary to look for some extra errors that may show up on the application. By incorporating the test framework, the fourth step is used to write automated tests for such areas.
5. **Cover Lateral Damage:** The fifth phase is about accumulating all of the unit tests for the code that requires modification. As when you run these unit tests, they must pass.
6. **Fix & Validate:** The last stage is the fix and validation that emphasizes fixing the bugs followed by running all the test scripts to check whether they pass.

Debugging Strategies

1. **Brute Force:** Study the system for a larger duration in order to understand the system. It helps the debugger to construct different representations of systems to be debugging depending on the need. A study of the system is also done actively to find recent changes made to the software.
2. **Backtracking:** Backward analysis of the problem which involves tracing the program backward from the location of the failure message in order to identify the region of faulty code. A detailed study of the region is conducted to find the cause of defects.
3. **Forward analysis** of the program involves tracing the program forwards using breakpoints or print statements at different points in the program and studying the results. The region where the wrong outputs are obtained is the region that needs to be focused on to find the defect.
4. **Using the past experience** of the software debug the software with similar problems in nature. The success of this approach depends on the expertise of the debugger.
5. **Cause elimination:** it introduces the concept of binary partitioning. Data related to the error occurrence are organized to isolate potential causes.

Debugging Tools

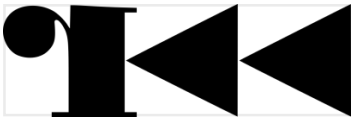
The debugging tool can be understood as a computer program that is used to test and debug several other programs. Presently, there are many public domain software such as **gdb** and **dbx** in the market, which can be utilized for debugging. These software offers console-based command-line interfaces. Some of the automated debugging tools include code-based tracers, profilers, interpreters, etc.
Here is a list of some of the widely used debuggers:

o Radare2
o WinDbg
o Valgrind

Radare2

Radare2 is known for its reverse engineering framework as well as binary analysis. It is made up of a small set of utilities, either utilized altogether or independently from the command line. It is also known as r2.

It is constructed around disassembler for computer software for generating assembly language source code from machine-executable code. It can support a wide range of executable formats for distinct architectures of processors and operating systems.



WinDbg

WinDbg is a multipurpose debugging tool designed for Microsoft Windows operating system. This tool can be used to debug the memory dumps created just after the Blue Screen of Death that further arises when a bug check is issued. Besides, it is also helpful in debugging the user-mode crash dumps, which is why it is called post-mortem debugging.

Valgrind

The Valgrind exist as a tool suite that offers several debugging and

profiling tools to facilitate users in making faster and accurate program. Memcheck is one of its most popular tools, which can successfully detect memory-related errors caused in C and C++ programs as it may crash the program and result in unpredictable behavior.

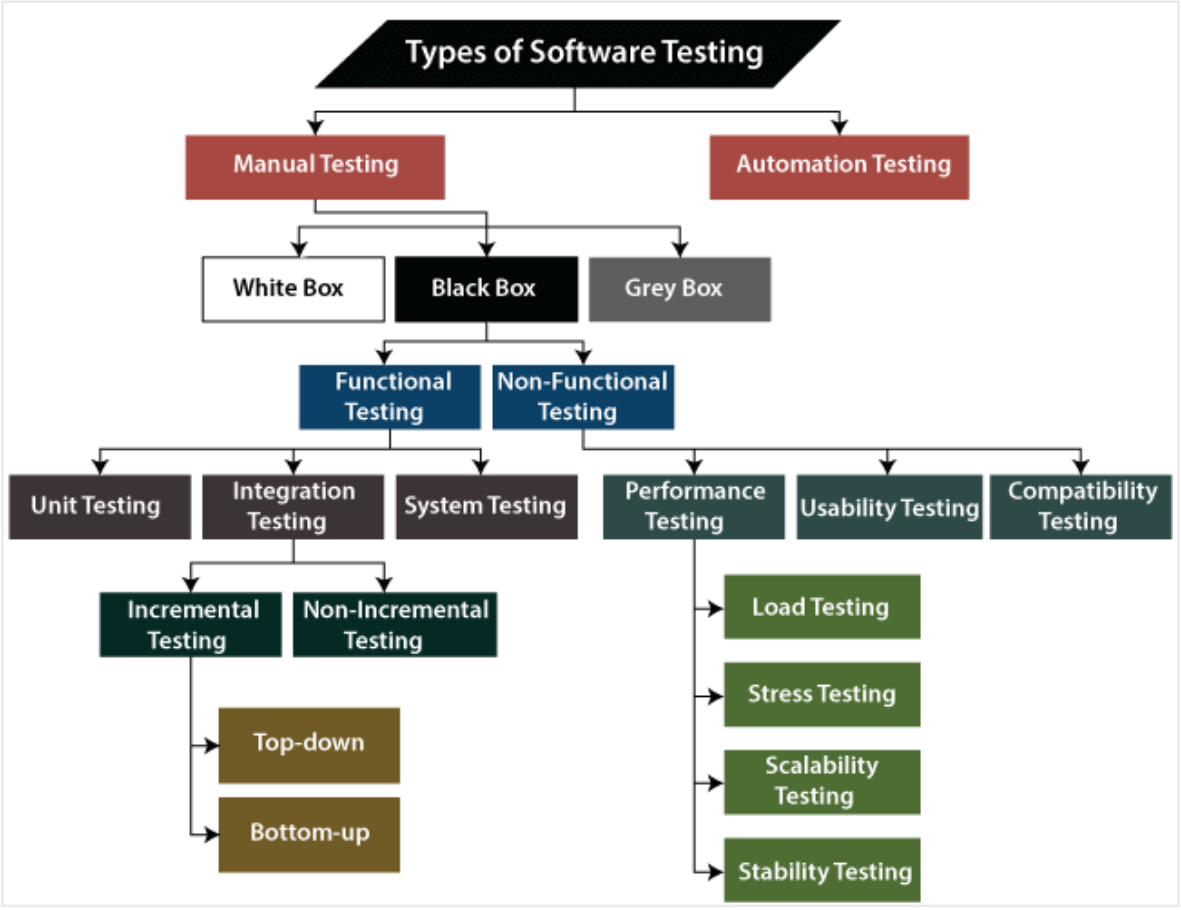
Types of Software Testing

In this section, we are going to understand the various types of software testing, which can be used at the time of the Software Development Life Cycle.

As we know, **software testing** is a process of analyzing an application's functionality as per the customer prerequisite.

If we want to ensure that our software is bug-free or stable, we must perform the various types of software testing because testing is the only method that makes our application bug-free.

.



The different types of Software Testing

The categorization of software testing is a part of diverse testing activities, such as **test strategy, test deliverables, a defined test objective, etc**. And software testing is the execution of the software to find defects.

The purpose of having a testing type is to confirm the **AUT** (Application Under Test).

To start testing, we should have a **requirement, application-ready, necessary resources available**. To maintain accountability, we should assign a respective module to different test engineers.

Manual Testing

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

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

handed it to the tester for retesting.

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

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

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.

Types of Manual Testing

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

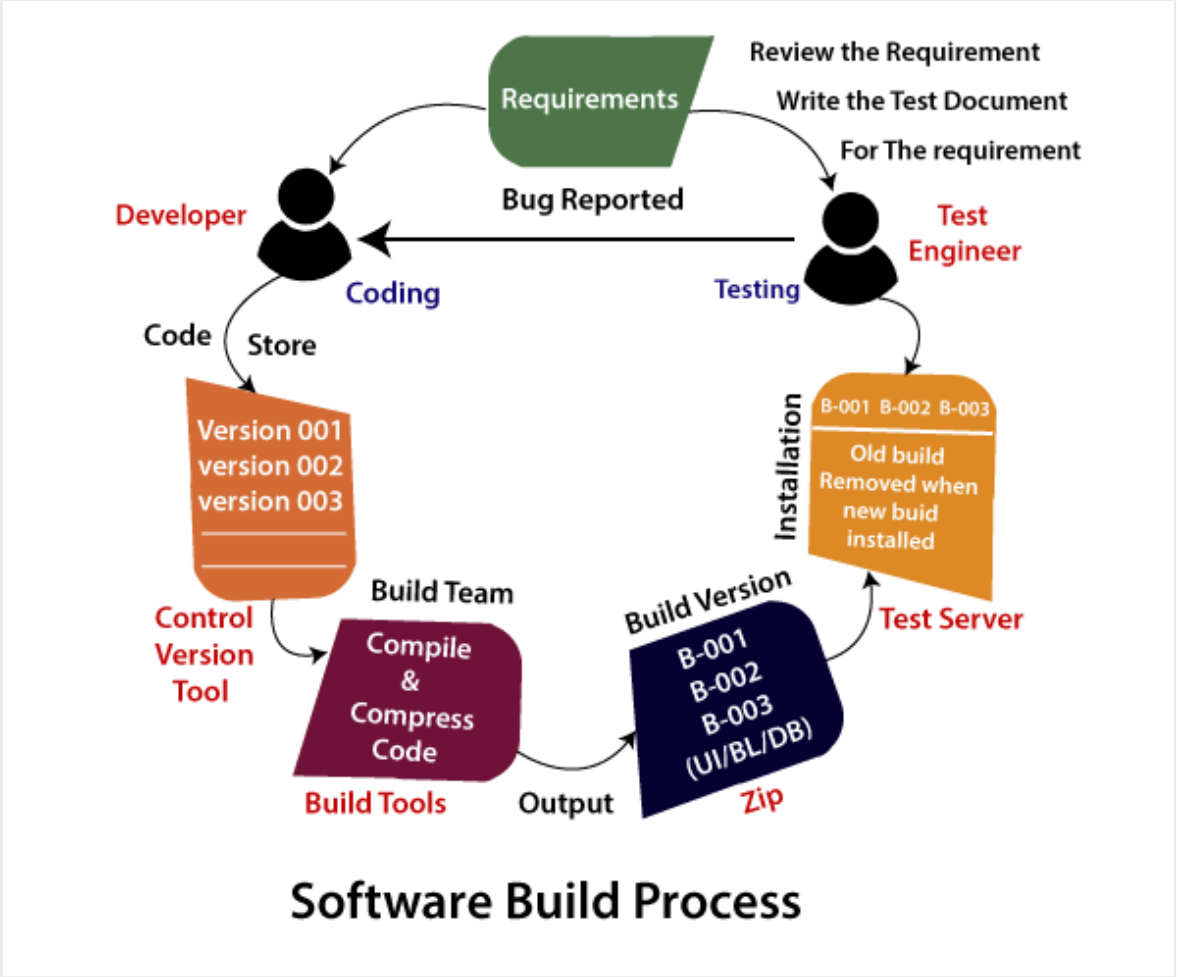
o White Box Testing
o Black Box Testing
o Gray Box Testing

How to perform Manual Testing

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

Software Build Process

o Once the requirement is collected, it will provide to the two different team development and testing team.
o After getting the requirement, the concerned developer will start writing the code.
o And in the meantime, the test engineer understands the requirement and prepares the required documents, up to now the developer may complete the code and store in the **Control Version tool**.
o After that, the code changes in the UI, and these changes handle by one separate team, which is known as the **build team**.
o This build team will take the code and start compile and compress the code with the help of a build tool. Once we got some output, the output goes in the zip file, which is known as **Build** (application or software).Each Build will have some unique number like (B001, B002).
o Then this particular Build will be installed in the test server. After that, the test engineer will access this test server with the help of the Test URL and start testing the application.
o If the test engineer found any bug, he/she will be reported to the concerned developer.
o Then the developer will reproduce the bug in the test server and fix the bug and again store the code in the Control version tool, and it will install the new updated file and remove the old file; this process is continued until we get the stable Build.
o Once we got the stable Build, it will be handed over to the customer.



Note1

o Once we collect the file from the Control version tool, we will use the build tool to compile the code from high-level language to machine level language. After compilation, if the file size will increase, so we will compress that particular file and dumped into the test server.

o This process is done by **Build team**, **developer** (if build team is not there, a developer can do it) or the **test lead** (if the build team directly handle the zip and install the application to the test server and inform the test engineer).

o Generally, we can't get a new Build for every bug; else, most of the time will be wasted only in creating the builds.

Note2

Build team

The main job of the build team is to create the application or the Build and converting the high-level language into low-level language.

Build

It is software, which is used to convert the code into application format. And it consists of some set of features and bug fixes that are handed over to the test engineer for testing purposes until it becomes

stable.

Control version tool

It is a software or application, which is used for the following purpose:

o In this tool, we can save different types of files.

o It is always secured because we access the file from the tools using the same login credentials.

o The primary objective of the tools is to track the changes done for the existing files.

Advantages of Manual Testing

o It does not require programming knowledge while using the Black box method.

o It is used to test dynamically changing GUI designs.

o Tester interacts with software as a real user so that they are able to discover usability and user interface issues.

o It ensures that the software is a hundred percent bug-free.

o It is cost-effective.

o Easy to learn for new testers.

Disadvantages of Manual Testing

o It requires a large number of human resources.

o It is very time-consuming.

o Tester develops test cases based on their skills and experience. There is no evidence that they have covered all functions or not.

o Test cases cannot be used again. Need to develop separate test cases for each new software.

o It does not provide testing on all aspects of testing.

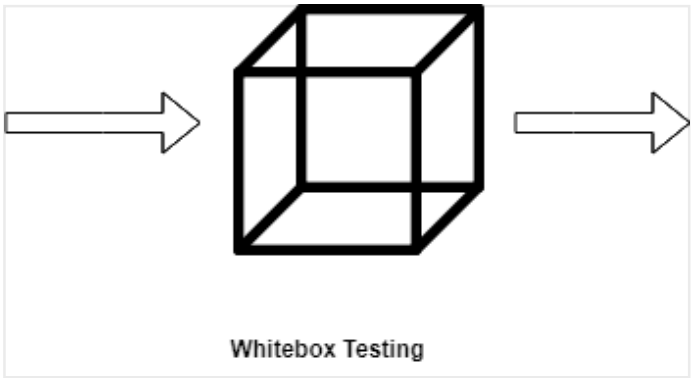
o Since two teams work together, sometimes it is difficult to understand each other's motives, it can mislead the process.

White Box Testing

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

The term 'white box' is used because of the internal perspective of the system. The clear box or white box or transparent box name

denote the ability to see through the software's outer shell into its inner workings.
Developers do white box testing. In this, the developer will test every line of the code of the program. The developers perform the White-box testing and then send the application or the software to the testing team, where they will perform the black box testing and verify the application along with the requirements and identify the bugs and sends it to the developer.
The developer fixes the bugs and does one round of white box testing and sends it to the testing team. Here, fixing the bugs implies that the bug is deleted, and the particular feature is working fine on the application.69.4M
Here, the test engineers will not include in fixing the defects for the following reasons:
o Fixing the bug might interrupt the other features. Therefore, the test engineer should always find the bugs, and developers should still be doing the bug fixes.
o If the test engineers spend most of the time fixing the defects, then they may be unable to find the other bugs in the application.
The white box testing contains various tests, which are as follows:
o Path testing
o Loop testing
o Condition testing
o Testing based on the memory perspective
o Test performance of the program



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

Generic steps of white box testing

o Design all test scenarios, test cases and prioritize them according to high priority number.
o This step involves the study of code at runtime to examine the resource utilization, not accessed areas of the code, time taken by

various methods and operations and so on.
o In this step testing of internal subroutines takes place. Internal subroutines such as nonpublic methods, interfaces are able to handle all types of data appropriately or not.
o This step focuses on testing of control statements like loops and conditional statements to check the efficiency and accuracy for different data inputs.
o In the last step white box testing includes security testing to check all possible security loopholes by looking at how the code handles security.

Reasons for white box testing

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

Advantages of White box testing

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

Disadvantages of White box testing

o White box testing is too much time consuming when it comes to large-scale programming applications.
o White box testing is much expensive and complex.
o It can lead to production error because it is not detailed by the developers.
o White box testing needs professional programmers who have a detailed knowledge and understanding of programming language and implementation.

Techniques Used in White Box Testing

Data Flow Testing	Data flow testing is a group of testing strategies that examines the control flow of programs in order to explore the sequence of variables according to the sequence of events.

Control Flow Testing	Control flow testing determines the execution order of statements or instructions of the program through a control structure. The control structure of a program is used to develop a test case for the program. In this technique, a particular part of a large program is selected by the tester to set the testing path. Test cases represented by the control graph of the program.
Branch Testing	Branch coverage technique is used to cover all branches of the control flow graph. It covers all the possible outcomes (true and false) of each condition of decision point at least once.
Statement Testing	Statement coverage technique is used to design white box test cases. This technique involves execution of all statements of the source code at least once. It is used to calculate the total number of executed statements in the source code, out of total statements present in the source code.

Decision Testing	This technique reports true and false outcomes of Boolean expressions. Whenever there is a possibility of two or more outcomes from the statements like do while statement, if statement and case statement (Control flow statements), it is considered as decision point because there are two outcomes either true or false.

Difference between white-box testing and black-box testing

Following are the significant differences between white box testing and black box testing:

White-box testing	Black box testing
The developers can perform white box testing.	The test engineers perform the black box testing.
To perform WBT, we should have an understanding of the programming languages.	To perform BBT, there is no need to have an understanding of the programming languages.
In this, we will look into the source code and test the logic of the code.	In this, we will verify the functionality of the application based on the requirement specification.
In this, the developer should know about the internal design of the code.	In this, there is no need to know about the internal design of the code.

Black box testing

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

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

it is given back to the development team for correction.

Generic steps of black box testing

o The black box test is based on the specification of requirements, so it is examined in the beginning.

o In the second step, the tester creates a positive test scenario and an adverse test scenario by selecting valid and invalid input values to check that the software is processing them correctly or incorrectly.

o In the third step, the tester develops various test cases such as decision table, all pairs test, equivalent division, error estimation, cause-effect graph, etc.

o The fourth phase includes the execution of all test cases.

o In the fifth step, the tester compares the expected output against the actual output.

o In the sixth and final step, if there is any flaw in the software, then it is cured and tested again.

Test procedure

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

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

Play Videox

Test cases

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

Techniques Used in Black Box Testing

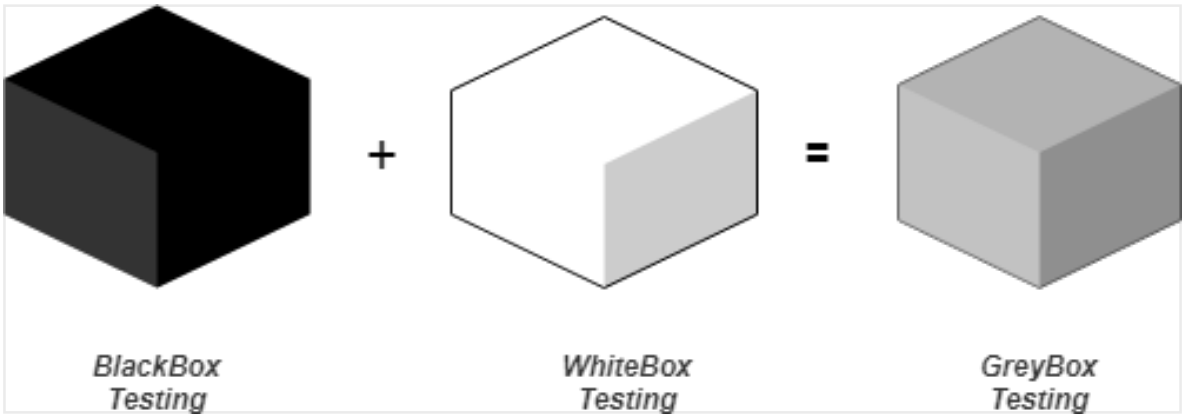
Decision Table Technique	Decision Table Technique is a systematic approach where various input combinations and their respective system behavior are captured in a tabular form. It is appropriate for the functions that have a logical relationship between two and more than two inputs.

Boundary Value Technique	Boundary Value Technique is used to test boundary values, boundary values are those that contain the upper and lower limit of a variable. It tests, while entering boundary value whether the software is producing correct output or not.
State Transition Technique	State Transition Technique is used to capture the behavior of the software application when different input values are given to the same function. This applies to those types of applications that provide the specific number of attempts to access the application.
All-pair Testing Technique	All-pair testing Technique is used to test all the possible discrete combinations of values. This combinational method is used for testing the application that uses checkbox input, radio button input, list box, text box, etc.
Cause-Effect Technique	Cause-Effect Technique underlines the relationship between a given result and all the factors affecting the result.It is based on a collection of requirements.

Equivalence Partitioning Technique	Equivalence partitioning is a technique of software testing in which input data divided into partitions of valid and invalid values, and it is mandatory that all partitions must exhibit the same behavior.
Error Guessing Technique	Error guessing is a technique in which there is no specific method for identifying the error. It is based on the experience of the test analyst, where the tester uses the experience to guess the problematic areas of the software.
Use Case Technique	Use case Technique used to identify the test cases from the beginning to the end of the system as per the usage of the system. By using this technique, the test team creates a test scenario that can exercise the entire software based on the functionality of each function from start to end.

GreyBox Testing

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



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

Why GreyBox testing?

Reasons for GreyBox testing are as follows

o It provides combined benefits of both Blackbox testing and WhiteBox testing.

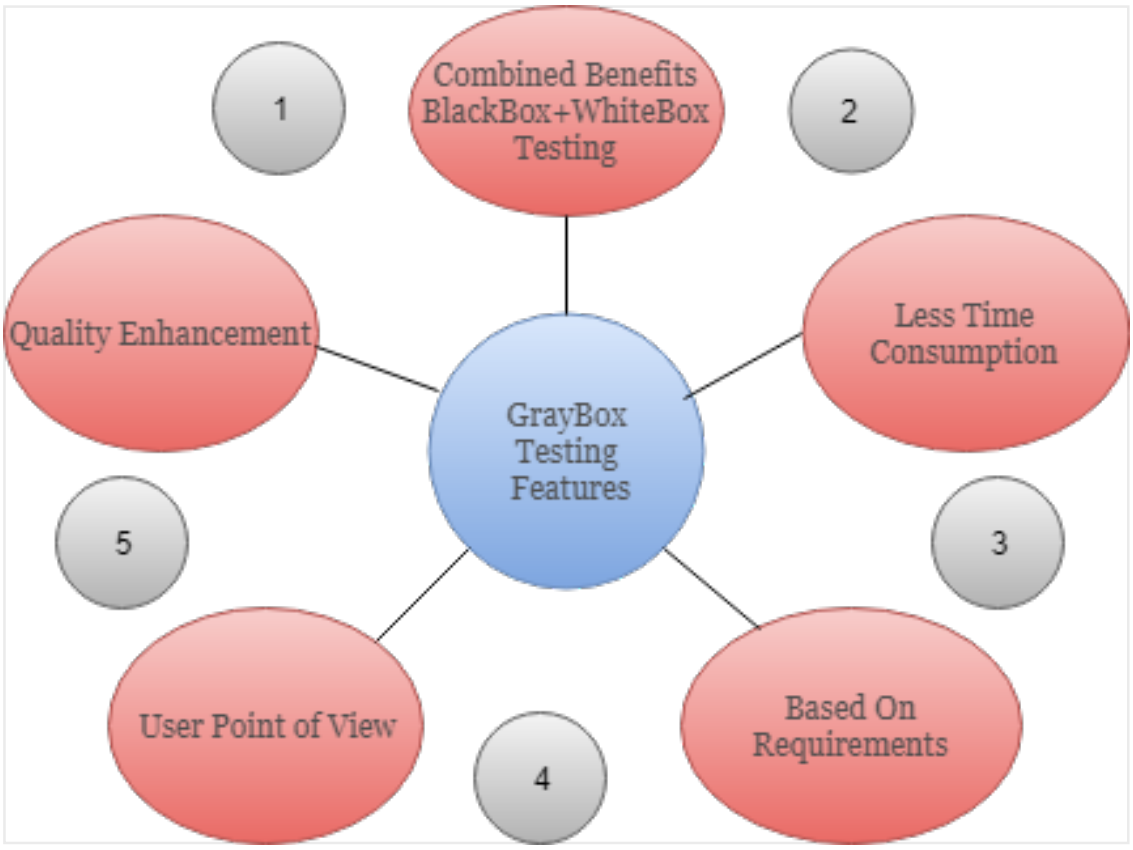
o It includes the input values of both developers and testers at the same time to improve the overall quality of the product.

o It reduces time consumption of long process of functional and non-functional testing.

o It gives sufficient time to the developer to fix the product defects.

o It includes user point of view rather than designer or tester point of view.

o It involves examination of requirements and determination of specifications by user point of view deeply.



GreyBox Testing Strategy

Grey box testing does not make necessary that the tester must design test cases from source code. To perform this testing test cases can be designed on the base of, knowledge of architectures, algorithm, internal states or other high -level descriptions of the program behavior. It uses all the straightforward techniques of black box testing for function testing. The test case generation is based on requirements and preset all the conditions before testing the program by assertion method.

Generic Steps to perform Grey box Testing are:

1. First, select and identify inputs from BlackBox and WhiteBox testing inputs.
2. Second, Identify expected outputs from these selected inputs.
3. Third, identify all the major paths to traverse through during the testing period.
4. The fourth task is to identify sub-functions which are the part of main functions to perform deep level testing.
5. The fifth task is to identify inputs for subfunctions.
6. The sixth task is to identify expected outputs for subfunctions.
7. The seventh task includes executing a test case for Subfunctions.
8. The eighth task includes verification of the correctness of result.

The test cases designed for Greybox testing includes Security related, Browser related, GUI related, Operational system related and

Database related testing.

Techniques of Grey box Testing

Matrix Testing

This testing technique comes under Grey Box testing. It defines all the used variables of a particular program. In any program, variable are the elements through which values can travel inside the program. It should be as per requirement otherwise, it will reduce the readability of the program and speed of the software. Matrix technique is a method to remove unused and uninitialized variables by identifying used variables from the program.

Regression Testing

Regression testing is used to verify that modification in any part of software has not caused any adverse or unintended side effect in any other part of the software. During confirmation testing, any defect got fixed, and that part of software started working as intended, but there might be a possibility that fixed defect may have introduced a different defect somewhere else in the software. So, regression testing takes care of these type of defects by testing strategies like retest risky use cases, retest within a firewall, retest all, etc.

Pattern Testing

Pattern testing is applicable to such type of software that is developed by following the same pattern of previous software. In these type of software possibility to occur the same type of defects. Pattern testing determines reasons of the failure so they can be fixed in the next software.

Usually, automated software testing tools are used in Greybox methodology to conduct the test process. Stubs and module drivers provided to a tester to relieve from manually code generation.

Black Box Testing vs. White Box Testing vs. Grey Box Testing

Index	Black Box Testing	White Box Testing	Grey Box Testing

1	Knowledge of internal working structure (Code) is not required for this type of testing. Only GUI (Graphical User Interface) is required for test cases.	Knowledge of internal working structure (Coding of software) is necessarily required for this type of testing.	Partially Knowledge of the internal working structure is required.
2	Black Box Testing is also known as functional testing, data-driven testing, and closed box testing.	White Box Testing is also known as structural testing, clear box testing, code-based testing, and transparent testing.	Grey Box Testing is also known as translucent testing as the tester has limited knowledge of coding.

3	The approach towards testing includes trial techniques and error guessing method because tester does not need knowledge of internal coding of the software.	White Box Testing is proceeded by verifying the system boundaries and data domains inherent in the software as there is no lack of internal coding knowledge.	If the tester has knowledge of coding, then it is proceeded by validating data domains and internal system boundaries of the software.
4	The testing space of tables for inputs (inputs to be used for creating test cases) is pretty huge and largest among all testing spaces.	The testing space of tables for inputs (inputs to be used for creating test cases) is less as compared to Black Box testing.	The testing space of tables for inputs (inputs to be used for creating test cases) is smaller than Black Box and White Box testing.

5	It is very difficult to discover hidden errors of the software because errors can be due to internal working which is unknown for Black Box testing.	It is simple to discover hidden errors because it can be due to internal working which is deeply explored in White Box testing.	Difficult to discover the hidden error. Might be found in user level testing.
6	It is not considered for algorithm testing.	It is well suitable and recommended for algorithm testing.	It is not considered for algorithm testing.
7	Time consumption in Black Box testing depends upon the availability of the functional specifications.	White Box testing takes a long time to design test cases due to lengthy code.	Test cases designing can be done in a short time period.
8	Tester, developer and the end user can be the part of testing.	Only tester and developer can be a part of testing; the end user can not involve.	Tester, developer and the end user can be the part of testing.

9	It is the least time-consuming process among all the testing processes.	The entire testing process is the most time consuming among all the testing processes.	less time consuming than White Box testing.
10	Resilience and security against viral attacks are covered under Black Box testing.	Resilience and security against viral attacks are not covered under White Box testing.	Resilience and security against viral attacks are not covered under Grey Box testing.
11	The base of this testing is external expectations internal behavior is unknown.	The base of this testing is coding which is responsible for internal working.	Testing based on high-level database diagrams and dataflow diagrams.
12	It is less exhaustive than White Box and Grey Box testing methods.	It is most exhaustive between Black Box and Grey Box testing methods.	Partly exhaustive; depends upon the type of test cases are coding based or GUI based.