ECHLEON INSTITUTE OF TECHNOLOGY

UNIT 1

Introduction to Debugging

Debugging is the technique of monitoring a software code and finding and removing any potential errors, bugs, or defects that can cause the application to behave unexpectedly or crash. Debugging plays a very crucial role in the software development life cycle.

Debugging involves multiple steps, like identifying the source of the bug, correcting the problem, or fixing the bug. This sounds simple enough. However, debugging really becomes a nightmare when the various subsystems or modules are tightly coupled. Sometimes, it takes more time to debug or fix bugs than it takes to do the actual coding. To simplify the process, developers can locate the bug and remove them using various tools and techniques.

To resolve the defect in the program or application, the developer has to identify the piece of source code that is causing the problem. Various debugging tools can be used depending on the stage of development. Developers can trace the program execution step by step by evaluating the values of the variables and the return values of the functions.

The types of errors and the debugging techniques vary depending on the stage of development and the environment.

Debugging :
Debugging is the process of finding and resolving defects or problems within a computer program that prevent correct operation of computer software or a system.
Need for debugging:
Once errors are known during a program code, it's necessary to initial establish the precise program statements liable for the errors and so to repair them.

Challenges in Debugging:
There are lot of problems at the same time as acting the debugging. These are the following:

- Debugging is finished through the individual that evolved the software program and it's miles difficult for that person to acknowledge that an error was made.
- Debugging is typically performed under a tremendous amount of pressure to fix the supported error as quick as possible.
- It can be difficult to accurately reproduce input conditions.
- Compared to the alternative software program improvement activities, relatively little research, literature and formal preparation exist at the procedure of debugging.

Debugging Approaches:
The following are a number of approaches popularly adopted by programmers for debugging.

- **Brute Force Method:**
  This is the foremost common technique of debugging however is that the least economical method. during this approach, the program is loaded with print statements to print the intermediate values with the hope that a number of the written values can facilitate to spot the statement in error. This approach becomes a lot of systematic with the utilisation of a symbolic program (also known as a source code debugger), as a result of values of various variables will be simply checked and breakpoints and watch-points can be easily set to check the values of variables effortlessly.

- **Backtracking:**
  This is additionally a reasonably common approach. during this approach, starting from the statement at which an error symptom has been discovered, the source code is derived backward till the error is discovered. sadly, because the variety of supply lines to be derived back will increase, the quantity of potential backward methods will increase and should become unimaginably large so limiting the utilisation of this approach.

- **Cause Elimination Method:**
  In this approach, a listing of causes that may presumably have contributed to the error symptom is developed and tests are conducted to eliminate every error. A connected technique of identification of the error from the error symptom is that the package fault tree analysis.

- **Program Slicing:**
  This technique is analogous to backtracking. Here the search house is reduced by process slices. A slice of a program for a specific variable at a particular statement is that the set of supply lines preceding this statement which will influence the worth of that variable

Debugging Guidelines:
Debugging is commonly administrated by programmers supported their ingenuity. The subsequent are some general tips for effective debugging:

- Many times debugging needs an intensive understanding of the program style. making an attempt to rectify supported a partial understanding of the system style and implementation might need an excessive quantity of effort to be placed into debugging even straightforward issues.

- Debugging might generally even need a full plan of the system. In such cases, a typical mistake that novice programmers usually create is trying to not fix the error however its symptoms.

- One should be watched out for the likelihood that a slip correction might introduce new errors. so when each spherical of error-fixing, regression testing should be administrated.

Types of Possible Errors

To understand different types of debugging, we need to understand the different types of errors. Errors are usually of three types:

- Build and compile-time errors
- Runtime errors
- Logic errors

Build and compile-time errors happen at the development stage when the code is being built. These errors are thrown by the compiler or the interpreter while building the source code. In other words, build or compile-time errors prevent the application from even starting. These errors often result from syntax errors, like missing semicolons at the end of a statement or class not found. These errors are easy to spot and rectify because most IDE or compilers find them for you. The compiler or the interpreter will tell you the exact piece of code that is causing the problem.

Runtime errors occur and can be identified only while running the application. They occur only when the source code doesn't have any compiler or syntax error, and the compiler or the interpreter cannot identify the runtime error during the build stage. Mostly, runtime errors depend on the user input or the environment. These kinds of errors can be identified by using try-catch blocks in your program and logging the error message properly.

Logic errors occur after the program is successfully compiled and running and it gives you an output. A logic error is when the result of the program is incorrect. These errors can not be caught using the try-catch blocks. Logic errors are also called semantic errors, and they occur due to some incorrect logic used by the developer to solve a problem while building the application.

Types of Debugging

Not all errors can be treated or debugged in the same way. The developer has to set up the right strategy to fix different errors. There are two types of debugging techniques: reactive debugging and preemptive debugging.

Most debugging is reactive—a defect is reported in the application or an error occurs, and the developer tries to find the root cause of the error to fix it. Solving build errors is easier, and generally, the compiler or build system clearly tells you the errors.

Solving a runtime error can be done using a debugger, which can provide additional information about the error and the stack trace of the error. It will tell you exactly the line where the fault is happening or exception is raised.

Solving logic errors is trickier. We don't get clues as we do in other errors from the compiler or the stack trace. We need to use other strategies to identify the code causing the error.

Reactive Debugging

Reactive debugging refers to any debugging protocol that is employed after the bug manifests itself. Reactive debugging is deployed to reduce runtime and logic errors. Examples of reactive debugging are print debugging and using a debugger.

Print debugging

Print debugging is the print statements developers write in their source code (printf in C or System.Out.print in Java or print in Python). Developers write them to print out the values of the variables to the console or to the file to get a better sense of what is happening. These printed statements help find the root cause of the problem, especially when a variable has unexpected values.

Let's say, for example, the source code has three functions, foo, bar, and baz. The source code is printing wrong values in baz, and baz is using foo and bar functions. To find the root cause, developers can add the print statement in the foo return values first, then bar, and then baz. The developer will identify the function causing the problem by narrowing down the search. Once we find the function returning the wrong value, we can analyze the variables or logic inside it.

Debugger

Print debugging is an easy technique, and sometimes developers use it to find the root cause of an error. However, it is a time-consuming and tedious process if the developer is dealing with several modules, functions, and variables. If a developer finds themself writing more than 10 print statements to identify the issue, then it's time to switch to a debugger.

The debugger allows the developer to trace the execution of the program and identify the state of the variable functions as the program runs. In a sense, it is not too different from a print debugger. What makes it better for larger applications is that it prints every single variable and line of source code as the program executes. Integrated development environments like Eclipse, Jetbrains IntelliJ, Pycharm, and Visual Studio include built-in GUI debuggers. There are many command-line debuggers like GDB for C and pdb for Python available in the market.

Additionally, sometimes one can face a situation where the application in question runs on a different host. Collecting data from that system thus becomes really challenging. In such cases, remote debuggnging allows developers to trace the issue on a different host. Most modern IDEs support remote debugging. You can, at times, also couple remote debugging with multi-client debugging tools when the deployment architecture is multi-node or micro-service.

Preemptive Debugging

Preemptive debugging involves writing code that doesn't impact the functionality of the program but helps developers, either catch bugs sooner or debug the source code easily when the bug occurs.

Assertions
Most programming languages have the inbuilt ability to add assertions in the source code. An assertion is a binary condition that is false. The assertions will make the execution of the code exit immediately with a helpful message.

Example:

```
assert value >= 0, "Cannot divide with a negative value"
```

Logging
Logging is not very different from print debugging. Logging statements of the variables or the error trace are added to the source code during the development. These logging statements will not affect the functionality of the application but writes the log statement to the file or console or database, etc.

Note, the logging statements are not added in reaction to the errors or bugs in the source code. They are added from the get-go because the information included in the log statements can be useful in identifying which functions are receiving the proper inputs and returning the expected result and which are not. This helps us streamline the search.

```
log.info("persisting the item {} Quantity {}".format(item, quantity))
```

What is software testing?

Software testing is the process of assessing the functionality of a software program. The process checks for errors and gaps and whether the outcome of the application matches desired expectations before the software is installed and goes live. Software testing is a process of executing a program or application with the intent of finding the software bugs.

- It can also be stated as the **process of** validating **and** verifying that a software program or application or product:
    - Meets the business and technical requirements that guided it's design and development
    - Works as expected
    - Can be implemented with the same characteristic.

Why is software testing important?

Software testing is the culmination of application development through which software testers evaluate code by questioning it. This evaluation can be brief or proceed until all stakeholders are satisfied. Software testing identifies bugs and issues in the development process so they're fixed prior to product launch. Software Testing is necessary because we all make mistakes. Some of those mistakes are unimportant, but some of them are expensive or dangerous. We need to check everything and anything we produce because things can always go wrong – humans make mistakes all the time.

Since we assume that our work may have mistakes, hence we all need to check our own work. However some mistakes come from bad assumptions and blind spots, so we might make the same mistakes when we check our own work as we made when we did it. So we may not notice the flaws in what we have done.
Ideally, we should get someone else to check our work because another person is more likely to spot the flaws.
There are several reasons which clearly tells us as why Software Testing is important and what are the major things that we should consider while testing of any product or application.

The following are important reasons why software testing techniques should be incorporated into application development:

- **Identifies defects early.** Developing complex applications can leave room for errors. Software testing is imperative, as it identifies any issues and defects with the written code so they can be fixed before the software product is delivered.
- **Improves product quality.** When it comes to customer appeal, delivering a quality product is an important metric to consider. An exceptional product can only be delivered if it's tested effectively before launch. Software testing helps the product pass quality assurance (QA) and meet the criteria and specifications defined by the users.
- **Increases customer trust and satisfaction.** Testing a product throughout its development lifecycle builds customer trust and satisfaction, as it provides visibility into the product's strong and weak points. By the time customers receive the product, it has been tried and tested multiple times and delivers on quality.
- **Detects security vulnerabilities.** Insecure application code can leave vulnerabilities that attackers can exploit. Since most applications are online today, they can be a leading vector for cyber attacks and should be tested thoroughly during various stages of application development. For example, a web application published without proper software testing can easily fall victim to a cross-site scripting attack where the attackers try to inject malicious code into the user's web browser by gaining access through the vulnerable web application. The nontested application thus becomes the vehicle for delivering the malicious code, which could have been prevented with proper software testing.
- **Helps with scalability.** A type of nonfunctional software testing process, scalability testing is done to gauge how well an application scales with increasing workloads, such as user traffic, data volume and transaction counts. It can also identify the point where an application might stop functioning and the reasons behind it, which may include meeting or exceeding a certain threshold, such as the total number of concurrent app users.

- **Saves money.** Software development issues that go unnoticed due to a lack of software testing can haunt organizations later with a bigger price tag. After the application launches, it can be more difficult to trace and resolve the issues, as software patching is generally more expensive than testing during the development stages.

Types of software testing

There are many types of software testing, but the two main categories are dynamic testing and static testing. Dynamic testing is an assessment that's conducted while the program is executed; static testing examines the program's code and associated documentation. Dynamic and static methods are often used together.

The following are the main types of software testing methodologies:

- **Integration testing.** This groups together two or more modules of an application to ensure they function collectively. This type of testing also reveals interface, communication and data flow defects between modules.
- **Unit testing.** Typically conducted during the application development phase, the purpose of unit testing is to ensure that each individual unit or component performs as expected. This is a type of white box testing and test automation tools -- such as NUnit, JUnit and xUnit -- are typically used to execute these tests.
- **Functional testing.** This entails checking functions against functional requirements. A common way to conduct functional testing is by using the black box testing
- **Security testing.** This ensures the software is free of potential vulnerabilities, known flaws and security loopholes that might affect the user system and data. Security testing is generally conducted through penetration testing.
- **Performance testing.** This tests the performance and speed of an application under a given workload.
- **Regression testing.** This verifies whether adding new features causes a decline in the functionality of an application.
- **Stress testing.** This assesses the strength of software by testing how much load it can take before reaching a breaking point. This is a type of nonfunctional test.
- **Acceptance testing.** This evaluates the entire system against the desired requirements and ensures the project is complete.

WHAT ARE SOFTWARE TESTING OBJECTIVES AND PURPOSE?

Software Testing has different goals and objectives.The major objectives of Software testing are as follows:

- Finding defects which may get created by the programmer while developing the software.
- Gaining confidence in and providing information about the level of quality.
- To prevent defects.
- To make sure that the end result meets the business and user requirements.
- To ensure that it satisfies the BRS that is Business Requirement Specification and SRS that is System Requirement Specifications.
- To gain the confidence of the customers by providing them a quality product.

Software testing helps in finalizing the software application or product against business and user requirements. It is very important to have good test coverage in order to test the software application completely and make it sure that it’s performing well and as per the specifications.

While determining the test coverage the test cases should be designed well with maximum possibilities of finding the errors or bugs. The test cases should be very effective. This objective can be measured by the number of defects reported per test cases. Higher the number of the defects reported the more effective are the test cases.
Once the delivery is made to the end users or the customers they should be able to operate it without any complaints. In order to make this happen the tester should know as how the customers are going to use this product and accordingly they should write down the test scenarios and design the test cases. This will help a lot in fulfilling all the customer’s requirements.
Software testing makes sure that the testing is being done properly and hence the system is ready for use. Good coverage means that the testing has been done to cover the various areas like functionality of the application, compatibility of the application with the OS, hardware and different types of browsers, performance testing to test the performance of the application and load testing to make sure that the system is reliable and should not crash or there should not be any blocking issues. It also determines that the application can be deployed easily to the machine and without any resistance. Hence the application is easy to install, learn and use.

WHAT ARE THE GOALS OF SOFTWARE TESTING?

Software testing ensures that a program is operating in accordance with its specifications. The QA team uses manual testing and automation testing to assess an application.

Testing is a crucial aspect of the software development process. The application will not be released into the public domain unless this is done.

Both methods of testing ensure that a program is stable, engaging, and user-friendly.

Testing, on the other hand, is a difficult task because every user has a different perspective and expectation. Also, we understand that not everyone will appreciate your product, but we should do everything we can to make it the best it can be.

As a result, we make your application user-friendly through quality testing so that the largest number of people may enjoy it.

AIM OF SOFTWARE TESTING:-

Identification of Bugs and Errors

The tester begins testing after the developer has finished coding. QA validates each module under multiple scenarios throughout testing.

They then gather all the mistakes and bugs and provide them to the developer to be fixed.

Product of High Quality

The fundamental goal of testing is to keep the product's quality high. Furthermore, testing has its own cycle, with each phase focusing solely on quality.

Justification in the Form of a Requirement

The QA team verifies if the program adheres to the SRS (System Requirement Specification) document during testing.

Provides Self-Assuredness

The software's features are continually checked by the testing team. It must meet the needs of the business while also instilling confidence.

Enhances Business Growth

A high-quality delivery boosts a company's potential. It is integral to understand that we cannot compromise on quality, which can only be achieved through software testing.

You must have seen the need for testing after reading these objectives. Additionally, it will be a fantastic decision if you want to pursue a job in this industry.

What is the Psychology of testing?

- The comparison of the mindset of the tester and the developer.
- The balance between self-testing and independent testing.
- There should be clear and courteous communication and feedback on defects between tester and developer.

Comparison of the mindset of the tester and developer:
The testing and reviewing of the applications are different from the analysing and developing of it. By this we mean to say that if we are building or developing applications we are working positively to solve the problems during the development process and to make the product according to the user specification.
However while testing or reviewing a product we are looking for the defects or failures in the product. Thus building the software requires a different mindset from testing the software.
The balance between self-testing and independent testing:
The comparison made on the mindset of the tester and the developer in the above article is just to compare the two different perspectives.
It does not mean that the tester cannot be the programmer, or that the programmer cannot be the tester, although they often are separate roles. In fact programmers are the testers.
They always test their component which they built. While testing their own code they find many problems so the programmers, architect and the developers always test their own code before giving it to anyone. However we all know that it is difficult to find our own mistakes.
So, programmers, architect, business analyst depend on others to help test their work. This other person might be some other developer from the same team or the Testing specialists or professional testers.
Giving applications to the testing specialists or professional testers allows an independent test of the system.

This degree of independence avoids author bias and is often more effective at finding defects and failures.
There is several level of independence in software testing which is listed here from the lowest level of independence to the highest:
i. Tests by the person who wrote the item.
ii. Tests by another person within the same team, like another programmer.
iii. Tests by the person from some different group such as an independent test team.
iv. Tests by a person from a different organization or company, such as outsourced testing or certification by an external body.
Clear and courteous communication and feedback on defects between tester and developer:
We all make mistakes and we sometimes get annoyed and upset or depressed when   someone points them out. So, when as testers we run a test which is a good test from our viewpoint because we found the defects and failures in the software.
But at the same time we need to be very careful as how we react or report the defects and failures to the programmers. We are pleased because we found a good bug but how will the requirement analyst, the designer, developer, project manager and customer react.

- The people who build the application may react defensively and take this reported defect as personal criticism.
- The project manager may be annoyed with everyone for holding up the project.
- The customer may lose confidence in the product because he can see defects.

Because testing can be seen as destructive activity we need to take care while reporting our defects and failures as objectively and politely as possible.

What is fundamental test process in software testing?

Testing is a process rather than a single activity. This process starts from test planning then designing **test cases**, preparing for execution and evaluating status till the test closure. So, we can divide the activities within the fundamental test process into the following basic steps:

1) Planning and Control
2) Analysis and Design
3) Implementation and Execution
4) Evaluating exit criteria and Reporting
5) Test Closure activities

1) Planning and Control:

Test planning has following major tasks:
i. To determine the scope and **risks** and identify the objectives of testing.
ii. To determine the test approach.
iii. To implement the test policy and/or the **test strategy**. (Test strategy is an outline that describes the testing portion of the **software development cycle**. It is created to inform PM, testers and developers about some key issues of the testing process. This includes the testing objectives, method of testing, total time and resources required for the project and the testing environments.).
iv. To determine the required test resources like people, test environments, PCs, etc.
v. To schedule test analysis and design tasks, test implementation, execution and evaluation.
vi. To determine the **Exit criteria** we need to set criteria such as **Coverage criteria.** (Coverage criteria are the percentage of statements in the software that must be executed during testing. This will help us track whether we are completing test activities correctly. They will show us which tasks and checks we must complete for a particular level of testing before we can say that testing is finished.)

Test control has the following major tasks:
i. To measure and analyze the results of reviews and testing.
ii. To monitor and document progress, **test coverage** and exit criteria.
iii. To provide information on testing.
iv. To initiate corrective actions.
v. To make decisions.

2) Analysis and Design:

Test analysis and Test Design has the following major tasks:
i. To review the **test basis.** (The test basis is the information we need in order to start the test analysis and create our own test cases. Basically it’s a documentation on which test cases are based, such as requirements, design specifications, product risk analysis, architecture and interfaces. We can use the test basis documents to understand what the system should do once built.)
ii. To identify test conditions.
iii. To design the tests.
iv. To evaluate testability of the requirements and system.
v. To design the test environment set-up and identify and required infrastructure and tools.

3) Implementation and Execution:
During test implementation and execution, we take the test conditions into **test cases** and procedures and other **testware** such as scripts for automation, the test environment and any other test infrastructure. (Test cases is a set of conditions under which a tester will determine whether an application is working correctly or not.)
(Testware is a term for all utilities that serve in combination for testing a software like scripts, the test environment and any other test infrastructure for later reuse.)

Test implementation has the following major task:
i. To develop and prioritize our test cases by using techniques and create **test data** for those tests. (In order to test a software application you need to enter some data for testing most of the features. Any such specifically identified data which is used in tests is known as test data.)
We also write some instructions for carrying out the tests which is known as **test procedures.**
We may also need to automate some tests using **test harness** and automated tests scripts. (A test harness is a collection of software and test data for testing a program unit by running it under different conditions and monitoring its behavior and outputs.)

ii. To create test suites from the test cases for efficient test execution.
(Test suite is a collection of test cases that are used to test a software program to show that it has some specified set of behaviours. A test suite often contains detailed instructions and information for each collection of test cases on the system configuration to be used during testing. Test suites are used to group similar test cases together.)

iii. To implement and verify the environment.

Test execution has the following major task:
i. To execute test suites and individual test cases following the test procedures.
ii. To re-execute the tests that previously failed in order to confirm a fix. This is known as **confirmation testing or re-testing.**
iii. To log the outcome of the test execution and record the identities and versions of the software under tests. The **test log** is used for the audit trial. (A test log is nothing but, what are the test cases that we executed, in what order we executed, who executed that test cases and what is the status of the test case (pass/fail). These descriptions are documented and called as test log.).
iv. To Compare actual results with expected results.
v. Where there are differences between actual and expected results, it report discrepancies as Incidents.

4) Evaluating Exit criteria and Reporting:
Based on the risk assessment of the project we will set the criteria for each test level against which we will measure the “enough testing”. These criteria vary from project to project and are known as **exit criteria**.
Exit criteria come into picture, when:
— Maximum test cases are executed with certain pass percentage.
— Bug rate falls below certain level.
— When achieved the deadlines.

Evaluating exit criteria has the following major tasks:
i. To check the test logs against the exit criteria specified in test planning.
ii. To assess if more test are needed or if the exit criteria specified should be changed.
iii. To write a test summary report for stakeholders.

5) Test Closure activities:
Test closure activities are done when software is delivered. The testing can be closed for the other reasons also like:

- When all the information has been gathered which are needed for the testing.
- When a project is cancelled.
- When some target is achieved.
- When a maintenance release or update is done.

Test closure activities have the following major tasks:
i. To check which planned deliverables are actually delivered and to ensure that all incident reports have been resolved.
ii. To finalize and archive testware such as scripts, test environments, etc. for later reuse.
iii. To handover the testware to the maintenance organization. They will give support to the software.
iv To evaluate how the testing went and learn lessons for future releases and projects.

Software Testing Economics

Cost of Quality:

Cost of quality is a term that is used to quantify the total cost of failure, appraisal, and prevention costs associated with the production of software.

The three categories of costs associated with producing quality products are:

1. **Prevention Costs**: Money required preventing errors and do the job right the first time.

2. **Appraisal Costs**: Money spent to review completed products against requirements. Appraisal includes cost of inspections, testing and reviews.

3. **Failure Costs**: All costs associated with defective products that have been delivered to the user or moved into production.

The concept of "testing economics" in the context of software testing refers to the economic factors involved in software testing activities. It emphasizes that tests must provide more value than they cost, not just to write, but to modify, debug, and run

. Understanding the costs and benefits of testing activities is crucial for maximizing effectiveness and directing effort investment

. The economic impact of software testing is evident in the cost of defects and the importance of early defect detection to save on fixing costs

. Additionally, the price of software testing is commonly lower than the price associated with main faults, making testing an investment in quality

. Therefore, considering the economic aspects of software testing is essential for efficient and cost-effective testing processes.

Economics should determine how and how much you test your software. Your tests must provide more value than they cost, not just to write, but to modify, debug, and run. That means your testing strategy will change over the course of product development – through design, prototyping, construction, iteration, deployment, and maintenance.

For example, much software is written *speculatively*, before it is known whether it will be valuable. The goal at that stage is to figure out its value, not to make it robust, bug free, or maintainable by people other than the authors. Writing and rewriting automated tests as the software goes through rapid changes might not make economic sense. During maintenance, on the other hand, functionality changes slowly and users expect it to keep working reliably. It then usually makes sense to have a large number of automated tests that provide maintainers confidence that the software works as intended after a change.

MODELS FOR TESTING:

Software Testing is one of the parts of the software development life cycle. In our current market, there are so many individual models used in the software development process and there is each approach has its own advantages and disadvantages. so, you should select a specific model based on the project requirements and complications of the project. Let's see different types of software testing models with their benefits and drawbacks.

5 Types of Software Testing Models

Different types of software testing models are as below:

1. Waterfall Model
2. V Model
3. Agile Model
4. Spiral Model
5. Iterative Model

Read the brief about software testing models.

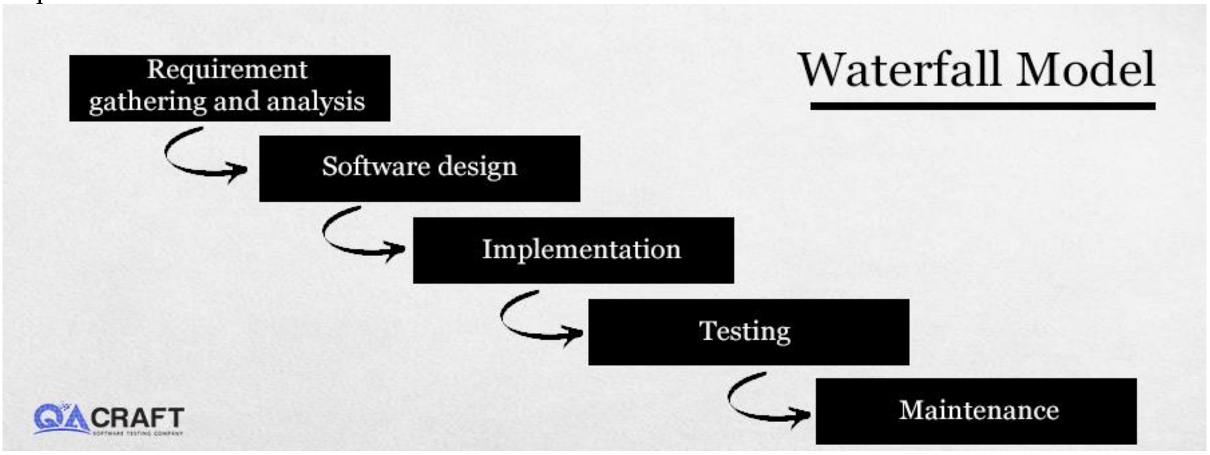
1. Waterfall Model

In "The Waterfall" Model process, the whole process of software development is distributed into separate steps. In this model, generally, the result of one phase works as the input for the next phase in sequence

In the waterfall model basically 4 different phases– requirement gathering and analysis phase, software design, programmed implementation and testing, and maintenance. All 4 phases are coming one by one in the given above order.

The first phase of this model is requirement gathering and analysis, all possible system requirements for developing a particular software are observed and determined. This in turn depends on the software requirement specifications which include detailed information about the wishes of the end user. Based on this a Requirement Specification is.

The document is created which works as an input for the next phase, i.e., the software design phase. here one important thing is that once you move into the next phase it won't be possible to update the requirements. So, you must be very clear and careful about the end-user requirements.



Advantages

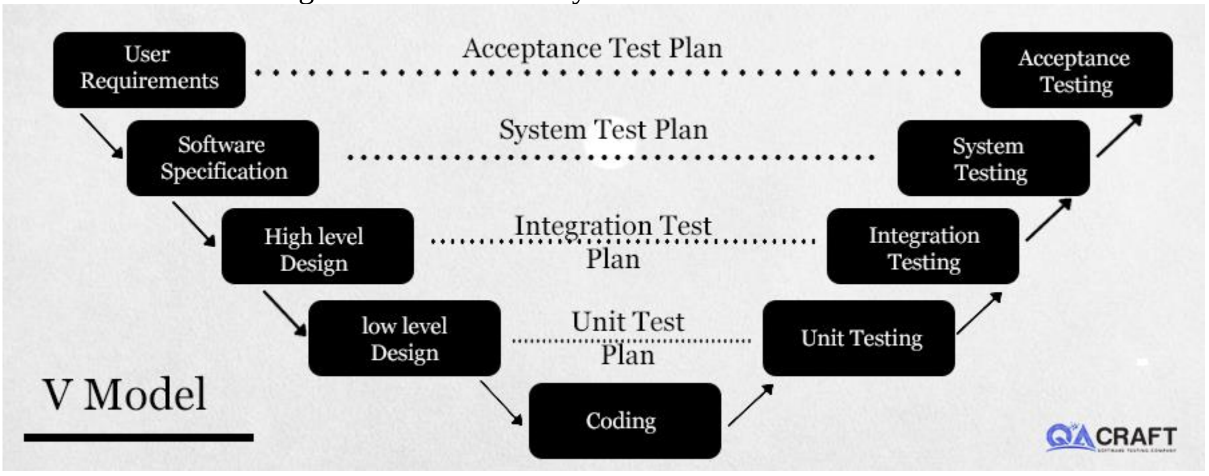
- Easy to implement and maintain.
- The initial phase of requirement gathering and analysis is helpful in saving time later in the development phase
- The resources requirement is minimum and after the completion of each phase, testing is done

Disadvantages

- Here not possible to change or update the requirements.
- You cannot make changes to the previous phase when once you are into the next phase.
- You Cannot start the next phase until the previous phase is completed.

2. V Model

The **V Model** is considered best for the waterfall model. In this model, the development and testing activities are carried out side by side in the downhill and uphill shapes. Here development and testing phases work parallelly. Also, testing begins at the unit level and reaches toward the integration of the entire system.



Advantages

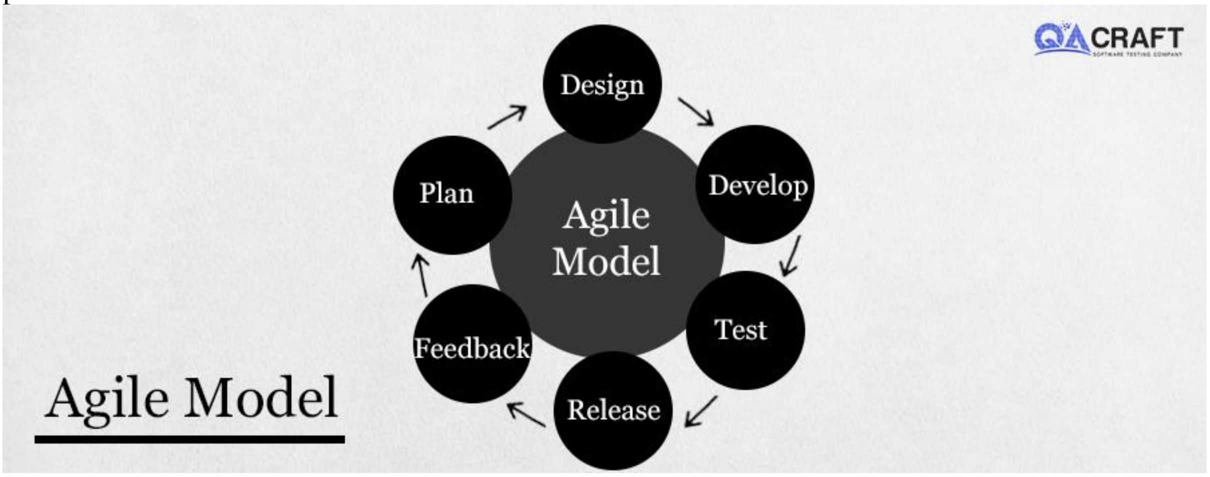
- It is simple to use because testing activities like planning and test designing are already done before coding.
- This model increases the possibilities of success and saves time.
- Bugs are mostly found at an initial stage and the downward flow of bugs is generally avoided.

Disadvantages

- It is a strict model.
- The software is developed during the implementation phase so initial prototypes of the product are not available.
- If there are modifications in the middle, you need to update the test document.

3. Agile Model
In the Agile model, requirements, and solutions are developed by the collaboration between various cross-functional teams. **The agile model** is also known as an iterative and

incremental model. The agile software testing model focuses on process flexibility and customer satisfaction by rapid delivery of working software products and by breaking the product into little incremental builds.



Advantages

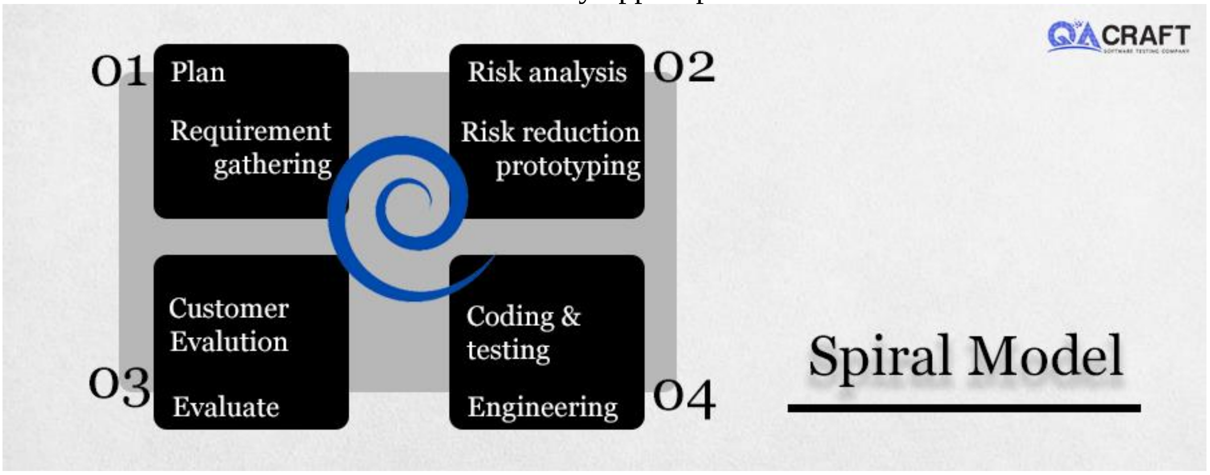
- It makes sure customer satisfaction with the fast and continuous development of deliverables.
- The continuous communication between the customers, developers, and testers makes it a flexible model.
- You can develop the working software rapidly and adapt to modify requirements regularly.

Disadvantages

- It is tricky to assess the effort required at the start of the cycle for large and complicated software development cases.
- Due to continuous communication with the customer, the project can go off track if the customer is not clear about the target.

4. Spiral Model

This software testing model is almost similar to the Agile model but with more attention to risk analysis. The different phases of the spiral model are – planning, risk analysis, engineering, and evaluation. In this case, you need to collect the requirements and perform the risk assessment at the base level and every upper spiral builds on it.



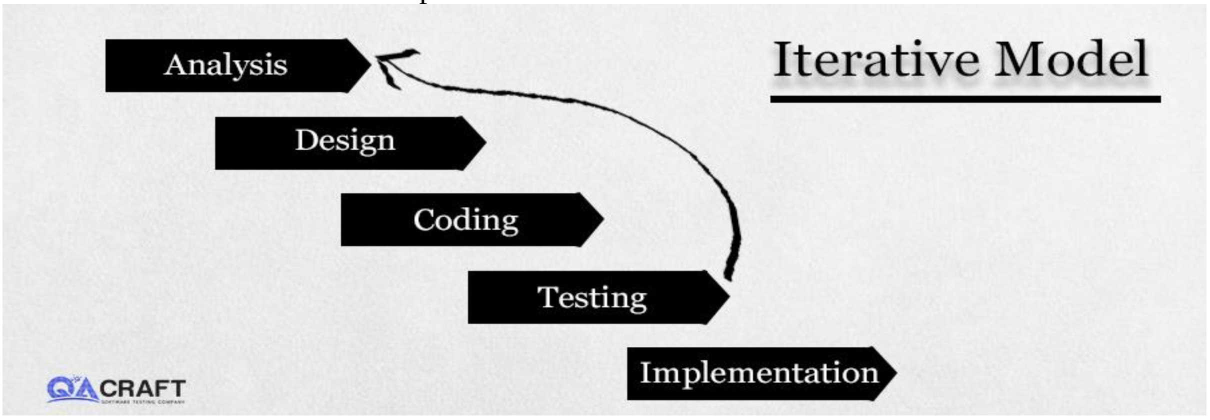
Advantages

- It is suitable for complicated and large systems.
- You can add functionalities depending on the modified circumstances.
- Software is generated early in the cycle.

Disadvantages

- It is a costly model which requires highly specialized expertise in risk analysis
- It does not work well on easy projects.

5. Iterative Model
The Iterative model does not need a whole list of requirements before starting the project. The development process initiates with the requirements of the functional part, which can be extended later. The process is repeated and allows new versions of the product for every cycle. Every iteration includes the development of a divided component of the system which is added to the functional developed earlier.



Advantages

- It is simple to control the risks as high-risk tasks are completed first.
- The improvement is easily measurable.
- Difficulties and risks defined within one iteration can be avoided in the next sprints.

Disadvantages

- The iterative model needs extra resources than the waterfall model.
- The process is tough to manage.
- The risks may not be totally determined even at the last stage of the project.

Effective Software Testing

Effective Software Testing is a hands-on guide to creating bug-free software. Written for developers, it guides you through all the different types of testing, from single units up to entire components. You'll also learn how to engineer code that facilitates testing and how to write easy-to-maintain test code .test effectiveness is a measure of how effectively testing is performed, and test efficiency measures the efficiency of test processes. In software testing, test effectiveness depicts the percentage of bugs detected by the testing team and the overall number of bugs found in the software

Here are 11 ways in which software testing automation can help a company improve the quality assurance of its softwares.

1. Prepare a test plan

'A good beginning is half the battle won.'

It means that half the work is complete when there is a good start to it. Effective test planning entails documentation of test plans and processes for a project.

Software quality management includes different document types supporting quality assurance (QA) processes. A quality management plan, test policy, test strategy, test plan, and test cases are various quality documents that specify objectives and methods that a company can follow to achieve overall good product quality.

2. Report bugs on time

Efficient software testing requires reporting bugs quickly after recognizing them. A good report is essential to enable developers to understand the problem and find a timely solution to prevent cascading effects of the bugs.

A bug report is an efficient means of communication between a quality assurance engineer and developers. A bug report should include step-by-step instructions to reproduce a bug and all relevant information to avoid confusion. It should also consist of screenshots highlighting the defects for greater clarity.

3. Test automation for efficient processes

Software testing automation helps to improve testing efficiency by automating repetitive manual tasks and quick identification of bugs.

It improves test coverage, reduces human error, and optimizes software testing budgets. Regression testing, data-driven testing, testing of complex functionalities, performance testing, and smoke testing are feasible tests for automation.

One cannot automate all tests, and according to a study, test automation services make up for 50% of testing in companies.

A company can employ software testing automation in agile workflows and DevOps methodology with continuous integration and delivery practices. The organization needs to select appropriate QA Automation tools to enable it to automate software testing.

4. Invest in software testing tools

The demand for software testing automation continues to increase with higher adoption of Agile and DevOps methodology. Therefore, a company should invest in software testing tools that are more reliable than human testing to identify bugs.

The testing tool complements the QA personnel's testing efforts and helps the developers improve productivity and accuracy. Some top testing tools are Selenium WebDriver, Appium, Tomcat, etc.

5. Implement exploratory testing

Exploratory testing is a product assessment process without predetermined test cases to gauge the product's performance in real-world conditions.

The technique requires an experienced team, since the test cases are designed and executed instantaneously. The test results are analyzed to identify bugs and take corrective actions immediately.

Exploratory testing is crucial as it involves real-life scenarios from user perspectives.

6. Build a structured testing organization

Software testing involves different activities requiring varied skill sets. The different roles are specified at the planning stage in a testing plan for seamless execution. Software test engineers, test automation engineers, test analysts, test architects, and test managers are the expected roles within software quality assurance.

A test analyst identifies test conditions and features to test, a software test engineer tests the overall system using appropriate methodology, and a test automation engineer develops a script for automated tests.

7. Use shift-left approach for an early start and frequent testing

DevOps methodology with continuous integration and delivery (CI/CD) pipelines has accelerated software development and business innovation. Software testing also needs to evolve to keep pace, which means testing needs to start early and perform continuously.

A shift-left testing approach means testing is broken down into small parts and is done from the beginning of the development process rather than in the end as in the traditional model. Testing activities match the development cycle in frequency throughout the development stages. It also requires test data design and provisioning to keep up with the fast pace of simultaneous development and testing.

8. Conduct formal technical reviews

A formal technical review is a software quality control activity performed by QA and development teams to detect errors in logic, function, or any other software segment.

The objective is to ensure that the company builds the software to meet specific requirements and match predetermined standards.

You can schedule FTR when your product is in the final development and testing stages.

9. Documentation and reporting of tests

Documentation helps maintain testing records that testing personnel can refer to in the future. Your QA team should religiously record testing plans, bug fixes, and general observations to help software test engineers and developers to keep track of the progress and remain updated with the latest information.

10. Perform user acceptance testing

User acceptance testing (UAT) is performed at the end of the product development by actual users before the QA personnel move the software system to the production environment.

The team conducts UAT to ensure the software systems or product performs as expected in real-world situations with the intended user base. It ensures that the product meets both technical and business specifications.

11. Leverage test management tools for standardization

A test management tool helps QA teams manage software testing end-to-end. It helps to manage test case environments, automated tests, and bugs.

The test management tool helps the team standardize the processes across various tools and improve reusability across the projects.

What Are the Limitations of Testing?

Testing run throughout development provides a framework that streamlines the process while maximizing the quality of the end product. While this should result in a product that works effectively and consistently, handling every surprise input that's thrown at it, there are limitations to the testing process. Balancing these is about knowing how much testing is *enough* and what type of testing is necessary to get as close to these ideal outcomes as possible.

Limitations of Testing in Software Development

There are certain limitations of testing in software testing that need to be considered such as incomplete test coverage, the presence of undetectable errors, time and resource constraints, reliance on test data, and the inability to guarantee absolute correctness. It's important to acknowledge these limitations and implement strategies to mitigate their impact during the testing process.

Key Limitations of Testing in Software Development:

- **Testing cannot typically uncover things that are missing or unknown**:

Tests can be written to find and identify known issues. Still, if there are issues that aren't well understood, there's no way to design an appropriate test to find them. Therefore, testing simply cannot guarantee that an application is free from errors.

- **It's impossible to test for all conditions**:

There is effectively an infinite number and variety of inputs that can go into an application, and testing them all, even if theoretically possible, is never going to be practical. Testing must cover the known probable inputs, and this is a balance between quality and deadlines.

- **Testing usually gives no insight into the root causes of errors**:

These causes need to be identified to prevent repeating the same mistakes in the future. Yet, testing will only tell whether or not the issues are present.
Other limitations:

- **Time and resource constraints:**

Limited time and resources may restrict the extent of testing that can be performed.

- **Reliance on test data:**

Testing relies on the availability of representative and comprehensive test data for accurate results.

- **Inability to guarantee absolute correctness:**

Testing can provide confidence in the software's functionality, but it cannot guarantee that it is entirely error-free.
In summary, the limitations of testing in software testing come from practical, financial, and time restrictions. The result of these restrictions is a set of hard truths that testing cannot be exhaustive, can only be completed to a particular budget, and will always be a compromise between the release date and the quality of software.
So, how do you compensate for these limitations of testing?

How to Overcome the Limitations of Testing in Software Development
The first thing to understand is that these limitations exist in every piece of software that's ever been released. The best applications from the most diligent developers all shared these limitations of testing, so it should be immediately obvious that they don't have to be feared. Instead, knowing how to mitigate them is the key to making high-quality products that meet deadlines and satisfy customer needs.
Let's go over the three limitations we listed above and how to overcome them.
1. Testing cannot typically uncover things that are missing or unknown.
The obvious solution to this problem is to know as much as possible about what to expect. Of course, this might be easier said than done. Still, when you consider that testing is as complex and difficult as the engineering process itself, it seems reasonable to assign an equally-qualified team to the testing process.
There will always be unknown unknowns, but with an experienced team of testers, you'll have a wealth of insights into what to expect and what to test for, and this dramatically reduces the number of unknown errors that will slip through the cracks.
2. It's impossible to test for all conditions.
The key to working around this issue is finding the balance between testing speed, accuracy, scope, and budget for your particular software. Automation solves many problems for smaller

tests running numerous times. If you’ve designated enough resources to your testing team, you’ll solve many problems by identifying what conditions to test.
Modern software is usually too complex to test in every feasible way and still release on a competitive schedule, so learning to prioritize the requirements and functions that are most likely to contain critical errors is the key.
Another important thing to keep in mind is the potential for requirement-specification changes down the line. If you design your tests with maintenance in mind, you’ll cover many future testings needs too.
3. Testing usually gives no insight into the root causes of errors.
For this, your testing methodologies need to incorporate or at least accommodate root cause analyses. These typically involve defining the problem, brainstorming the possible root causes, and implementing corrective procedures to fix it. From there, your testing processes need to be altered to prevent the cause from being repeated.
Of course, the best way to avoid this process is to make testing something that happens early and often and build the software with as few possible errors. Still, when errors inevitably appear, it’s important to know how to use them as a learning process and a way to improve your processes.
So, working around the limitations involves targeting your testing approach with the most powerful tools available to you. Whether that’s a highly-experienced team for testing or the most up-to-date automation solutions, it’s generally a matter of coming in prepared to do the best you can under the circumstances.
In this way, your testing objectives and attitudes should be aligned with what is optimal rather than what is perfect, the latter of which is a physical impossibility.

Why software testing is important?

Software testing saves money
There's a lot of importance of software testing. Cost-effectiveness is one of the top reasons why companies should go for software and automation testing services. It helps avoid the extra costs that occur to fix issues after the product is released to the market.
Security
It is reason why software testing is important as well as automation testing, both should be taken into consideration. It is considered to be the most sensitive part. There are a bunch of situations in which the information and details of the users are stolen and they are used for the benefits. It is considered to be the reason why people look for tested and reliable products. As a specific product undergoes testing, the user can be ensured that they are going to receive a reliable product. Testing makes products more vulnerable.
Customer satisfaction
The reason why testing is important is because it makes sure that the software is user-friendly and as per the user expectations. That makes it capable of being used by the customers it is intended for. Those who expertise in software application pentesting are familiar with the needs of customers, and unless the software can satisfy a customer’s needs, it would be a practically useless investment. Different kinds of software have different kinds of customers. That’s why just like developers, testers also tend to specialise in certain kinds of software designs. That’s what makes software testing all the more resourceful in gaining customer confidence.
Performance
If the performance of the software is low, you will find that it brings your reputation down in the market. Users are not going to trust any people that's why testing is important. There are

chances that the reputation of your organization is going to suffer. Software testing also helps in determining the performance of the Product.

VERIFICATION AND VALIDATION:

Verification is the process of checking that a software achieves its goal without any bugs. It is the process to ensure whether the product that is developed is right or not. It verifies whether the developed product fulfills the requirements that we have. Verification is static testing.
Verification means **Are we building the product right?**

Validation is the process of checking whether the software product is up to the mark or in other words product has high level requirements. It is the process of checking the validation of product i.e. it checks what we are developing is the right product. it is validation of actual and expected product. Validation is the dynamic testing.
Validation means **Are we building the right product?**

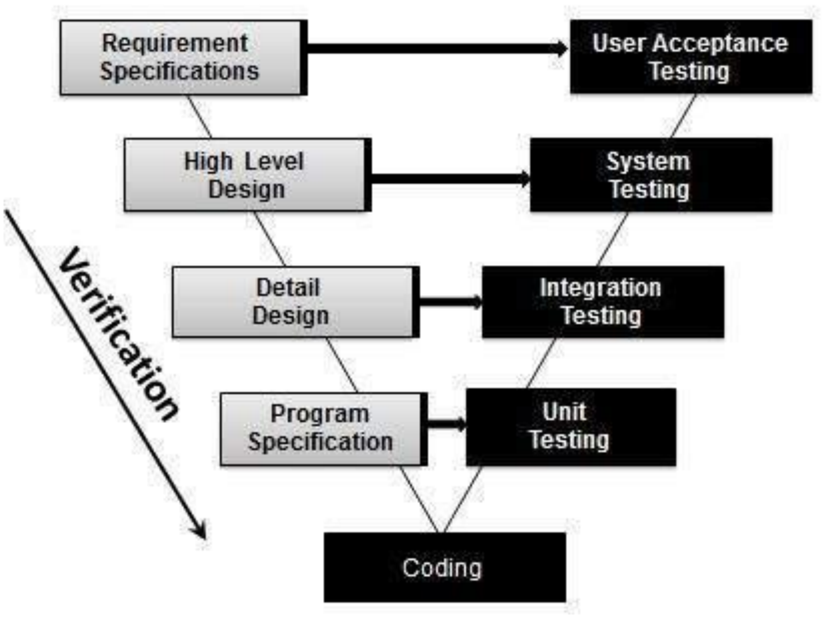
The difference between Verification and Validation is as follow:

Verification	**Validation**
It includes checking documents, design, codes and programs.	It includes testing and validating the actual product.
Verification is the static testing.	Validation is the dynamic testing.
It does *not* include the execution of the code.	It includes the execution of the code.
Methods used in verification are reviews, walkthroughs, inspections and desk-checking.	Methods used in validation are Black Box Testing, White Box Testing and non-functional testing.
It checks whether the software conforms to specifications or not.	It checks whether the software meets the requirements and expectations of a customer or not.
It can find the bugs in the early stage of the development.	It can only find the bugs that could not be found by the verification process.
The goal of verification is application and software architecture and specification.	The goal of validation is an actual product.

Verification	**Validation**
Quality assurance team does verification.	Validation is executed on software code with the help of testing team.
It comes before validation.	It comes after verification.
It consists of checking of documents/files and is performed by human.	It consists of execution of program and is performed by computer.

Verification Testing - Workflow:

verification testing can be best demonstrated using V-Model. The artefacts such as test Plans, requirement specification, design, code and test cases are evaluated.



Activities:

- Reviews
- Walkthroughs
- Inspection

Verification and Validation Testing

In this section, we will learn about verification and validation testing and their major differences.

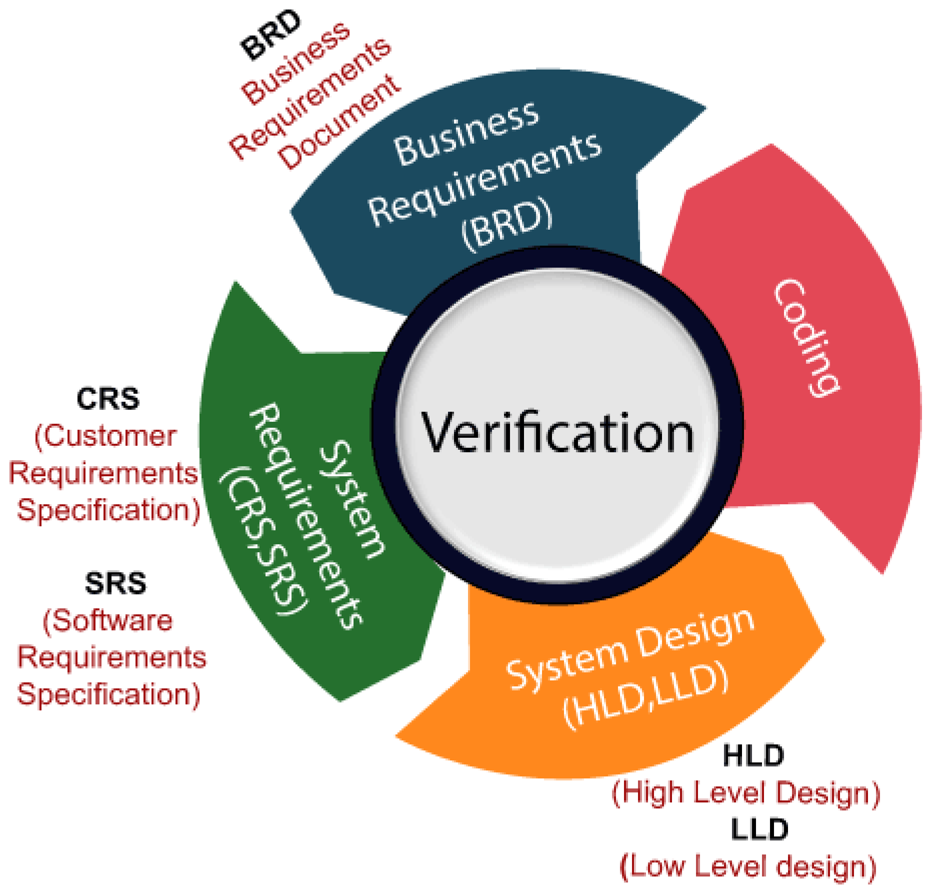
Verification testing

Verification testing includes different activities such as business requirements, system requirements, design review, and code walkthrough while developing a product.

It is also known as static testing, where we are ensuring that "**we are developing the right product or not**". And it also checks that the developed application fulfilling all the requirements given by the client.

Activities involved in verification:

1. Inspections
2. Reviews
3. Walkthroughs
4. Desk-checking

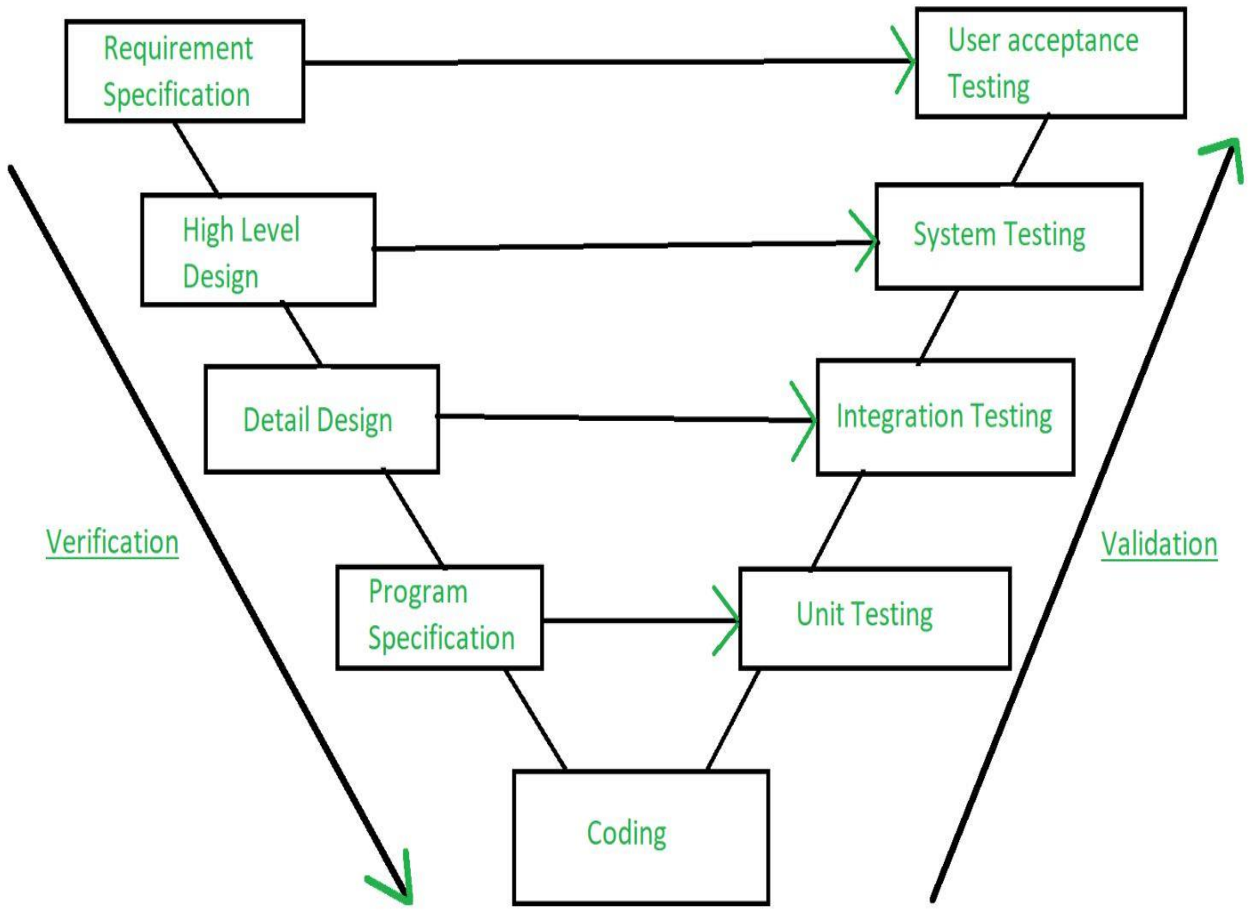

BRD
Business Requirements Document
Business Requirements (BRD)
Coding
Verification
CRS
(Customer Requirements Specification)
SRS
(Software Requirements Specification)
System Requirements (CRS,SRS)
System Design (HLD,LLD)
HLD
(High Level Design)
LLD
(Low Level design)

Validation testing

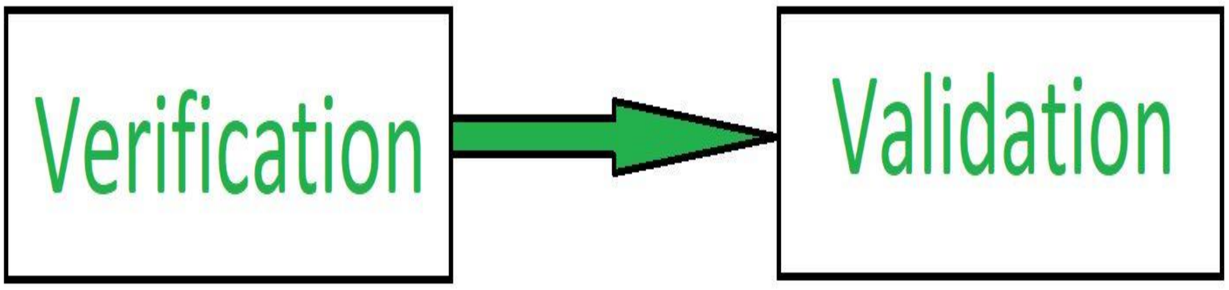
Validation testing is testing where tester performed functional and non-functional testing. Here **functional testing** includes Unit Testing (UT), Integration Testing (IT) and System Testing (ST), and **non-functional** testing includes User acceptance testing (UAT).

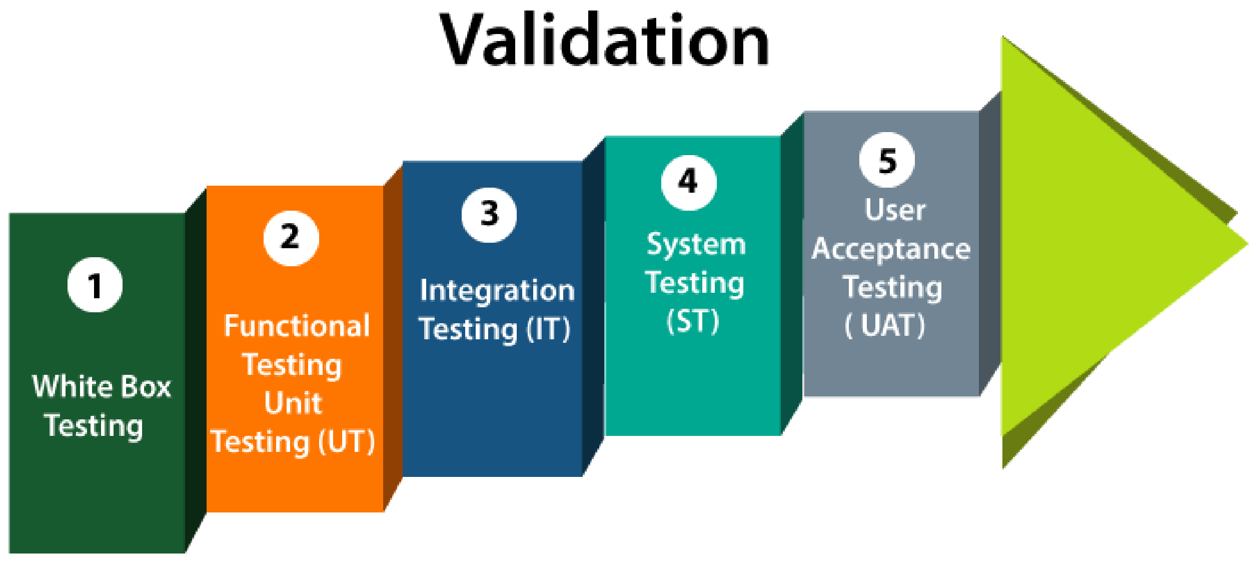
Validation testing is also known as dynamic testing, where we are ensuring that **"we have developed the product right."** And it also checks that the software meets the business needs of the client.
Validation is the process of checking whether the software product is up to the mark or in other words product has high level requirements. It is the process of checking the validation of product i.e., it checks what we are developing is the right product. it is validation of actual and expected product.
Validation is the **Dynamic Testing**.
Activities involved in validation:

1. Black box testing
2. White box testing
3. Unit testing
4. Integration testing



Note: Verification is followed by Validation.





> *Note: Verification and Validation process are done under the V model of the software development life cycle.*

Difference between verification and validation testing

Verification	Validation
We check whether we are developing the right product or not.	We check whether the developed product is right.
Verification is also known as **static testing**.	Validation is also known as **dynamic testing**.
Verification includes different methods like Inspections, Reviews, and Walkthroughs.	Validation includes testing like functional testing, system testing, integration, and User acceptance testing.
It is a process of checking the work-products (not the final product) of a development cycle to decide whether the product meets the specified requirements.	It is a process of checking the software during or at the end of the development cycle to decide whether the software follow the specified business requirements.

Quality assurance comes under verification testing.	**Quality control** comes under validation testing.
The execution of code does not happen in the verification testing.	In validation testing, the execution of code happens.
In verification testing, we can find the bugs early in the development phase of the product.	In the validation testing, we can find those bugs, which are not caught in the verification process.
Verification testing is executed by the Quality assurance team to make sure that the product is developed according to customers' requirements.	Validation testing is executed by the testing team to test the application.
Verification is done before the validation testing.	After verification testing, validation testing takes place.
In this type of testing, we can verify that the inputs follow the outputs or not.	In this type of testing, we can validate that the user accepts the product or not.

What is Requirements Verification?

Requirements Verification is the process of confirming that the system requirements contain all the necessary elements of well-written requirements. Requirements verification is a critical step in system development, as it helps ensure that the product meets its objectives and functions as intended.

Before design, the requirements should be verified and approved to prevent rework. If the quality criteria are not checked, requirement verification will inevitably be done during product development and creation processes. Requirements that are missing or incorrect can lead to products that don't meet customer expectations. Requirements verification is important to do early and often to prevent these issues.

Importance of Requirements Verification:

The main goals of requirements verification are to ensure completeness, correctness, and consistency of the system requirements.

This phase can uncover missing requirements, ambiguous, or invalid ones, reducing rework and cost overruns. It's far more effective to resolve a little problem upfront than it is in the future when hundreds of lines of code or a completely manufactured complex product must be tracked down and fixed.

Requirements verification is necessary because it helps ensure that the system to be built will meet its objectives and functions as intended. Incomplete, incorrect, or inconsistent requirements can lead to problems during system development, testing, and deployment.

What is Requirements Verification?

Requirements Verification is the process of confirming that the system requirements contain all the necessary elements of well-written requirements. Requirements verification is a critical step in system development, as it helps ensure that the product meets its objectives and functions as intended.

Before design, the requirements should be verified and approved to prevent rework. If the quality criteria are not checked, requirement verification will inevitably be done during product development and creation processes. Requirements that are missing or incorrect can lead to products that don't meet customer expectations. Requirements verification is important to do early and often to prevent these issues.

Importance of Requirements Verification:

The main goals of requirements verification are to ensure completeness, correctness, and consistency of the system requirements.

This phase can uncover missing requirements, ambiguous, or invalid ones, reducing rework and cost overruns. It's far more effective to resolve a little problem upfront than it is in the future when hundreds of lines of code or a completely manufactured complex product must be tracked down and fixed.

Requirements verification is necessary because it helps ensure that the system to be built will meet its objectives and functions as intended. Incomplete, incorrect, or inconsistent requirements can lead to problems during system development, testing, and deployment.

Difference Between Verification and Validation:

People often get confused between verification and validation. Actually, they are not the same.

According to the IREB glossary of Requirements Engineering Terminology:

Validation is the process of confirming that an item (a system, a work product or a part thereof) matches its stakeholders' needs. In Requirements Engineering, validation is the process of confirming that the documented requirements match their stakeholders' needs; in other words: **whether the right requirements have been specified**.

Verification is the process of confirming that an item (a system, a work product, or a part thereof) fulfills its specification. Requirements verification is the process of confirming that the requirements have been documented properly and satisfy the quality criteria for requirements; in other words, **whether the requirements have been specified right**.

In simpler terms, Requirements verification is the process of confirming that the system requirements contain all the necessary elements of well-written requirements. Requirements validation is the process of confirming that the written requirements agree with the stakeholders' requests.

In other words, verification is about checking whether the requirements are complete, correct, and consistent. Validation is about checking whether the requirements describe the intended system objectives and functions.

Quality criteria in Requirements Verification:

The quality of requirements is crucial for successful software development. There are several sets of quality criteria that requirements must meet, such as those defined by IREB or the SMART rule. The IREB syllabus defines six quality criteria that every single requirement must meet, including *adequacy, necessity, unambiguity, completeness, understandability, and verifiability*. Additionally, the set of requirements should meet other quality criteria, including consistency, non-redundancy, completeness, modifiability, traceability, and conformance.

It is important to select the most relevant set of quality criteria for the project and ensure that the requirements are fulfilled them. However, not all requirements need to meet all quality criteria to the same extent. Instead, the value of each requirement should be considered in the context of the project, and the corresponding quality criteria should be prioritized accordingly. By verifying that requirements meet relevant quality criteria, the risk of failure in software development can be reduced.

Techniques Used in Requirements Verification:

There are several techniques that can be used for requirements verification to ensure that the requirements meet the necessary quality criteria. Some of the commonly used techniques include:

1. **Inspection:** This technique involves a systematic review of the requirements by a team of experts to identify any errors, omissions, or inconsistencies. It can be conducted manually or using automated tools.
2. **Testing:** Testing involves designing and executing tests to verify that the requirements meet the desired functionality and quality criteria. It can be conducted at different levels, such as unit testing, integration testing, and acceptance testing.
3. **Walkthrough:** In a walkthrough, the requirements are reviewed by a group of stakeholders who provide feedback and identify any issues or concerns. It is typically less formal than an inspection.
4. **Prototyping:** Prototyping involves creating a simplified version of the software to validate the requirements and identify any issues or limitations. It can help stakeholders visualize and understand the system better.
5. **Simulation:** Simulation involves creating a model of the system and testing its behavior under different scenarios. It can help identify issues with the requirements that may not be apparent in static documentation.
6. **Traceability Analysis:** Traceability analysis involves tracking the relationships between the requirements and other artifacts, such as design documents and test cases, to ensure that the requirements are complete, consistent, and verifiable.

These techniques can be used individually or in combination to verify that the requirements meet the necessary quality criteria and to reduce the risk of errors and inconsistencies in the final software product.

VERIFICATION OF HL DESIGN:

All the requirements mentioned in SRS documents are addressed in this phase and work in direction of designing the solution. High level design takes the second place in SDLC, wherein there is a high probability of finding bugs. Therefore, high level/design must be verified as next step stop in early testing. If bug goes undetected in high level design phase, then its cost of fixing increases with every phase.

High level design is divided in three parts -

1. Data design.
2. Architectural design.
3. Interface design.

1) Data design:

- It creates model of a data that is represented at a high level of abstraction.
- At the program component level, the design of data structure and associated algorithms required to manipulate them is essential to create high quality applications.

Verification of Data Design.

- Check for sizes of data structure have been estimated properly.
- Check the provisions of overflow.
- Check consistency of data format with requirements.
- Check data usage is consistent with its declaration.
- Check for relationship among data object in data dictionary.

- Check consistency of database and data warehouse with requirement specified in SRS.

ii) Verification of Architectural design:

- it focuses on representation of structure of s/w component, their properties and interactions.

Verification of architectural design:

- check for every functional requirement from SRS is included in design.
- check all exception handling conditions.
- Verify process of transform mapping and transaction mapping transition from requirement model to architectural design.
- Check functionality of each module according to requirement specified.
- Check interdependence and interface between modules.
- Tester should also verify for coupling and cohesion. A good design should have low coupling and high cohesion.

iii) Interface design:

- It creates effective communication medium between interfaces of different software modules, interfaces between software systems and any other external entity and interface between user and software system.

Verification of interface design:

- Check all interfaces between modules according to architectural design.
- Check all interfaces between software and non-human producer and consumer information.
- Check interfaces between human and computer.
- Check interfaces for their consistency.
- Check response time for all interface within required ranges.
- Check for help facility and error handling messages.

- For typed command interactions, check mapping between every menu option and their corresponding command.

iv) Verification of low-level design:

- In low level design a detailed design of modules and data are prepared such that an operational software is ready. For this SOD is preferred where all modules and their interfaces are defined.
- For verification of low-level design SRS and SDD of individual modules are referred.

4 verify SRS of individual module.

4 verify SDD of each individual module.

4 in low level design, data structure, interfaces and algorithms are represented by design notations, verify the consistency of every item with their design notations.

Verification of User-interface Design

The points to be considered for the verification of user-interface design are:

- Check all the interfaces between modules according to the architectural design.
- Check all the interfaces between software and other non-human producer and consumer of information.
- Check all the interfaces between the human and computer.
- Check all the aforementioned interfaces for their consistency.
- Check the response time for all the interfaces are within required ranges. It is very essential for real-time systems where response time is very crucial.
- For a Help Facility, verify the following:

  (i) The representation of Help is in its desired manner

  (ii) The user returns to the normal interaction from Help
- For error messages and warnings, verify the following:

  (i) Whether the message clarifies the problem

  (ii) Whether the message provides constructive advice for recovering from the error
- For typed command interaction, check the mapping between every menu option and their cor responding commands.

User Interface Testing

To ensure visual aesthetic of web application is well accepted, UI and Usability testing become a key aspect of the overall QA practice. Any application which can be accessed through an URL is a Web-based application. In such applications, we mainly test the front end of the application which is to be used by the end user.

Each browser displays web pages differently, so it is important that page should look same on different browsers. If a webpage is displayed distorted and unmanaged then it will lead viewers to exit the webpage. So, a website should undergo UI testing for better results.

Browser testing comprises of below two types:

Functionality testing

Testing of different functions throughout the application. It involves validating all the navigations as well as all field values which are present in the front-end pages using all positive as well as negative scenarios.

UI Testing

Testing the look and feel factor of the web page. Look and feel factor includes display type, font, alignment, radio button, checkbox etc.

- Areas covered in UI testing are Usability, Look & feel, Navigation controls/navigation bars, instructions, and technical information style, images, tables, accessibility etc.
- For testing for accessibility, we have to check with W3C-Web content accessibility guidelines.

Difference between High Level Design and Low-Level Design:

High Level Design:

High Level Design in short HLD is the general system design means it refers to the overall system design. It describes the overall description/architecture of the application. It includes the description of system architecture, data base design, brief description on systems, services, platforms and relationship among modules. It is also known as macro level/system design. It is created by solution architect. It converts the Business/client requirement into High Level Solution. It is created first means before Low Level Design.

Low Level Design:

Low Level Design in short LLD is like detailing HLD means it refers to component-level design process. It describes detailed description of each and every module means it includes actual logic for every system component and it goes deep into each modules specification. It is also known as micro level/detailed design. It is created by designers and developers. It converts the High-Level Solution into Detailed solution. It is created second means after High Level Design.

Difference between High Level Design and Low-Level Design:

S.no.	HIGH LEVEL DESIGN	LOW LEVEL DESIGN
01.	High Level Design is the general system design means it refers to the overall system design.	Low Level Design is like detailing HLD means it refers to component-level design process.
02.	High Level Design in short called as HLD.	Low Level Design in short called as LLD.
03.	It is also known as macro level/system design.	It is also known as micro level/detailed design.
04.	It describes the overall description/architecture of the application.	It describes detailed description of each and every module.
05.	High Level Design expresses the brief functionality of each module.	Low Level Design expresses details functional logic of the module.
06.	It is created by solution architect.	It is created by designers and developers.
07.	Here in High Level Design the participants are design team, review team, and client team.	Here in Low Level Design participants are design team, Operation Teams, and Implementers.

S.no.	HIGH LEVEL DESIGN	LOW LEVEL DESIGN
08.	It is created first means before Low Level Design.	It is created second means after High Level Design.
09.	In HLD the input criteria are Software Requirement Specification (SRS).	In LLD the input criteria are reviewed High Level Design (HLD).
10.	High Level Solution converts the Business/client requirement into High Level Solution.	Low Level Design converts the High-Level Solution into Detailed solution.
11.	In HLD the output criteria are data base design, functional design and review record.	In LLD the output criteria are program specification and unit test plan.

WHAT IS VALIDATION TESTING: DEFINITION, PHASES & TYPES!

What is Validation Testing?

The definition of validation testing in software engineering is in place to determine if the existing system complies with the system requirements and performs the dedicated functions for which it is designed along with meeting the goals and needs of the organization.

The Advantages of Validation Testing:

- *To ensure customer satisfaction*
- *To be confident about the product*
- *To fulfill the client's requirement until the optimum capacity*
- *Software acceptance from the end-user*

Whenever any particular software is tested then the main motive is to check the quality against the defects being found. The developers fix the bugs and the software is rechecked to make sure that absolutely no bugs are left out in that. This not only shoots the product's quality but also its user acceptance.

What are the Phases of the Validation Testing Process?

- **Define Requirements**– Mapping out the plan for the requirement gathering process. This will not only include planning out the entire process beforehand but also mapping out the exact requirements that are needed.

- **Team Selection**– To get a qualified, knowledgeable, and skilled team onboard. The team selection process involves selecting the individuals as per their past capacity and technical tuning so that they can easily attune themselves to the nature of the bug.
- **Maintaining Documentation**– Any form of testing requires voluminous user specification documentation along with several release cases, test cases, and manuals that have to be jolted down so that they are no confusion even in case of an exit of any core member of the team.

- **Validation Report**– The software is evaluated as per user specifications and a proper validation report is submitted in order to cross-check the evaluations along with getting an estimated date and round-off for the bug removal and for the system to start functioning properly.

- **Incorporation of changes**–Incorporate the changes that have been validated in the last stage.

What are the Types of Validation Testing?

Validation testing types a V-shaped testing pattern, which includes its variations and all the activities that it consists of are:

Unit Testing – It is an important type of validation testing. The point of the unit testing is to search for bugs in the product segment. Simultaneously, it additionally confirms crafted modules and articles which can be tried independently.

Integration testing -This is a significant piece of the validation model wherein the interaction between, where the association between the various interfaces of the pertaining component is tried. Alongside the communication between the various pieces of the framework, the connection of the framework with the PC working framework, document framework, equipment, and some other programming framework it may cooperate with, is likewise tried.

System testing – System testing is done when the whole programming framework is prepared. The principal worry of framework testing is to confirm the framework against the predefined necessities. While doing the tests, the tester isn't worried about the internals of the framework however checks if the framework acts according to desires.

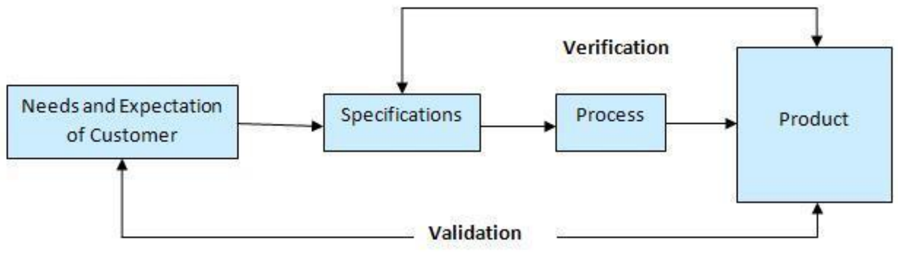
User acceptance testing – During this testing, the tester actually needs to think like the customer and test the product concerning client needs, prerequisites, and business forms and decide if the product can be given over to the customer or not.

What is validation testing?

Validation testing is the practice of ensuring that software meets the quality standards set by the customer and that the product meets customer requirements. It is one of many types of testing in software.
In certain cases, validation testing is also sometimes called acceptance testing because it is usually performed once the developer has finished writing the code. In this sense, it is a process of checking whether a product can be accepted based on certain criteria or requirements that are defined in advance by the consumer or user.
Businesses often perform validation testing as part of their software development life cycle (SDLC). Both waterfall and agile models require code review to ensure that it works as intended and does not contain defects, bugs, or errors.
What is Validation in software testing? or What is software validation?

Validation is determining if the system complies with the requirements and performs functions for which it is intended and meets the organization's goals and user needs

- Validation is done at the end of the development process and takes place after **verifications** are completed.
- It answers the question like: **Am I building the right product?**
- Am I accessing the right data (in terms of the data required to satisfy the requirement).
- It is a High level activity.
- Performed after a work product is produced against established criteria ensuring that the product integrates correctly into the environment.
- Determination of correctness of the final software product by a development project with respect to the user needs and requirements.



Software verification and validation

According to the **Capability Maturity Model (CMM)** we can also define validation as The process of evaluating software during or at the end of the development process to determine whether it satisfies specified requirements. [IEEE-STD-610].

A product can pass while verification, as it is done on the paper and no running or functional application is required. But, when same points which were verified on the paper is actually developed then the running application or product can fail while validation. This may happen because when a product or application is build as per the specification but these specifications are not up to the mark hence they fail to address the user requirements.

Advantages of Validation:

1. During verification if some **defects** are missed then during validation process it can be caught as failures.
2. If during verification some specification is misunderstood and development had happened then during validation process while executing that functionality the difference between the actual result and expected result can be understood.
3. Validation is done during testing like feature testing, integration testing, system testing, load testing, compatibility testing, stress testing, etc.
4. Validation helps in building the right product as per the customer's requirement and helps in satisfying their needs.

Validation is basically done by the testers during the testing. While validating the product if some deviation is found in the actual result from the expected result then a bug is reported or an **incident is raised**. Not all incidents are bugs. But all bugs are incidents. Incidents can also be of type 'Question' where the functionality is not clear to the tester.

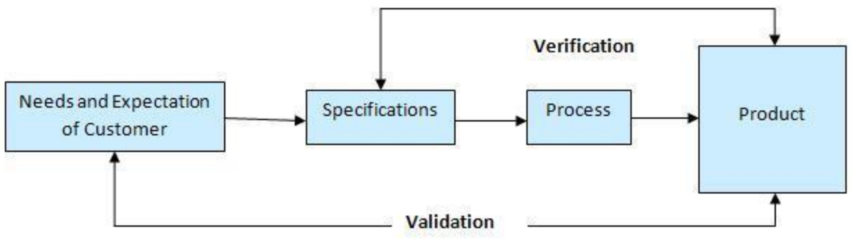
Hence, validation helps in unfolding the exact functionality of the features and helps the testers to understand the product in much better way. It helps in making the product more user friendly.

What is Verification in software testing? or What is software verification?

Verification makes sure that the product is designed to deliver all functionality to the customer.

- Verification is done at the starting of the development process. It includes **reviews** and meetings, **walk-throughs**, **inspection**, etc. to evaluate documents, plans, code, requirements and specifications.

- Suppose you are building a table. Here the verification is about checking all the parts of the table, whether all the four legs are of correct size or not. If one leg of table is not of the right size it will imbalance the end product. Similar behavior is also noticed in case of the software product or application. If any feature of software product or application is not up to the mark or if any **defect** is found then it will **result into the failure** of the end product. Hence, verification is very important. It takes place at the starting of the development process.



- Software verification and validation
- It answers the questions like: **Am I building the product right?**
- Am I accessing the data right (in the right place; in the right way).
- It is a Low level activity
- Performed during development on key artifacts, like walkthroughs, reviews and inspections, mentor feedback, training, checklists and standards.
- Demonstration of consistency, completeness, and correctness of the software at each stage and between each stage of the development life cycle.

According to the **Capability Maturity Model (CMM)** we can also define verification as the process of evaluating software to determine whether the products of a given development phase satisfy the conditions imposed at the start of that phase. [IEEE-STD-610].

Advantages of Software Verification :

1. Verification helps in lowering down the count of the defect in the later stages of development.
2. Verifying the product at the starting phase of the development will help in understanding the product in a better way.
3. It reduces the chances of failures in the software application or product.
4. It helps in building the product as per the customer specifications and needs.

Types of Software Testing

Testing is the process of executing a program to find errors. To make our software perform well it should be error-free. If testing is done successfully it will remove all the errors from the software. In this article, we will discuss first the principles of testing and then we will discuss, the different types of testing.

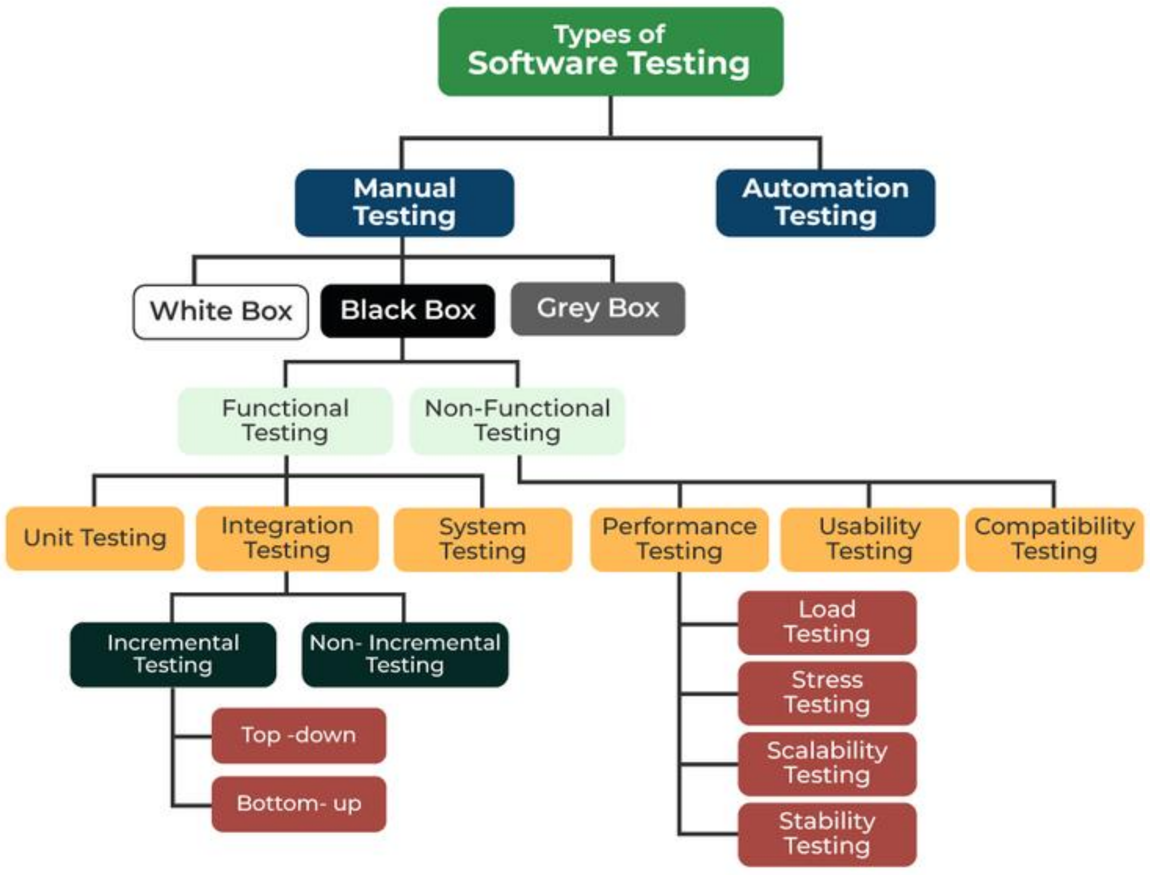
Principles of Testing

- All the tests should meet the customer's requirements.
- To make our software testing should be performed by a third party.

- Exhaustive testing is not possible. As we need the optimal amount of testing based on the risk assessment of the application.
- All the tests to be conducted should be planned before implementing it
- It follows the Pareto rule(80/20 rule) which states that 80% of errors come from 20% of program components.
- Start testing with small parts and extend it to large parts.
- Types of Testing

There are basically 10 types of Testing.

- Unit Testing
- Integration Testing
- System Testing
- Functional Testing
- Acceptance Testing
- Smoke Testing
- Regression Testing
- Performance Testing
- Security Testing
- User Acceptance Testing



Types of Software Testing

Unit Testing

Unit testing is a method of testing individual units or components of a software application. It is typically done by developers and is used to ensure that the individual units of the software are working as intended. Unit tests are usually automated and are designed to test specific parts of the code, such as a particular function or method. Unit testing is done at the lowest level of the software development process, where individual units of code are tested in isolation.

Advantages of Unit Testing: Some of the advantages of Unit Testing are listed below.

- It helps to identify bugs early in the development process before they become more difficult and expensive to fix.
- It helps to ensure that changes to the code do not introduce new bugs.
- It makes the code more modular and easier to understand and maintain.
- It helps to improve the overall quality and reliability of the software.

- It’s important to keep in mind that Unit Testing is only one aspect of software testing and it should be used in combination with other types of testing such as integration testing, functional testing, and acceptance testing to ensure that the software meets the needs of its users.
- It focuses on the smallest unit of software design. In this, we test an individual unit or group of interrelated units
-

Integration Testing

Integration testing is a method of testing how different units or components of a software application interact with each other. It is used to identify and resolve any issues that may arise when different units of the software are combined. Integration testing is typically done after unit testing and before functional testing and is used to verify that the different units of the software work together as intended.

Different Ways of Performing Integration Testing: There are different ways of Integration Testing that are discussed below.

- Top-down integration testing: It starts with the highest-level modules and differentiates them from lower-level modules.
- Bottom-up integration testing: It starts with the lowest-level modules and integrates them with higher-level modules.
- Big-Bang integration testing: It combines all the modules and integrates them all at once.
- Incremental integration testing: It integrates the modules in small groups, testing each group as it is added.

Advantages of Integrating Testing

1. It helps to identify and resolve issues that may arise when different units of the software are combined.
2. It helps to ensure that the different units of the software work together as intended.
3. It helps to improve the overall reliability and stability of the software.
4. It’s important to keep in mind that Integration testing is essential for complex systems where different components are integrated together.
5. As with unit testing, integration testing is only one aspect of software testing and it should be used in combination with other types of testing such as unit testing, functional testing, and acceptance testing to ensure that the software meets the needs of its users.

The objective is to take unit-tested components and build a program structure that has been dictated by design. Integration testing is testing in which a group of components is combined to produce output.

Regression Testing

Regression testing is a method of testing that is used to ensure that changes made to the software do not introduce new bugs or cause existing functionality to break. It is typically done after changes have been made to the code, such as bug fixes or new features, and is used to verify that the software still works as intended.

Regression testing can be performed in different ways, such as:

- **Retesting**: This involves testing the entire application or specific functionality that was affected by the changes.
- **Re–execution**: This involves running a previously executed test suite to ensure that the changes did not break any existing functionality.
- **Comparison**: This involves comparing the current version of the software with a previous version to ensure that the changes did not break any existing functionality.

Advantages of Regression Testing

- It helps to ensure that changes made to the software do not introduce new bugs or cause existing functionality to break.
- It helps to ensure that the software continues to work as intended after changes have been made.
- It helps to improve the overall reliability and stability of the software.
- It's important to keep in mind that regression testing is an ongoing process that should be done throughout the software development
- lifecycle to ensure that the software continues to work as intended. It should be automated as much as possible to save time and resources. Additionally, it's important to have a well-defined regression test suite that covers

Every time a new module is added leads to changes in the program. This type of testing makes sure that the whole component works properly even after adding components to the complete program.

Smoke Testing

Smoke Testing is done to make sure that the software under testing is ready or stable for further testing

It is called a smoke test as the testing of an initial pass is done to check if it did not catch fire or smoke in the initial switch-on.

Alpha Testing

Alpha testing is a type of validation testing. It is a type of acceptance testing that is done before the product is released to customers. It is typically done by QA people.

Example:

When software testing is performed internally within the organisation.

Beta Testing

The beta test is conducted at one or more customer sites by the end-user of the software. This version is released for a limited number of users for testing in a real-time environment.

Example:

When software testing is performed for the limited number of people.

System Testing

System Testing is carried out on the whole system in the context of either system requirement specifications or functional requirement specifications or in the context of both. The software is tested such that it works fine for the different operating systems. It is covered under the black box testing technique. In this, we just focus on the required input and output without focusing on internal work. In this, we have security testing, recovery testing, stress testing, and performance testing.
Example:
This includes functional as well as nonfunctional testing.

Stress Testing

In Stress Testing, we give unfavorable conditions to the system and check how they perform in those conditions.
Example:
i. Test cases that require maximum memory or other resources are executed.
ii. Test cases that may cause thrashing in a virtual operating system.
ii. Test cases that may cause excessive disk requirement Performance Testing.
It is designed to test the run-time performance of software within the context of an integrated system. It is used to test the speed and effectiveness of the program. It is also called load testing. In it, we check, what is the performance of the system in the given load.
Example:
Checking several processor cycles.

Object-Oriented Testing

Object-Oriented Testing testing is a combination of various testing techniques that help to verify and validate object-oriented software. This testing is done in the following manner:

- Testing of Requirements,
- Design and Analysis of Testing,
- Testing of Code,
- Integration testing,
- System testing,
- User Testing.

Acceptance Testing

Acceptance testing is done by the customers to check whether the delivered products perform the desired tasks or not, as stated in the requirements. We use Object-Oriented Testing for discussing test plans and for executing the projects.
Advantages of Software Testing

- Improved software quality and reliability.
- Early identification and fixing of defects.
- Improved customer satisfaction.
- Increased stakeholder confidence.
- Reduced maintenance costs.

Disadvantages of Software Testing

- Time-Consuming and adds to the project cost.
- This can slow down the development process.

- Not all defects can be found.
- Can be difficult to fully test complex systems.
- Potential for human error during the testing process.