**Notes for MCA-II (Semester- III)**

**Subject :- Software Testing & Quality Assurance**

**(Subject Code:- IT-33)**

**Chapter: 2] Software Testing Fundamentals**

**2.1 Definition & Objectives:**

* **Definitions [as per IEEE / ANSI]:**
* **Testing:**

Testing is the process of operating a system or component under specified conditions, observing or recording results and making an evaluation of some aspect of the system or component.

* Testing is the process of analyzing a software item to detect the difference between existing & required conditions and to evaluate the features of the software items.
* **Verification:**

Verification is the process of evaluating a system or component to determine whether the products of a given development phase satisfy the conditions imposed at the start of the phase.

* **Validation:**

Validation is the process of evaluating a system or component during or at the end of the development process to determine whether it satisfies the specified requirements.

**Testing = verification + Validation**

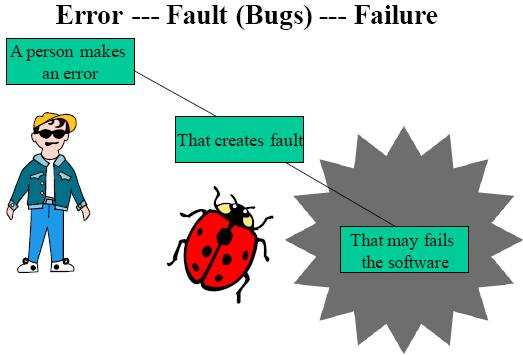
* Software testing is the process of examining the software product against its requirements. Thus it is a process that involves verification of product with respect to its written requirements and conformance of requirements with user needs.
* From another perspective, software testing is the process of executing software product on test data and examining its output vis-à-vis the documented behavior.
* **Testing Objectives:**
* S/w testing remove bugs from s/w modules / statements/ function / branch / looping etc.
* S/w testing removes errors in the system at interfaces.
* S/w testing provides a system as per the requirement specification.
* S/w testing meets the design specifications.
* S/w testing ensures the correct data flow.
* S/w testing ensures that the process & desired product standards are followed.
* S/w testing gives the quality product.
* S/w testing is required to review the requirement specifications, design & coding.
* The correct approach to testing a scientific theory is not to try to verify it, but to seek to refute the theory. That is to prove that it has errors. (Popper 1965)
* The goal of testing is to expose latent defects in a software system before it is put to use.
* A software tester tries to break the system. The objective is to show the presence of a defect not the absence of it.
* Testing cannot show the absence of a defect. It only increases your confidence in the software.
* This is because exhaustive testing of software is not possible – it is simply too expensive and needs virtually infinite resources.

**2.2 Role of Testing & Its effect on Quality:-**

**Software Testing** is a method to check whether the actual software product matches expected requirements and to ensure that software product is[Defect](https://www.guru99.com/defect-management-process.html)free. It involves execution of software/system components using manual or automated tools to evaluate one or more properties of interest. The purpose of software testing is to identify errors, gaps or missing requirements in contrast to actual requirements.

**2.3 Causes of Software Failure :**

* **What is a BUG?**
* Bug is the ***fault*** i.e. an error in software which if executed may cause failure. A manifestation of error in software.
* A fault in program which causes the program to perform in an unintended or unanticipated manner which is found in development environment.
* ***Failure*** is nothing but deviation of the software from its expected delivery or service.
* An ***Error*** is nothing but a human action that produces an incorrect result.
* A mismatch between the program and its specification is an error in the program if and only if the specification exists and is correct. It is deviation from actual and expected values.
* An error found in product after shifting it to customer side. An incorrect step process of data definition in computer program within the context of s/w process is a ***Defect*** or fault in product.



**Figure : Process of occurring Bug**

*“Failure is an event; Fault is a state of the software caused by error.”*

* **Types of Software Bugs:**

Following are the most common software errors that aid you in software testing. This helps you to identify errors systematically and increases the efficiency and productivity of software testing. Also, you can use this as a checklist while preparing test cases and while performing testing.

* **Types of bugs/errors with examples:**
* **User Interface Errors:**

Missing/Wrong Function Doesn’t do what the user expects, Missing information, Misleading, Confusing information, Wrong content in Help text, Inappropriate error messages. Performance issues - Poor responsiveness, Can't redirect output, inappropriate use of key board.

* **Error Handling:**

Inadequate - protection against corrupted data, tests of user input, version control; Ignores – overflow, data comparison, Error recovery – aborting errors, recovery from hardware problems.

* **Boundary related errors:**

Boundaries in loop, space, time, memory, mishandling of cases outside boundary.

* **Calculation errors:**

Bad Logic, Bad Arithmetic, Outdated constants, Calculation errors, incorrect conversion from one data representation to another, Wrong formula, Incorrect approximation.

* **Initial and Later states:**

Failure to - set data item to zero, to initialize a loop-control variable, or re-initialize a pointer, to clear a string or flag, Incorrect initialization.

* **Control flow errors:**

Wrong returning state assumed, Exception handling based exits, Stack underflow/overflow, Failure to block or un-block interrupts, Comparison sometimes yields wrong result, Missing/wrong default, and Data Type errors.

* **Errors in Handling or Interpreting Data:**

Un-terminated null strings, overwriting a file after an error exit or user abort.

* **Race Conditions:**

Assumption that one event or task finished before another begins, Resource races, Tasks starts before its prerequisites are met, Messages cross or don't arrive in the order sent.

* **Load Conditions:**

Required resources are not available, No available large memory area, Low priority tasks not put off, Doesn't erase old files from mass storage, Doesn't return unused memory.

* **Hardware:**

Wrong Device, Device unavailable, Underutilizing device intelligence, Misunderstood status or return code, Wrong operation or instruction codes. Source, Version and ID Control: No Title or version ID, Failure to update multiple copies of data or program files.

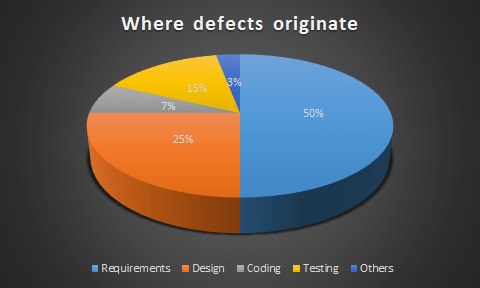
* **Testing Errors:**

Failure to notice/report a problem, Failure to use the most promising test case, Corrupted data files, Misinterpreted specifications or documentation, Failure to make it clear how to reproduce the problem, Failure to check for unresolved problems just before release, Failure to verify fixes, Failure to provide summary report.

* **2.4 Economics of Testing** :- Testing is about **getting real feedback quickly**, reducing wasteful testing activities, and putting a mirror in front of our applications. It becomes advantageous to understand the costs of these activities and direct the effort investment where it's most beneficial

There is a definite economic impact of software testing. One economic impact is from the cost of defects and the another is the way we perform testing. Cost of defect is a very real and very tangible cost. The second point the way we perform testing will not consider it now.

**Where defects originate?**



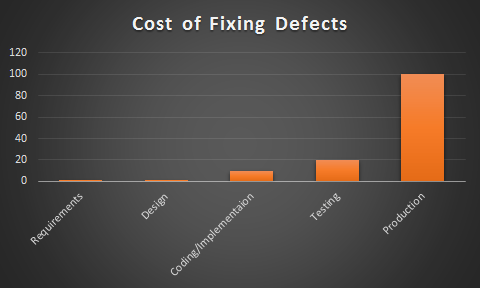
One of the most commonly understood facts about defects is that most defects originate in the requirements definition phase of a project. The next runner-up is the design phase.

Some problems in getting accurate, clear, and testable requirements are:

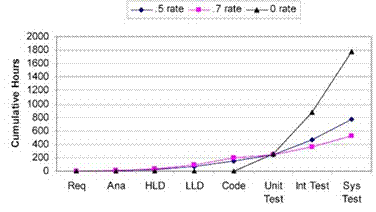
* Many people do not have a solid requirements gathering process
* Few people have been trained in or understand the dynamics of requirements
* Projects, people, and the world around us change very quickly
* The English language is ambiguous and even what we consider clear language can be interpreted differently by different people.

**Relative cost of fixing defects**

One of the known facts about software defects is that the longer they go undetected, the more expensive they are to fix. Although research differs on the exact ratios, the general rule is 1:10:100. That is, if a defect costs one unit (hour, dollar, etc.) to fix in requirements and design, it costs 10 units to fix in testing (system/acceptance) and over 100 times to fix in production. This cost of defects doesn’t even take into account the impact cost of defects. These t of fixing in production may be even higher than 100 times. Costs could be attributed to lost revenue, reimbursements, fraud, lost customers, bad public relations, and litigation.

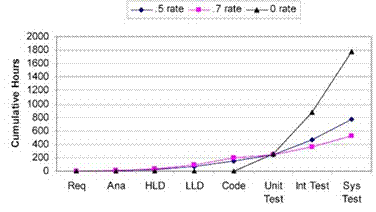


The following chart is a summary of the relative cost of fixing defects detected at various stages in the software development lifecycle process in a typical project. The cost of defect removal increases exponentially as the development lifecycle progresses. In addition, the later defects are found and fixed, the greater the risk to the business they pose.



**Defect insertion and detection points**

Defects are introduced to a system at a number of points during the software development lifecycle. The following chart is a model of a typical development lifecycle that illustrates the usual points of defect insertion and detection. The left-most line here shows at which points in the project lifecycle the defects are introduced. The second line illustrates the traditional points at which defects are detected and removed.

****

**Conclusion**

* Most defects are created in the early stages of a project
* Most defects are found in the later stages of a project
* It costs 10 to 100 times as much to fix a defect in the later phases of a project.

So, what does all of this mean? The main conclusion is that most people perform testing too late in the process. These people wonder why testing is so expensive and why their projects are often over budget.

*If you really want to make your testing more efficient and reduce the overall cost of testing and defects, test early in the project and continue testing throughout the project.*

* **2.5 Seven Testing Principle** :-
* Testing shows presence of defects
* Exhaustive testing is not possible
* Early testing
* Defect clustering
* Pesticide paradox
* Testing is context dependent
* Absence of errors fallacy
* **Testing shows the presence of defects:** The goal of software testing is to make the software fail. Software testing reduces the presence of defects. Software testing talks about the presence of defects and doesn’t talk about the absence of defects. Software testing can ensure that defects are present but it can not prove that software is defect-free. Even multiple testing can never ensure that software is 100% bug-free. Testing can reduce the number of defects but not removes all defects.

 Testing talks about the presence of defects and don’t talk about the absence of defects. i.e. Software Testing reduces the probability of undiscovered defects remaining in the software but even if no defects are found, it is not a proof of correctness.

But what if, you work extra hard, taking all precautions & make your software product 99% bug-free. And the software does not meet the needs & requirements of the clients.

* **Exhaustive testing is not possible:-**

Yes! Exhaustive testing is not possible. Instead, we need the optimal amount of testing based on the risk assessment of the application.

It is the process of testing the functionality of the software in all possible inputs (valid or invalid) and pre-conditions is known as exhaustive testing. Exhaustive testing is impossible means the software can never test at every test case. It can test only some test cases and assume that the software is correct and it will produce the correct output in every test case. If the software will test every test case then it will take more cost, effort, etc., and which is impractical.

In your opinion, Which operation is most likely to cause your Operating system to fail?

I am sure most of you would have guessed, Opening 10 different application all at the same time.

* **Early Testing:-**

Early Testing – Testing should start as early as possible in the Software Development Life Cycle. So that any defects in the requirements or design phase are captured in early stages. It is much cheaper to fix a Defect in the early stages of testing. But how early one should start testing? It is recommended that you start finding the bug the moment the requirements are defined.

To find the defect in the software, early test activity shall be started. The defect detected in the early phases of SDLC will be very less expensive. For better performance of software, software testing will start at the initial phase i.e. testing will perform at the requirement analysis phase.

* **Defect Clustering:-**

Defect Clustering which states that a small number of modules contain most of the defects detected. This is the application of the Pareto Principle to software testing: approximately 80% of the problems are found in 20% of the modules.

By experience, you can identify such risky modules. But this approach has its own problems

If the same tests are repeated over and over again, eventually the same test cases will no longer find new bugs.

A small number of the module can contain most of the defects. Pareto Principle to software testing state that 80% of software defect comes from 20% of modules.

* **Pesticide Paradox:-**

Repeating the same test cases, again and again, will not find new bugs. So it is necessary to review the test cases and add or update test cases to find new bugs.

Repetitive use of the same pesticide mix to eradicate insects during farming will over time lead to the insects developing resistance to the pesticide Thereby ineffective of pesticides on insects. The same applies to software testing. If the same set of repetitive tests are conducted, the method will be useless for discovering new defects.

To overcome this, the test cases need to be regularly reviewed & revised, adding new & different test cases to help find more defects.

Testers cannot simply depend on existing test techniques. He must look out continually to improve the existing methods to make testing more effective. But even after all this sweat & hard work in testing, you can never claim your product is bug-free.

* **Testing is context-dependent:-**

The testing approach depends on the context of the software developed. Different types of software need to perform different types of testing. For example, The testing of the e-commerce site is different from the testing of the Android application.

Testing is context dependent which basically means that the way you test an e-commerce site will be different from the way you test a commercial off the shelf application. All the developed software’s are not identical. You might use a different approach, methodologies, techniques, and types of testing depending upon the application type. For instance testing, any POS system at a retail store will be different than testing an ATM machine.

* **Absence of errors fallacy:-**

If a built software is 99% bug-free but it does not follow the user requirement then it is unusable. It is not only necessary that software is 99% bug-free but it is also mandatory to fulfill all the customer requirements.

It is possible that software which is 99% bug-free is still unusable. This can be the case if the system is tested thoroughly for the wrong requirement. Software testing is not mere finding defects, but also to check that software addresses the business needs. The absence of Error is a Fallacy i.e. Finding and fixing defects does not help if the system build is unusable and does not fulfill the user’s needs & requirements.

* **2.6 Software Testing Life Cycle (STLC):**

Software Testing Life Cycle process is an integral part of the Software Development Life Cycle. The overall aspect of STLC phase deals with testing and rectifying any error code generating within the program under various test conditions.

**What is STLC?**

STLC is simply a testing phase in the SDLC development. Validation and Authentication is tried and tested in this phase. The only limitation of this cycle is that it is limited to respective individual phase and is carried out by a group of skilled testers and technology evangelistic.

**Figure : Software Testing Life Cycle**

**STLC has below phases:**

**1. Requirement Analysis/Review:**

* This is a very important phase in STLC. Here the focus is on understanding the requirements of the system with the viewpoint of testing in mind.
* In this phase the QA interacts with the Business Analyst, System Analyst, Development Manager/Team Lead, etc. or if required the QA may also interact with Client to completely understand the requirements of the system.
* During this phase the QA takes many important decisions like what are the testing types & techniques to be performed, feasibility for automation testing implementation, etc.

**2. Test Planning:**

* In this phase the QA/QA Lead/QA Manager plans for the complete testing process. Important documents like Test Strategy, Test Plan and Effort Estimation are derived from this phase.
* Everything regarding testing like selection of the testing tools, test efforts estimations, planning resources, determining roles and responsibilities of the personnel involved in the process, planning for the training required, etc. are decided in this phase.
* This phase is very important as any small mistake in this phase can result in major issues in the project regarding time, money, efforts, etc.

**3. Test Designing:**

* Creation, Review & Update of Test Cases as well as Test Scripts are done in this phase. The test cases prepared by the QA team are reviewed and approved.
* Test data may also be created in this phase by the QA team if test environment is available to them.

**4. Test Environment Setup:**

* Test Environment is the actual system/environment/setup where the testing team will be testing the application. Test environment is prepared by understanding the required system architecture, software & hardware requirements, etc.
* Many times it happens that testing team is not involved in setting up the test environment. In such scenarios, it is preferable that the testing team should implement Smoke Testing to verify the readiness of the test environment before starting the actual testing.

**5. Test Execution:**

* The test cases which were prepared earlier are executed in this phase. In this phase, the testers test the websites. Different testing techniques as well as methods are implemented and executed on the software/application to break the system and find bugs.
* Bugs are reported to the development team. The development team resolves the bugs and the system is retested to ensure that it is bug free and ready to go live.

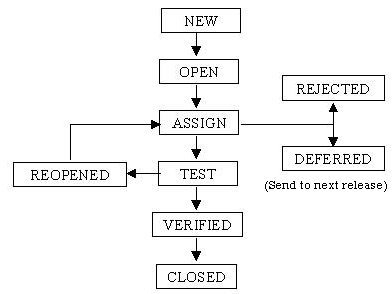
**6. Test Closure:**

* When the testing team is confident that all the reported bugs are resolved and the system is ready according to the client’s requirements, the software testing life cycle enters the last stage ie. Test Closure stage.
* In this stage, evaluation is done for the complete testing cycle, test closure reports are prepared, proper analysis and documentation is done for the major or critical bugs so that such situations can be handled efficiently and effectively in future projects, etc.

Each of these phases of STLC has their Entry & Exit criteria.

**Bug life cycle (Defect Life Cycle):**

In software development process, the bug or defect has its own bug life cycle. The bug or defect should go through the life cycle. The bug attains different states in the life cycle. Any bug to get fixed needs to pass from various phases of life cycle. The life cycle of the bug can be shown diagrammatically as follows,



**Figure: Bug life cycle**

**1. New**

When the bug is posted for the first time, its state will be “NEW”. This means that the bug is not yet approved and assigned.

**2. Open**

After a QA Engineer or tester has posted a bug, the lead of the tester approves that the bug is genuine and he changes the state as “OPEN”.

**3. Assign**

Once the lead engineer changes the state as “OPEN”, he assigns the bug to corresponding developer or developer team. The state of the bug now is changed to “ASSIGN”.

**4. Test**

Once the developer fixes the bug, he has to assign the bug to the testing team for next round of testing. Before he releases the software with bug fixed, he changes the state of bug to “TEST”. It specifies that the bug has been fixed and is released to testing team for next round of testing.

**5. Deferred**

The bug, changed to deferred state means the bug is expected to be fixed in next releases instead of in the current candidate build. The reasons for changing the bug to Deferred state have many factors like priority of the bug may be low, lack of time for the release or the bug may not have major effect on the software.

**6. Rejected**

If the developer feels that the bug is not valid or genuine, he rejects the bug. Then the state of the bug is changed to “REJECTED”.

**7. Duplicate**

If the bug is repeated twice or the two bugs mention the same concept of the bug, then one bug status is changed to “DUPLICATE”.

**8. Verified**

Once the bug is fixed and the status is changed to “TEST”, the tester tests the bug. If the bug is not present in the software, he approves that the bug is fixed and changes the status to “VERIFIED”.

**9. Reopened**

If the bug still exists even after the bug is fixed by the developer, the tester changes the status to “REOPENED”. The bug traverses the life cycle once again.

**10. Closed**

Once the bug is fixed, it is tested by the tester. If the tester feels that the bug no longer exists in the software, he changes the status of the bug to “CLOSED”. This state means that the bug is fixed, tested and approved.

2.7 **Verification & Validation Concepts:**

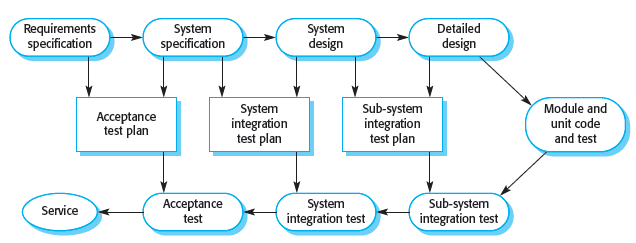
* Verification and validation are the processes in which we check a product against its specifications and the expectations of the users who will be using it. According to a known software engineering expert Berry Boehm, verification and validation are:
* **Verification**
* Does the product meet system specifications?
* Have you built the product right?
* Verification is the process of evaluating a system under development to determine that whether the system satisfy the conditions which were imposed at starting phase of system development
* **Validation**
* Does the product meet user expectations?
* Have you built the right product?
* Validation is the process of determining the correctness of the system w.r.t. user’s requirements.
* V & V is a process & uses two basic techniques for system checking & analysis:
* Software Inspections: Static analysis
* Software Testing: Dynamic Analysis
* **Testing:**
* Testing is the process of exercising the software to verify that it satisfies the specified users requirements and to detect faults.

**Verification & Validation Planning:**

* Verification and validation is an expensive process. For some systems, such as real-time

systems with complex non-functional constraints, more than half the system development budget may be spent on V & V.

* Careful planning is needed to get the most out of inspections and testing and to control the costs of the verification and validation process.
* You should start planning system validation and verification early in the development process.



**Figure : Test plans as a link between development and testing**

* The software development process model shown in above Figure is sometimes called the V-model.
* It is an instantiation of the generic waterfall model and shows that test plans should be derived from the system specification and design.
* This model also breaks down system V & V into a number of stages. Each stage is driven by tests that have been defined to check the conformance of the program with its design and specification.
* As part of the V & V planning process, you should decide on the balance between static and dynamic approaches to verification and validation, draw up standards and procedures for software inspections and testing, establish checklists to drive program inspections and define the software test plan.
* Test planning is concerned with establishing standards for the testing process, not just with describing product tests.
* It helps to managers allocate resources and estimate testing schedules, test plans are intended for software engineers involved in designing and carrying out system tests.
* The major components of a test plan for a large and complex system are shown in below figure. As well as setting out the testing schedule and procedures, the test plan defines the hardware and software resources that are required.

The V-model is a type of SDLC model where process executes in a sequential manner in V-shape. It is also known as Verification and Validation model. It is based on the association of a testing phase for each corresponding development stage. Development of each step directly associated with the testing phase. The next phase starts only after completion of the previous phase i.e. for each development activity, there is a testing activity corresponding to it.



* The left side of the model is Software Development Life Cycle – **SDLC**
* The right side of the model is Software Test Life Cycle – **STLC**
* The entire figure looks like a V, hence the name **V – model**

**Verification:** It involves static analysis technique (review) done without executing code. It is the process of evaluation of the product development phase to find whether specified requirements meet.

**Validation:** It involves dynamic analysis technique (functional, non-functional), testing done by executing code. Validation is the process to evaluate the software after the completion of the development phase to determine whether software meets the customer expectations and requirements.

**V-Model - Design**

Under the V-Model, the corresponding testing phase of the development phase is planned in parallel. So, there are Verification phases on one side of the ‘V’ and Validation phases on the other side. The Coding Phase joins the two sides of the V-Model.

The following illustration depicts the different phases in a V-Model of the SDLC.





**V-Model - Verification Phases**

There are several Verification phases in the V-Model, each of these are explained in detail below.

**Business Requirement Analysis**

This is the first phase in the development cycle where the product requirements are understood from the customer’s perspective. This phase involves detailed communication with the customer to understand his expectations and exact requirement. This is a very important activity and needs to be managed well, as most of the customers are not sure about what exactly they need. The **acceptance test design planning** is done at this stage as business requirements can be used as an input for acceptance testing.

**System Design**

Once you have the clear and detailed product requirements, it is time to design the complete system. The system design will have the understanding and detailing the complete hardware and communication setup for the product under development. The system test plan is developed based on the system design. Doing this at an earlier stage leaves more time for the actual test execution later.

**Architectural Design**

Architectural specifications are understood and designed in this phase. Usually more than one technical approach is proposed and based on the technical and financial feasibility the final decision is taken. The system design is broken down further into modules taking up different functionality. This is also referred to as **High Level Design (HLD)**

.

The data transfer and communication between the internal modules and with the outside world (other systems) is clearly understood and defined in this stage. With this information, integration tests can be designed and documented during this stage.

**Module Design**

In this phase, the detailed internal design for all the system modules is specified, referred to as **Low Level Design (LLD)**. It is important that the design is compatible with the other modules in the system architecture and the other external systems. The unit tests are an essential part of any development process and helps eliminate the maximum faults and errors at a very early stage. These unit tests can be designed at this stage based on the internal module designs.

**Coding Phase**

The actual coding of the system modules designed in the design phase is taken up in the Coding phase. The best suitable programming language is decided based on the system and architectural requirements.

The coding is performed based on the coding guidelines and standards. The code goes through numerous code reviews and is optimized for best performance before the final build is checked into the repository.

**Validation Phases**

The different Validation Phases in a V-Model are explained in detail below.

**Unit Testing**

Unit tests designed in the module design phase are executed on the code during this validation phase. Unit testing is the testing at code level and helps eliminate bugs at an early stage, though all defects cannot be uncovered by unit testing.

**Integration Testing**

Integration testing is associated with the architectural design phase. Integration tests are performed to test the coexistence and communication of the internal modules within the system.

**System Testing**

System testing is directly associated with the system design phase. System tests check the entire system functionality and the communication of the system under development with external systems. Most of the software and hardware compatibility issues can be uncovered during this system test execution.

**Acceptance Testing**

Acceptance testing is associated with the business requirement analysis phase and involves testing the product in user environment. Acceptance tests uncover the compatibility issues with the other systems available in the user environment. It also discovers the non-functional issues such as load and performance defects in the actual user environment.

**V- Model ─ Application**

V- Model application is almost the same as the waterfall model, as both the models are of sequential type. Requirements have to be very clear before the project starts, because it is usually expensive to go back and make changes. This model is used in the medical development field, as it is strictly a disciplined domain.

The following pointers are some of the most suitable scenarios to use the V-Model application.

* Requirements are well defined, clearly documented and fixed.
* Product definition is stable.
* Technology is not dynamic and is well understood by the project team.
* There are no ambiguous or undefined requirements.
* The project is short.

**V-Model - Pros and Cons**

The advantage of the V-Model method is that it is very easy to understand and apply. The simplicity of this model also makes it easier to manage. The disadvantage is that the model is not flexible to changes and just in case there is a requirement change, which is very common in today’s dynamic world, it becomes very expensive to make the change.

**The advantages of the V-Model method are as follows −**

* This is a highly-disciplined model and Phases are completed one at a time.
* Works well for smaller projects where requirements are very well understood.
* Simple and easy to understand and use.
* Easy to manage due to the rigidity of the model. Each phase has specific deliverables and a review process.

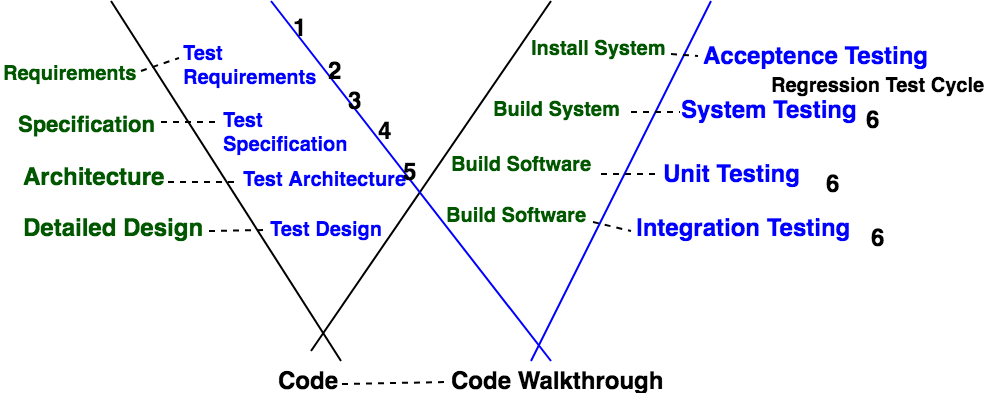
**The disadvantages of the V-Model method are as follows −**

* High risk and uncertainty.
* Not a good model for complex and object-oriented projects.
* Poor model for long and ongoing projects.
* Not suitable for the projects where requirements are at a moderate to high risk of changing.
* Once an application is in the testing stage, it is difficult to go back and change a functionality.
* No working software is produced until late during the life cycle.

**Paul Herzlich** introduced **W-Model** in 1993.

**W-model** is the most recent software development model where we start real testing activity simultaneously software development process starts. Where as software development process is a method in which a software or product is made through various stages of planning, development and testing before the final software or product is delivered. testing is such a stage that is extremely crucial to ensure the delivery of an optimum quality product.

* V-model and W-model are two of the most important models that are used in software testing.
* W-Model covers those activities which are skipped by V-Model and also, it deals with problems which couldn’t be catch by V-Model.
* W-Model approach attempts to address and tackle the shortcomings W-Model approach attempts to address and tackle the shortcomings of V-Model.
* W-model can be done only once the development of the product is complete with no modifications required to be done in between. This type of testing is most suitable for short projects.
* With the help of W-Model, we ensure that the testing of the product starts from the very first day of the inception of product and each phase of the product development is verified and validated.



**Phases of W-Model:**

Each phase is verified/validated. Dotted line shows that every phase in green is validated/tested through every phase in sky blue. Now, in the above figure,

* Point 1 refers to – Build Test Plan & Test Strategy.
* Point 2 refers to – Scenario Identification.
* Point 3 refers to –Test case preparation from Specification document and design documents.
* Point 4 refers to – Test case preparation from Specification document and design documents.
* Point 5 refers to – review of test cases and update as per the review comments.
* Point 6 refers to – Various testing methodologies such as Unit/integration testing, path testing, equivalence partition, boundary value, specification based testing, security testing, usability testing, performance testing.
* After this, there are regression test cycles and then User acceptance testing.

**Testing Techniques Used in W-Model:**

* + 1. Regression Testing
    2. Static Testing:

Static Testing is further divided into two parts:

* + - * **(a)** Review
      * **(b)** Static Analysis
    1. Dynamic Testing

**Advantages of W-Model:**

* + 1. In W-Model there is no strict division between constructive tasks on the left-hand side and the more destructive tasks on the right-hand side.
    2. During the test phase, the developer is responsible for the removal of defects and the correction of the implementation.
    3. Emphasis the fact that testing is more than just construction, execution and evaluation of test cases.
    4. The importance of the tests and the ordering of the individual activities for testing are clear.

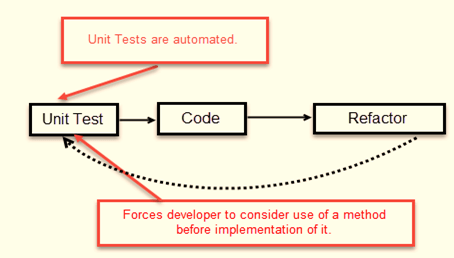
**Disadvantages of W-Model:**

* + 1. The real facts are simplified in this model.
    2. There is a need for a simple model if all people involved in a project are to accept it.
    3. For highly critical applications the test activities certainly have higher weighting or at least equal weighting with other activities.

## **What is Test Driven Development(TDD)?**

**Test Driven Development (TDD)** is software development approach in which test cases are developed to specify and validate what the code will do. In simple terms, test cases for each functionality are created and tested first and if the test fails then the new code is written in order to pass the test and making code simple and bug-free.

Test-Driven Development starts with designing and developing tests for every small functionality of an application. TDD framework instructs developers to write new code only if an automated test has failed. This avoids duplication of code. The TDD full form is Test-driven development.



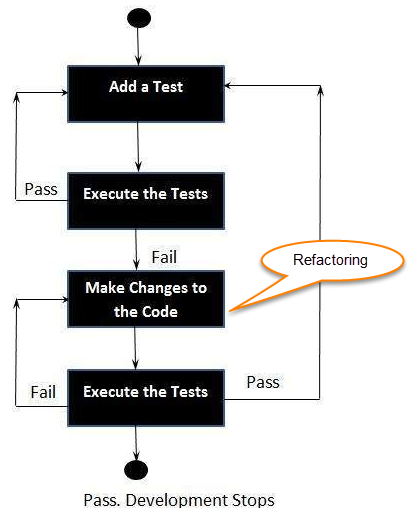
The simple concept of TDD is to write and correct the failed tests before writing new code (before development). This helps to avoid duplication of code as we write a small amount of code at a time in order to pass tests. (Tests are nothing but requirement conditions that we need to test to fulfill them).

Test-Driven development is a process of developing and running automated test before actual development of the application. Hence, TDD sometimes also called as **Test First Development.**

**How to perform TDD Test**

Following steps define how to perform TDD test,

1. Add a test.
2. Run all tests and see if any new test fails.
3. Write some code.
4. Run tests and Refactor code.
5. Repeat.



**Fig. :- Five Steps of Test-Driven Development**

* **2.9 Levels of Testing –**
* **2.9.1 Unit (Component) Testing:-**

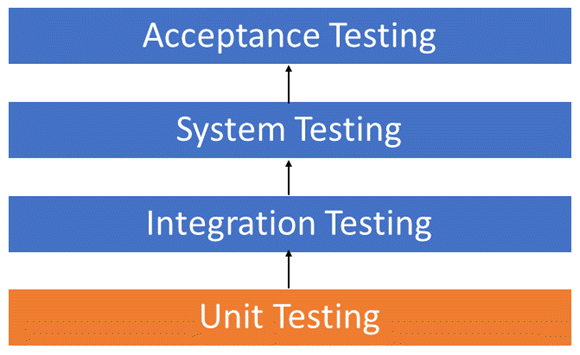
**UNIT TESTING** is a type of software testing where individual units or components of software are tested. The purpose is to validate that each unit of the software code performs as expected. Unit Testing is done during the development (coding phase) of an application by the developers. Unit Tests isolate a section of code and verify its correctness. A unit may be an individual function, method, procedure, module, or object.

In SDLC, STLC, V Model, Unit testing is first level of testing done before integration testing. Unit testing is a White Box testing technique that is usually performed by the developer. Though, in a practical world due to time crunch or reluctance of developers to tests, QA engineers also do unit testing.

**Why Unit Testing?**

**Unit Testing** is important because software developers sometimes try saving time doing minimal unit testing and this is myth because inappropriate unit testing leads to high cost[Defect](https://www.guru99.com/defect-management-process.html)fixing during [System Testing](https://www.guru99.com/system-testing.html), [Integration Testing](https://www.guru99.com/integration-testing.html) and even Beta Testing after application is built. If proper unit testing is done in early development, then it saves time and money in the end.

Here, are the key reasons to perform unit testing in software engineering:



1. Unit tests help to fix bugs early in the development cycle and save costs.
2. It helps the developers to understand the testing code base and enables them to make changes quickly
3. Good unit tests serve as project documentation
4. Unit tests help with code re-use. Migrate both your code **and** your tests to your new project. Tweak the code until the tests run again.

**Unit Testing is of two types**

* Manual
* Automated

Unit testing is commonly automated but may still be performed manually. Software Engineering does not favor one over the other but automation is preferred. A manual approach to unit testing may employ a step-by-step instructional document.

**Under the automated approach-**

* A developer writes a section of code in the application just to test the function. They would later comment out and finally remove the test code when the application is deployed.
* A developer could also isolate the function to test it more rigorously. This is a more thorough unit testing practice that involves copy and paste of code to its own testing environment than its natural environment. **Isolating the code helps in revealing unnecessary dependencies between the code being tested and other units or data spaces** in the product. These dependencies can then be eliminated.
* A coder generally uses a UnitTest Framework to develop automated test cases. Using an automation framework, the developer codes criteria into the test to verify the correctness of the code. During execution of the test cases, the framework logs failing test cases. Many frameworks will also automatically flag and report, in summary, these [failed test cases](https://www.guru99.com/run-failed-test-cases-in-testng.html). Depending on the severity of a failure, the framework may halt subsequent testing.
* The workflow of Unit Testing is 1) Create Test Cases 2) Review/Rework 3) Baseline 4) Execute Test Cases.

**Unit Testing Techniques:-**

The **Unit Testing Techniques** are mainly categorized into three parts which are Black box testing that involves testing of user interface along with input and output, White box testing that involves testing the functional behaviour of the software application and Gray box testing that is used to execute test suites, test methods, test cases and performing risk analysis.

Code coverage techniques used in Unit Testing are listed below:

* Statement Coverage
* Decision Coverage
* Branch Coverage
* Condition Coverage
* Finite State Machine Coverage

**Unit Test Example: Mock Objects**

Unit testing relies on mock objects being created to test sections of code that are not yet part of a complete application. Mock objects fill in for the missing parts of the program.

For example, you might have a function that needs variables or objects that are not created yet. In unit testing, those will be accounted for in the form of mock objects created solely for the purpose of the unit testing done on that section of code.

**Unit Testing Advantage**

* Developers looking to learn what functionality is provided by a unit and how to use it can look at the unit tests to gain a basic understanding of the unit API.
* Unit testing allows the programmer to refactor code at a later date, and make sure the module still works correctly (i.e. Regression testing). The procedure is to write test cases for all functions and methods so that whenever a change causes a fault, it can be quickly identified and fixed.
* Due to the modular nature of the unit testing, we can test parts of the project without waiting for others to be completed.

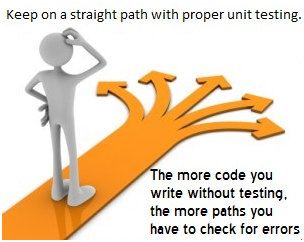
**Unit Testing Disadvantages**

* Unit testing can’t be expected to catch every error in a program. It is not possible to evaluate all execution paths even in the most trivial programs
* Unit testing by its very nature focuses on a unit of code. Hence it can’t catch integration errors or broad system level errors.

It’s recommended unit testing be used in conjunction with other testing activities.

**Unit Testing Best Practices**

* Unit Test cases should be independent. In case of any enhancements or change in requirements, unit test cases should not be affected.
* Test only one code at a time.
* Follow clear and consistent naming conventions for your unit tests
* In case of a change in code in any module, ensure there is a corresponding unit[Test Case](https://www.guru99.com/test-case.html)for the module, and the module passes the tests before changing the implementation
* Bugs identified during unit testing must be fixed before proceeding to the next phase in SDLC
* Adopt a “test as your code” approach. The more code you write without testing, the more paths you have to check for errors.



**Unit Testing Tools**

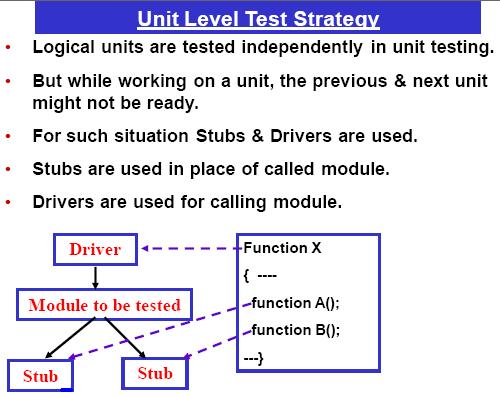
There are several automated unit test software available to assist with unit testing. We will provide a few examples below:

1. [Junit](https://www.guru99.com/junit-tutorial.html): Junit is a free to use testing tool used for Java programming language.  It provides assertions to identify test method. This tool test data first and then inserted in the piece of code.
2. [NUnit](https://nunit.org/):  NUnit is widely used unit-testing framework use for all .net languages.  It is an open source tool which allows writing scripts manually. It supports data-driven tests which can run in parallel.
3. [JMockit](https://jmockit.github.io/index.html):  JMockit is open source Unit testing tool.  It is a code coverage tool with line and path metrics. It allows mocking API with recording and verification syntax. This tool offers Line coverage, Path Coverage, and Data Coverage.
4. [EMMA](http://emma.sourceforge.net/):  EMMA is an open-source toolkit for analyzing and reporting code written in Java language. Emma support coverage types like method, line, basic block. It is Java-based so it is without external library dependencies and can access the source code.
5. [PHP Unit](https://phpunit.de/): PHP Unit is a unit testing tool for PHP programmer. It takes small portions of code which is called units and test each of them separately.  The tool also allows developers to use pre-define assertion methods to assert that a system behave in a certain manner.

Those are just a few of the available unit testing tools. There are lots more, especially for C languages and Java, but you are sure to find a unit testing tool for your programming needs regardless of the language you use.

**Summary**

* UNIT TESTING is defined as a type of software testing where individual units or components of a software are tested.
* As you can see, there can be a lot involved in unit testing. It can be complex or rather simple depending on the application being tested and the testing strategies, tools and philosophies used.
* Unit testing is a software testing method by which individual units of source code, sets of one or more computer program modules together with associated control data, usage procedures, and operating procedures are tested to determine if they are fit for use.
* Intuitively, one can view a unit as the smallest testable part of an application.
* In procedural programming, a unit could be an entire module, but it is more commonly an individual function or procedure.
* In object-oriented programming, a unit is often an entire interface, such as a class, but could be an individual method.
* Unit tests are short code fragments created by programmers or occasionally by white box testers during the development process.
* Unit tests are typically written and run by software developers to ensure that code meets its design and behaves as intended.
* Lowest level of testing.
* Also known as component testing, module testing.
* Tested in isolation
* A unit can be any smallest component of system like page, menu, module etc.
* Its objective is “To test a software unit for its primary attributes & behavior.”
* Its testing coverage are :
* Validation Rules
* Navigation Requirements
* User Interface
* Processing of the Unit
* Logical Path
* IF-THEN-ELSE
* DO-WHILE
* Checklist for Unit testing
* At Unit level testing tester has to check :
* Text,
* Numeric,
* Date
* Rich text
* Time
* Currency



* **2.9.2 :- Integration Testing :-**

INTEGRATION TESTING is defined as a type of testing where software modules are integrated logically and tested as a group. A typical software project consists of multiple software modules, coded by different programmers. The purpose of this level of testing is to expose defects in the interaction between these software modules when they are integrated

Integration Testing focuses on checking data communication amongst these modules. Hence it is also termed as ‘I & T’ (Integration and Testing), ‘String Testing’ and sometimes ‘Thread Testing’.

**Example of Integration Test Case**

Integration[Test Case](https://www.guru99.com/test-case.html)differs from other test cases in the sense it**focuses mainly on the interfaces & flow of data/information between the modules**. Here priority is to be given for the **integrating links** rather than the unit functions which are already tested.

Sample Integration Test Cases for the following scenario: Application has 3 modules say ‘Login Page’, ‘Mailbox’ and ‘Delete emails’ and each of them is integrated logically.

Here do not concentrate much on the Login Page testing as it’s already been done in [Unit Testing](https://www.guru99.com/unit-testing-guide.html). But check how it’s linked to the Mail Box Page.

Similarly Mail Box: Check its integration to the Delete Mails Module.

| **Test Case ID** | **Test Case Objective** | **Test Case Description** | **Expected Result** |
| --- | --- | --- | --- |
| **1** | Check the interface link between the Login and Mailbox module | Enter login credentials and click on the Login button | To be directed to the Mail Box |
| **2** | Check the interface link between the Mailbox and Delete Mails Module | From Mailbox select the email and click a delete button | Selected email should appear in the Deleted/Trash folder |

**Approaches, Strategies, Methodologies of Integration Testing:-**

Software Engineering defines variety of strategies to execute Integration testing, viz.

* Big Bang Approach :
* Incremental Approach: which is further divided into the following
  + Top Down Approach
  + Bottom Up Approach
  + Sandwich Approach – Combination of Top Down and Bottom Up

**Big Bang Testing:-**

**Big Bang Testing** is an Integration testing approach in which all the components or modules are integrated together at once and then tested as a unit. This combined set of components is considered as an entity while testing. If all of the components in the unit are not completed, the integration process will not execute.

**Advantages:**

* Convenient for small systems.

**Disadvantages:**

* Fault Localization is difficult.
* Given the sheer number of interfaces that need to be tested in this approach, some interfaces link to be tested could be missed easily.
* Since the Integration testing can commence only after “all” the modules are designed, the testing team will have less time for execution in the testing phase.
* Since all modules are tested at once, high-risk critical modules are not isolated and tested on priority. Peripheral modules which deal with user interfaces are also not isolated and tested on priority.

**Incremental Testing**

In the **Incremental Testing** approach, testing is done by integrating two or more modules that are logically related to each other and then tested for proper functioning of the application. Then the other related modules are integrated incrementally and the process continues until all the logically related modules are integrated and tested successfully.

Incremental Approach, in turn, is carried out by two different Methods:

* Bottom Up
* Top Down

**Stubs and Drivers**

**Stubs and Drivers** are the dummy programs in Integration testing used to facilitate the software testing activity. These programs act as a substitutes for the missing models in the testing. They do not implement the entire programming logic of the software module but they simulate data communication with the calling module while testing.

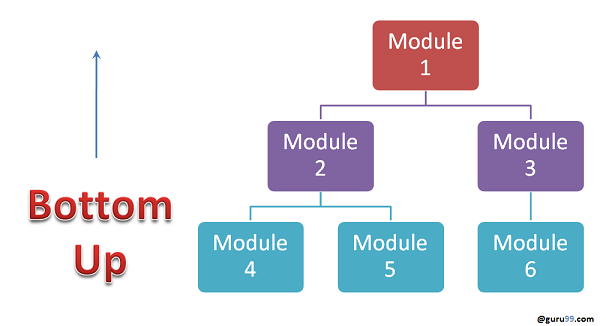
**Stub**: Is called by the Module under Test.

**Driver**: Calls the Module to be tested.

**Bottom-up Integration Testing**

**Bottom-up Integration Testing** is a strategy in which the lower level modules are tested first. These tested modules are then further used to facilitate the testing of higher level modules. The process continues until all modules at top level are tested. Once the lower level modules are tested and integrated, then the next level of modules are formed.

**Diagrammatic Representation**:



**Advantages:**

* Fault localization is easier.
* No time  is wasted waiting for all modules to be developed unlike Big-bang approach

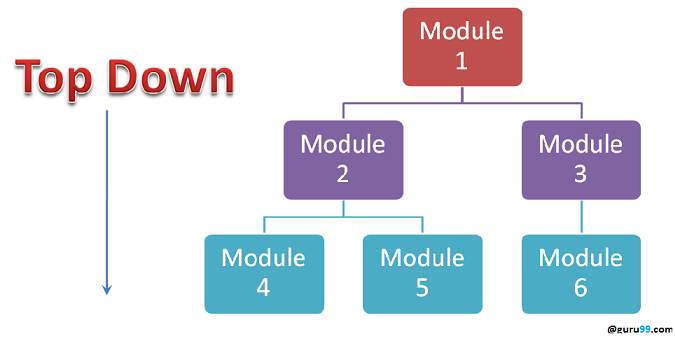
**Disadvantages:**

* Critical modules (at the top level of software architecture) which control the flow of application are tested last and may be prone to defects.
* An early prototype is not possible

**Top-down Integration Testing:-**

**Top Down Integration Testing** is a method in which integration testing takes place from top to bottom following the control flow of software system. The higher level modules are tested first and then lower level modules are tested and integrated in order to check the software functionality. Stubs are used for testing if some modules are not ready.

**Diagrammatic Representation:**



**Advantages:**

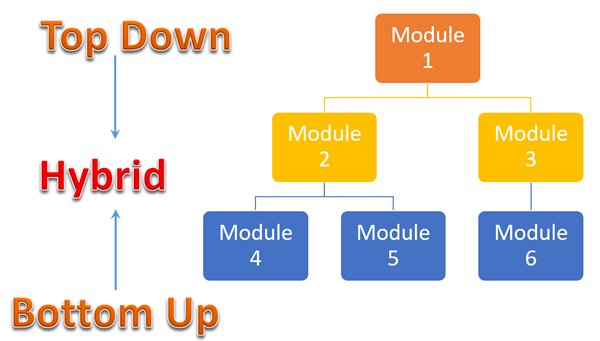
* Fault Localization is easier.
* Possibility to obtain an early prototype.
* Critical Modules are tested on priority; major design flaws could be found and fixed first.

**Disadvantages:**

* Needs many Stubs.
* Modules at a lower level are tested inadequately.

**Sandwich Testing:-**

**Sandwich Testing** is a strategy in which top level modules are tested with lower level modules at the same time lower modules are integrated with top modules and tested as a system. It is a combination of Top-down and Bottom-up approaches therefore it is called **Hybrid Integration Testing**. It makes use of both stubs as well as drivers.



**How to do Integration Testing?**

The Integration test procedure irrespective of the Software testing strategies (discussed above):

1. Prepare the Integration Tests Plan
2. Design the Test Scenarios, Cases, and Scripts.
3. Executing the test Cases followed by reporting the defects.
4. Tracking & re-testing the defects.
5. Steps 3 and 4 are repeated until the completion of Integration is successful.

**Brief Description of Integration Test Plans:**

It includes the following attributes:

* Methods/Approaches to testing (as discussed above).
* Scopes and Out of Scopes Items of Integration Testing.
* Roles and Responsibilities.
* Pre-requisites for Integration testing.
* Testing environment.
* Risk and Mitigation Plans.

**Entry and Exit Criteria of Integration Testing**

Entry and Exit Criteria to Integration testing phase in any software development model

**Entry Criteria:**

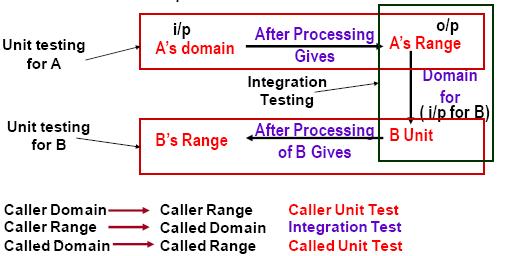
* Unit Tested Components/Modules
* All High prioritized bugs fixed and closed
* All Modules to be code completed and integrated successfully.
* Integration tests Plan, test case, scenarios to be signed off and documented.
* Required[Test Environment](https://www.guru99.com/test-environment-software-testing.html)to be set up for Integration testing

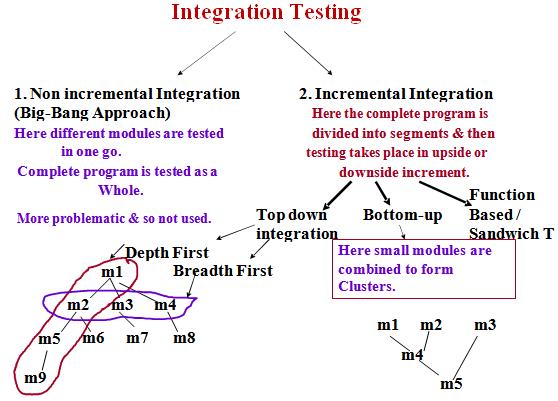
**Exit Criteria:**

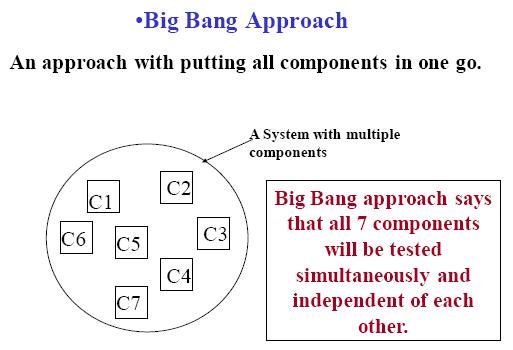
* Successful Testing of Integrated Application.
* Executed Test Cases are documented
* All High prioritized bugs fixed and closed
* Technical documents to be submitted followed by release Notes.

**Best Practices/ Guidelines for Integration Testing**

* First, determine the Integration[Test Strategy](https://www.guru99.com/how-to-create-test-strategy-document.html)that could be adopted and later prepare the test cases and test data accordingly.
* Study the Architecture design of the Application and identify the Critical Modules. These need to be tested on priority.
* Obtain the interface designs from the Architectural team and create test cases to verify all of the interfaces in detail. Interface to database/external hardware/software application must be tested in detail.
* After the test cases, it’s the test data which plays the critical role.
* Always have the mock data prepared, prior to executing. Do not select test data while executing the test cases.
* Integration testing (sometimes called integration and testing, abbreviated I&T) is the phase in software testing in which individual software modules are combined and tested as a group.
* It occurs after unit testing and before validation testing.
* Integration testing takes as its input modules that have been unit tested, groups them in larger aggregates, applies tests defined in an integration test plan to those aggregates, and delivers as its output the integrated system ready for system testing.
* The purpose of integration testing is to verify functional, performance, and reliability requirements placed on major design items.
* These "design items", i.e. assemblages (or groups of units), are exercised through their interfaces using black box testing, success and error cases being simulated via appropriate parameter and data inputs.
* Simulated usage of shared data areas and inter-process communication is tested and individual subsystems are exercised through their input interface.
* Test cases are constructed to test whether all the components within assemblages interact correctly, for example across procedure calls or process activations, and this is done after testing individual modules, i.e. unit testing.
* The overall idea is a "building block" approach, in which verified assemblages are added to a verified base which is then used to support the integration testing of further assemblages.
* Types of integration testing are big bang, top-down, and bottom-up.
* Other Integration Patterns are: Collaboration Integration, Backbone Integration, Layer Integration, Client/Server Integration, Distributed Services Integration and High-frequency Integration.
* Here the attention is focused on the information domain.
* Domain & range are two terms used to define this concept
* The set of output values produced by a function is called as the range of that function.
* **Domain is the set of input values over which function is defined.**



****

****

**Incremental Approach**

* One by one component is added to the base line for further testing. Base line is formed with minimum components.
* Incremental Approach types are

1. Top down Approach

2. Bottom up Approach

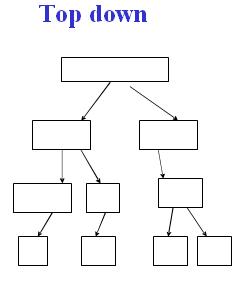
3. Functional Increment

3.1 Minimum capability Integration

3.2 Thread Integration

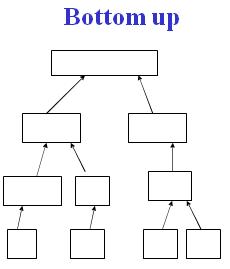
**1. Top down Approach / Testing:**

* This approach starts combining components from the highest level.
* Since it starts from top, so there will be missing pieces in tree, not integrated yet.
* For this top down uses STUBS for the missing components.
* STUB is a small self contained do-nothing program that simply returns its name.
* Stub replaces the called component in Integration.
* Top Down Testing is an approach to integrated testing where the top integrated modules are tested and the branch of the module is tested step by step until the end of the related module.



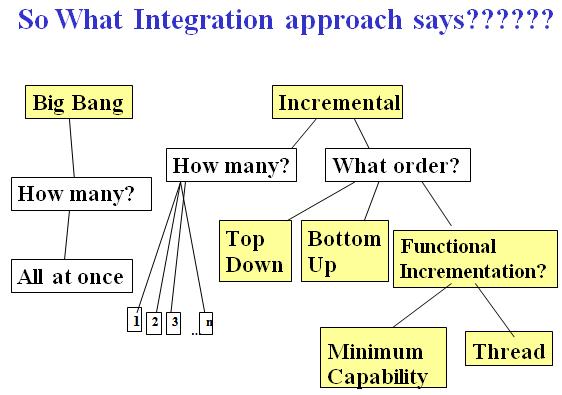
**2. Bottom Up Approach / Testing:**

* All components are integrated at lowest level of hierarchy.
* In this case calling structure is missing, so dummy calling modules are required to be written.
* This calling component is called as DRIVER
* Drivers are small programs used to drive the base line.
* Drivers are also called as test.
* Bottom Up Testing is an approach to integrated testing where the lowest level components are tested first, then used to facilitate the testing of higher level components.
* The process is repeated until the component at the top of the hierarchy is tested.
* All the bottom or low-level modules, procedures or functions are integrated and then tested.
* After the integration testing of lower level integrated modules, the next level of modules will be formed and can be used for integration testing.
* This approach is helpful only when all or most of the modules of the same development level are ready.
* This method also helps to determine the levels of software developed and makes it easier to report testing progress in the form of a percentage.



**3. Functional Increment:**

* It is a functional integration strategy with integration of modules required to achieve the basic functionality with minimum number of components to integrate together.
* Thread Integration:
* It is the capability with minimum number of modules integration with respect to time and history.



* **2.9.3 System Testing :-**

**System Testing** is a level of testing that validates the complete and fully integrated software product. The purpose of a system test is to evaluate the end-to-end system specifications. Usually, the software is only one element of a larger computer-based system. Ultimately, the software is interfaced with other software/hardware systems. System Testing is actually a series of different tests whose sole purpose is to exercise the full computer-based system.

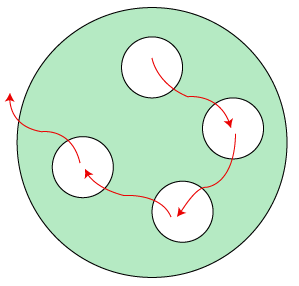
System Testing includes testing of a fully integrated software system. Generally, a computer system is made with the integration of software (any software is only a single element of a computer system). The software is developed in units and then interfaced with other software and hardware to create a complete computer system. In other words, a computer system consists of a group of software to perform the various tasks, but only software cannot perform the task; for that software must be interfaced with compatible hardware. System testing is a series of different type of tests with the purpose to exercise and examine the full working of an integrated software computer system against requirements.

To check the end-to-end flow of an application or the software as a user is known as System testing. In this, we navigate (go through) all the necessary modules of an application and check if the end features or the end business works fine, and test the product as a whole system.

**System Testing** is a testing technique performed to evaluate the complete system the system's compliance against specified requirements. In System testing, the functionalities of the system are tested from an end-to-end perspective.

System Testing is usually carried out by a team that is independent of the development team in order to measure the quality of the system unbiased. It includes both functional and Non-Functional testing.

It is end-to-end testing where the testing environment is similar to the production environment.



Suppose we open an application, let say www.rediff.com, and there we can see that an advertisement is displayed on the top of the homepage, and it remains there for a few seconds before it disappears. These types of Ads are done by the Advertisement Management System (AMS). Now, we will perform system testing for this type of field.

**The below application works in the following manner:**

Let's say that Amazon wants to display a promotion ad on January 26 at precisely 10:00 AM on the Rediff's home page for the country India.

Then, the sales manager logs into the website and creates a request for an advertisement dated for the above day.

He/she attaches a file that likely an image files or the video file of the AD and applies.

The next day, the AMS manager of Rediffmail login into the application and verifies the awaiting Ad request.

The AMS manager will check those Amazons ad requests are pending, and then he/she will check if the space is available for the particular date and time.

If space is there, then he/she evaluate the cost of putting up the Ad at 15$ per second, and the overall Ad cost for 10 seconds is approximate 150$.

The AMS manager clicks on the payment request and sends the estimated value along with the request for payment to the Amazon manager.

Then the amazon manager login into the Ad status and confirms the payment request, and he/she makes the payment as per all the details and clicks on the Submit and Pay

As soon as Rediff's AMs manager gets the amount, he/she will set up the Advertisement for the specific date and time on the Rediffmail's home page.

The various system test scenarios are as follows:

**Scenario1**: The first test is the general scenario, as we discussed above. The test engineer will do the system testing for the underlying situation where the Amazon manager creates a request for the Ad and that Ad is used at a particular date and time.

**Scenario2:** Suppose the Amazon manager feels that the AD space is too expensive and cancels the request. At the same time, the Flipkart requests the Ad space on January 26 at 10:00 AM. Then the request of Amazon has been canceled. Therefore, Flipkart's promotion ad must be arranged on January 26 at 10 AM.

* **2.9.4 User Acceptance Testing (UAT) :-**

**User Acceptance Testing (UAT)** is a type of testing performed by the end user or the client to verify/accept the software system before moving the software application to the production environment. UAT is done in the final phase of testing after functional, integration and system testing is done.

Acceptance testing is formal testing based on user requirements and function processing. It determines whether the software is conforming specified requirements and user requirements or not. It is conducted as a kind of Black Box testing where the number of required users involved testing the acceptance level of the system. It is the fourth and last level of software testing.

User acceptance testing (UAT) is a type of testing, which is done by the customer before accepting the final product. Generally, UAT is done by the customer (domain expert) for their satisfaction, and check whether the application is working according to given business scenarios, real-time scenarios.

In this, we concentrate only on those features and scenarios which are regularly used by the customer or mostly user scenarios for the business or those scenarios which are used daily by the end-user or the customer.

However, the software has passed through three testing levels (Unit Testing, Integration Testing, System Testing) But still there are some minor errors which can be identified when the system is used by the end user in the actual scenario.

**Note:**

It is done in the separate environment at the customer place, which is known as the UAT environment. The user acceptance testing is done by a different team called as domain expert who is known to the application.

Generally, small companies do not have a domain expert because there is no frequent changes happen in the application.

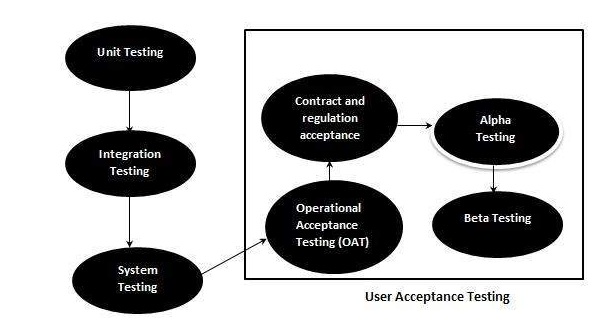
User acceptance testing, a testing methodology where the clients/end users involved in testing the product to validate the product against their requirements. It is performed at client location at developer's site.

For industry such as medicine or aviation industry, contract and regulatory compliance testing and operational acceptance testing is also carried out as part of user acceptance testing.

UAT is context dependent and the UAT plans are prepared based on the requirements and NOT mandatory to execute all kinds of user acceptance tests and even coordinated and contributed by testing team.

## **User Acceptance Testing - In SDLC**

The following diagram explains the fitment of user acceptance testing in the software development life cycle:



The acceptance test cases are executed against the test data or using an acceptance test script and then the results are compared with the expected ones.

## **Acceptance Criteria**

Acceptance criteria are defined on the basis of the following attributes:

* Functional Correctness and Completeness
* Data Integrity
* Data Conversion
* Usability
* Performance
* Timeliness
* Confidentiality and Availability
* Installability and Upgradability
* Scalability
* Documentation

## **Acceptance Test Plan - Attributes**

The acceptance test activities are carried out in phases. Firstly the basic tests are executed and if the test results are satisfactory then the execution of more complex scenarios are carried out.

The Acceptance test plan has the following attributes

* Introduction
* Acceptance Test Category
* operation Environment
* Test case ID
* Test Title
* Test Objective
* Test Procedure
* Test Schedule
* Resources

The acceptance test activities are designed to reach at one of the conclusions :

1. Accept the system as delivered
2. Accept the system after the requested modifications have been made
3. Do not accept the system

**Acceptance Test Report - Attributes**

The Acceptance test Report has the following attributes:

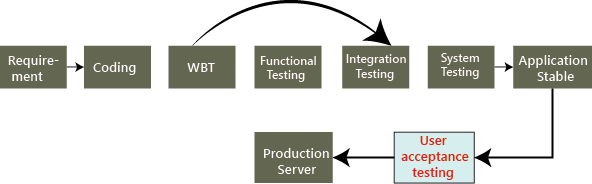
* Report Identifier
* Summary of Results
* Variations
* Recommendations
* Summary of To-DO List
* Approval Decision

## **Purpose of UAT**

The main **Purpose of UAT** is to validate end to end business flow. It does not focus on cosmetic errors, spelling mistakes or system testing. User Acceptance Testing is carried out in a separate testing environment with production-like data setup. It is kind of black box testing where two or more end-users will be involved.

Once the software has undergone through Unit Testing, Integration Testing and System Testing so, Acceptance Testing may seem redundant, but it is required due to the following reasons.

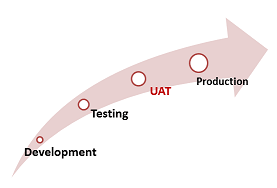
* During the development of a project if there are changes in requirements and it may not be communicated effectively to the development team.
* Developers develop functions by examining the requirement document on their own understanding and may not understand the actual requirements of the client.
* There's maybe some minor errors which can be identified only when the system is used by the end user in the actual scenario so, to find out these minor errors, acceptance testing is essential.



Once the application is bug-free, we handover it to the customer, no customer accept the application blindly before using it. Hence, they do one round of testing for their satisfaction, which is known as user acceptance testing.

## **Who Performs UAT?**

* Client
* End users



The acceptance testing can be performed by different persons in different cases.

**For example**, the blue-dart company gives the requirement to TCS for developing the application, and the TCS will accept the needs and agree to deliver the application in the two releases as we can see in the below image:

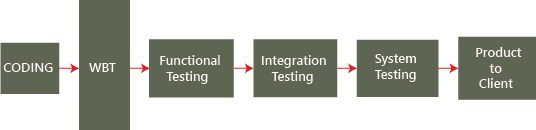
Acceptance Testing

On August 10, the test manager tells the project manager that there is a critical bug in the application, and that will take another four days to fix it.

Acceptance Testing

But the project manager said we have to deliver the software within a given time. It takes another 30 days to fix the defect, or otherwise, we will have to pay the penalty (fine) for each day after the given release date. Is this the real situation? NO, let us see three different cases and understand who perform the acceptance testing.

In this, we will discuss how the acceptance testing is performed, and here the test engineer will do the acceptance testing.



Mostly, the actual flow for testing the application will be seen in the above image, but here it is little difference, as we know where the end-to-end testing or system testing ends and the acceptance testing will proceed. To understand this scenario, follow the below process:

The blue-dart provides the requirements, and TCS develops the application and performs all the testing and handover to the blue-dart company.

Now the question arises the blue-dart will use the application as soon they get it from TCS? NO, the blue dart company has a group of test engineers after they get the software, and this team will start testing the application, and this end-to-end testing is done at the customer environment, which is called the **User Acceptance Testing**.

**Case 1**

Let us see the difference between **TCS test engineers** and **Blue-dart Engineers**:

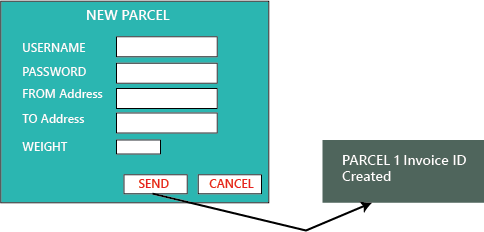
In **TCS**, the tester will perform the **functional testing, integration testing, and system testing**and whereas in **Blue-dart**, the tester will do only the **end-to-end or system testing, which is known as acceptance testing**.

The difference between end-to-end testing at TCS and blue-dart is as follows:

* The blue-dart test engineer is the one who gave the requirements
* The blue-dart engineer understands the product well
* The blue-dart engineer is a domain expert.
* They test the real-time data on the application.

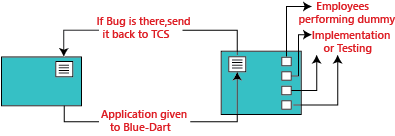
To understand this, we can see the below example, or if we have the application format is like this:

When the application is given to blue-dart test engineers, and they will perform testing and the application should generate a text message "**Parcel 1 invoice Id created**." It was not mentioned in the requirement, or it is there, and TCS does not fix it. Then fine(penalty) counts for TCS from that only, and whereas the test engineers at TCS will not knowing this, due to that, we can see the difference between the testing done at TCS and Blue-dart.



**Case2**

In this case, we will see how the Employee is becoming end-users and performing acceptance testing.



The application is developed and tested at the TCS environment and then sent to blue-dart. And in the Blue-dart, they have fewer test engineers, so they can't do acceptance testing. So for this, out of 300 employees of blue-dart, they will provide the application to the 30 employees and install the application to their systems and ask them to start using the application and find any defect or issues.

Now 30 employees will do the dummy implementation, which means they provide the data into the application and also written that data manually. And here, the employee becomes the end-user and also identify the bugs and issues while using the application.

These issues are verified against the requirements, and now the fine is charged for TCS (sometimes the penalty is charged on an hourly basis). If the identified bug is not as per requirement, then blue-dart can go for the **Request For Enhancement [REF] and Change Request [CR].**

Where **Request for enhancement** means that if the blue-dart feels that a particular module can be improved and developed in a better way, and then they can send the **Customer Requirement Specification [CRS]** as REF and TCS will follow the CRS and also make sure to do the necessary changes.

And the **Change Request** means, if the requirement has not been specified accurately, then blue-dart provides the exact needs and Request for changes.

Therefore, the acceptance testing can also be defined as end-to-end testing, which can be done by the engineers who are working in the client environment. Here, they take real-time scenarios and check whether the application is working fine or not, and also we can make real-time business scenarios because the end-user knows how the business flow works.

**Note:**

If we are getting more builds for acceptance testing, this means that:

* After receiving the application, the customer is getting more and more ideas, so they are asking for more and more changes.
* The quality of the software, which we delivered to customers, is not appropriate, and the development and testing both are not correctly done.
* The requirement which was given in the starting is not clear.

**Case3**

In this case, if the blue-dart customers become the end-users.

Here, the application is developed and tested and implemented at a blue-dart production server, and n-numbers of users start using the application, which is in the first release. While using the application, the blue-dart comes up with more number of features and enhancements, which is sent with the CRS to the TCS after that TCS will do the further changes in modules and sent it back to the blue-dart.

Hence, what is happing here, the application was developed when the requirement is collected by blue-dart from their end-users and customers.

The numbers of releases depend on the following facts:

* Difficulty of modules
* The number of modules.
* How the new module affects the old module.

**Note:**

**Hotfix:** In the production environment, whenever the customer identify the critical bug, we will do the following

* The developers fix the bugs.
* Small teams of test engineers will test the software.
* Re-install the application on the client environment.
* The client starts using the new software.

This entire process is known as a hotfix, and it can be done in a few hours or one day.

**For example:** If the significant module, suppose the Login module itself is not working at the production server, then the client will send it immediately for fixing it, and that has to be done as soon as possible.

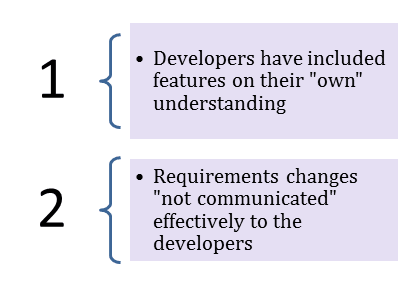
**Short release**

Between two major releases, this is a short release of improvements, and it happens when the client needs some small features to change on an urgent basis.

**For example**, if we have 60 developers, where the ten developers will come out, and out of 40 test engineers, the 3 test engineers will come out, and they develop and test the application. And before adding it to the production server, the customer does one short round of acceptance testing.

## **Need of User Acceptance Testing**

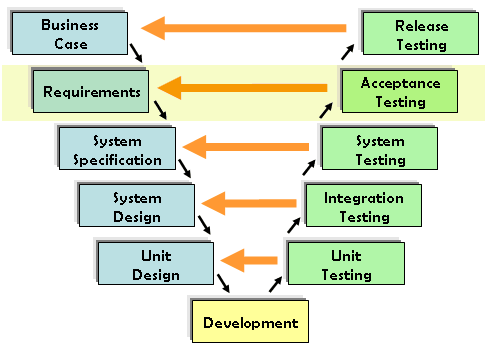
**Need of User Acceptance Testing** arises once software has undergone Unit, Integration and System testing because developers might have built software based on requirements document by their own understanding and further required changes during development may not be effectively communicated to them, so for testing whether the final product is accepted by client/end-user, user acceptance testing is needed.



* Developers code software based on requirements document which is their “own” understanding of the requirements and **may not actually be what the client needs from the software**.
* Requirements changes during the course of the project may not be communicated effectively to the developers.

## **Acceptance Testing and V-Model**

In VModel, User acceptance testing corresponds to the requirement phase of the Software Development life cycle(SDLC).



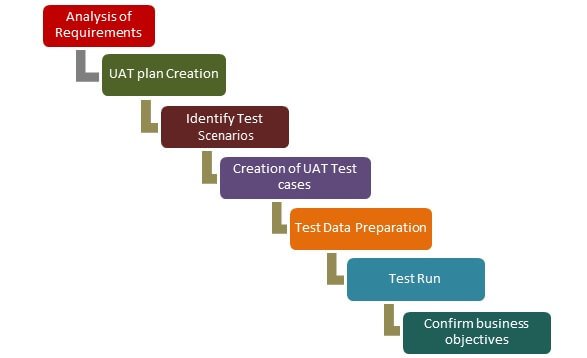
### **Prerequisites of User Acceptance Testing:**

Following are the entry criteria for User Acceptance Testing:

* Business Requirements must be available.
* Application Code should be fully developed
* Unit Testing, Integration Testing & System Testing should be completed
* No Showstoppers, High, Medium defects in System Integration Test Phase –
* Only Cosmetic error is acceptable before UAT
* Regression Testing should be completed with no major defects
* All the reported defects should be fixed and tested before UAT
* Traceability matrix for all testing should be completed
* UAT Environment must be ready
* Sign off mail or communication from System Testing Team that the system is ready for UAT execution

## **How to do UAT Testing**

UAT is done by the intended users of the system or software. This type of Software Testing usually happens at the client location which is known as Beta Testing. Once Entry criteria for UAT are satisfied, following are the tasks need to be performed by the testers:



**Fig.:- UAT Process**

* Analysis of Business Requirements
* Creation of UAT test plan
* Identify Test Scenarios
* Create UAT Test Cases
* Preparation of Test Data(Production like Data)
* Run the Test cases
* Record the Results
* Confirm business objectives

### **Step 1) Analysis of Business Requirements**

One of the most important activities in the UAT is to identify and develop test scenarios. These test scenarios are derived from the following documents:

* Project Charter
* Business Use Cases
* Process Flow Diagrams
* Business Requirements Document(BRD)
* System Requirements Specification(SRS)

### **Step 2) Creation of UAT Plan:**

The UAT test plan outlines the strategy that will be used to verify and ensure an application meets its business requirements. It documents entry and **exit criteria for UAT, Test scenarios and test cases approach and timelines of testing**.

### **Step 3) Identify Test Scenarios and Test Cases:**

Identify the test scenarios with respect to high-level business process and create test cases with clear test steps. Test Cases should sufficiently cover most of the UAT scenarios. Business Use cases are input for creating the test cases.

### **Step 4) Preparation of Test Data:**

It is best advised to use live data for UAT. Data should be scrambled for privacy and [security](https://www.guru99.com/ethical-hacking-tutorials.html) reasons. Tester should be familiar with the database flow.

### **Step 5) Run and record the results:**

Execute test cases and report bugs if any. Re-test bugs once fixed. [Test Management](https://www.guru99.com/test-management.html) tools can be used for execution.

### **Step 6) Confirm Business Objectives met:**

Business Analysts or UAT Testers needs to send a sign off mail after the UAT testing. After sign-off, the product is good to go for production. Deliverables for UAT testing are Test Plan, UAT Scenarios and Test Cases, Test Results and Defect Log

## **Exit criteria for UAT:**

Before moving into production, following needs to be considered:

* No critical defects open
* Business process works satisfactorily
* UAT Sign off meeting with all stakeholders

## **Qualities of UAT Testers:**



UAT Tester should possess good knowledge of the business. He should be independent and think as an **unknown user to the system**. Tester should be Analytical and Lateral thinker and combine all sort of data to make the UAT successful.

Tester or Business Analyst or Subject Matter Experts who understand the business requirements or flows can prepare test and data which are realistic to the business.

## **Best Practices:**

Following points needs to be considered to make UAT Success:

* Prepare UAT plan early in the project life cycle
* Prepare Checklist before the UAT starts
* Conduct Pre-UAT session during System Testing phase itself
* Set the expectation and define the scope of UAT clearly
* Test End to End business flow and avoid system tests
* Test the system or application with real-world scenarios and data
* Think as an Unknown user to the system
* Perform Usability Testing
* Conduct Feedback session and meeting before moving to production

## **Advantages of Acceptance Testing**

* It increases the satisfaction of clients as they test application itself.
* The quality criteria of the software is defined in an early phase so that the tester has already decided the testing points. It gives a clear view to testing strategy.
* The information gathered through acceptance testing used by stakeholders to better understand the requirements of the targeted audience.
* It improves requirement definition as client tests requirement definition according to his needs.

## **Disadvantages of Acceptance Testing**

According to the testing plan, the customer has to write requirements in their own words and by themselves but

1. Customers are not willing to do that; it defeats the whole point of acceptance testing.
2. If test cases are written by someone else, the customer does not understand them, so tester has to perform the inspections by themselves only.

## **Some Example Guidelines of UAT**

* Most of the times in regular software developing scenarios, UAT is carried out in the QA environment. If there is no staging or UAT environment
* UAT is classified into Beta and Alpha testing but it is not so important when software is developed for a service based industry
* UAT makes more sense when the customer is involved to a greater extent

### **Conclusion:**

* In Software Engineering, Full form of UAT is User Acceptance Testing.
* UAT is one of the many flavors of testing that has emerged over last twenty-five years.
* With UAT, the client can be sure “What to expect” from the product rather than assuming.
* The benefit of UAT is that there will be no surprises when the product is released to the market.
* **2.10 Test Types**
* **2.10.1 Functional Testing (Black Box) :-**

**Black Box Testing** is a software testing method in which the functionalities of software applications are tested without having knowledge of internal code structure, implementation details and internal paths. Black Box Testing mainly focuses on input and output of software applications and it is entirely based on software requirements and specifications. It is also known as Behavioral Testing.



The above Black-Box can be any software system you want to test. For Example, an operating system like Windows, a website like Google, a database like Oracle or even your own custom application. Under Black Box Testing, you can test these applications by just focusing on the inputs and outputs without knowing their internal code implementation.

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

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

Black box testing

## **Generic steps of black box testing**

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

## **Test procedure**

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

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

## **How to do Black Box Testing**

Here are the generic steps followed to carry out any type of Black Box Testing.

* Initially, the requirements and specifications of the system are examined.
* Tester chooses valid inputs (positive test scenario) to check whether SUT processes them correctly. Also, some invalid inputs (negative test scenario) are chosen to verify that the SUT is able to detect them.
* Tester determines expected outputs for all those inputs.
* Software tester constructs test cases with the selected inputs.
* The test cases are executed.
* Software tester compares the actual outputs with the expected outputs.
* Defects if any are fixed and re-tested.

Black box testing has its own life cycle called Software Testing Life Cycle ([STLC](https://www.guru99.com/software-testing-life-cycle.html)) and it is relative to every stage of Software Development Life Cycle of Software Engineering.

* **Requirement** – This is the initial stage of SDLC and in this stage, a requirement is gathered. Software testers also take part in this stage.
* Test Planning & Analysis – [Testing Types](https://www.guru99.com/types-of-software-testing.html) applicable to the project are determined. A[Test Plan](https://www.guru99.com/what-everybody-ought-to-know-about-test-planing.html)is created which determines possible project risks and their mitigation.
* **Design** – In this stage Test cases/scripts are created on the basis of software requirement documents
* **Test Execution**– In this stage Test Cases prepared are executed. Bugs if any are fixed and re-tested.
* **2.10.2 Non-functional testing (Testing of software product characteristics) :-**

In **non-functional testing** the quality characteristics of the component or system is tested. Non-functional refers to aspects of the software that may not be related to a specific function or user action such as scalability or security. Eg. How many people can log in at once?

Non-functional testing is a type of software testing to test non-functional parameters such as reliability, load test, performance and accountability of the software. The primary purpose of non-functional testing is to test the reading speed of the software system as per non-functional parameters. The parameters of non-functional testing are never tested before the functional testing.

Non-functional testing is also very important as functional testing because it plays a crucial role in customer satisfaction.

For example, non-functional testing would be to test how many people can work simultaneously on any software.

## **Why Non-Functional Testing**

Functional and Non-functional testing both are mandatory for newly developed software. Functional testing checks the correctness of internal functions while Non-Functional testing checks the ability to work in an external environment.

It sets the way for software installation, setup, and execution. The measurement and metrics used for internal research and development are collected and produced under non-functional testing.

Non-functional testing gives detailed knowledge of product behavior and used technologies. It helps in reducing the risk of production and associated costs of the software.

* **2.10.3 Structural testing (White-Box Testing ) :-**

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

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

Developers do white box testing. In this, the developer will test every line of the code of the program. The developers perform the White-box testing and then send the application or the software to the testing team, where they will perform the [black box testing](https://www.javatpoint.com/black-box-testing)

and verify the application along with the requirements and identify the bugs and sends it to the developer.

The developer fixes the bugs and does one round of white box testing and sends it to the testing team. Here, fixing the bugs implies that the bug is deleted, and the particular feature is working fine on the application.

**White Box Testing** is software testing technique in which internal structure, design and coding of software are tested to verify flow of input-output and to improve design, usability and security. In white box testing, code is visible to testers so it is also called Clear box testing, Open box testing, Transparent box testing, Code-based testing and Glass box testing.

It is one of two parts of the Box Testing approach to software testing. Its counterpart, Black box testing, involves testing from an external or end-user type perspective. On the other hand, White box testing in software engineering is based on the inner workings of an application and revolves around internal testing.

The term “White Box” was used because of the see-through box concept. The clear box or White Box name symbolizes the ability to see through the software’s outer shell (or “box”) into its inner workings. Likewise, the “black box” in “[Black Box Testing](https://www.guru99.com/black-box-testing.html)” symbolizes not being able to see the inner workings of the software so that only the end-user experience can be tested.

## **What do you verify in White Box Testing?**

White box testing involves the testing of the software code for the following:

* Internal security holes
* Broken or poorly structured paths in the coding processes
* The flow of specific inputs through the code
* Expected output
* The functionality of conditional loops
* Testing of each statement, object, and function on an individual basis

****

* **2.10.4 Testing related to changes – Confirmation (Re-testing) and Regression Testing :-**

## **Retesting :-**

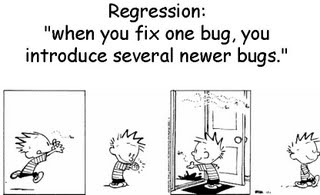
**Retesting** is a process to check specific test cases that are found with bug/s in the final execution. Generally, testers find these bugs while testing the software application and assign it to the developers to fix it. Then the developers fix the bug/s and assign it back to the testers for verification. This continuous process is called Retesting.

### **What is Regression Testing**?

[Regression Testing](https://www.guru99.com/regression-testing.html) is a type of software testing executed to check whether a code change has not unfavorably disturbed current features & functions of an Application

**Regression Testing:**

* Regression testing is a type of software testing that seeks to uncover new software bugs, or regressions, in existing functional and non-functional areas of a system after changes such as enhancements, patches or configuration changes, have been made to them.
* The intent of regression testing is to ensure that changes such as those mentioned above have not introduced new faults.
* One of the main reasons for regression testing is to determine whether a change in one part of the software affects other parts of the software.
* Regression testing include rerunning previously completed tests and checking whether program behavior has changed and whether previously fixed faults have re-emerged.
* Regression testing can be performed to test a system efficiently by systematically selecting the appropriate minimum set of tests needed to adequately cover a particular change.
* Regression testing can be used not only for testing the correctness of a program, but often also for tracking the quality of its output.
* For instance, in the design of a compiler, regression testing could track the code size, and the time it takes to compile and execute the test suite cases.
* Regression testing is the process of testing changes to computer programs to make sure that the older programming still works with the new changes.
* Regression testing is a normal part of the program development process and, in larger companies, is done by code testing specialists.
* Regression means retesting the unchanged parts of the application.
* Test cases are re-executed in order to check whether previous functionality of application is working fine and new changes have not introduced any new bugs.
* This test can be performed on a new build when there is significant change in original functionality or even a single bug fix.
* This is the method of verification.
* Verifying that the bugs are fixed and the newly added features have not created in problem in previous working version of software.



Regression testing is a black box testing techniques. It is used to authenticate a code change in the software does not impact the existing functionality of the product. Regression testing is making sure that the product works fine with new functionality, [bug](https://www.javatpoint.com/bug-in-software-testing) fixes, or any change in the existing feature.

Regression testing is a type of [software testing](https://www.javatpoint.com/software-testing-tutorial). Test cases are re-executed to check the previous functionality of the application is working fine, and the new changes have not produced any bugs.

Regression testing can be performed on a new build when there is a significant change in the original functionality. It ensures that the code still works even when the changes are occurring. Regression means Re-test those parts of the application, which are unchanged.

Regression tests are also known as the Verification Method. Test cases are often automated. [Test cases](https://www.javatpoint.com/test-case) are required to execute many times and running the same test case again and again manually, is time-consuming and tedious too.

### **Example of Regression testing**

Here we are going to take a case to define the regression testing efficiently:

Consider a product Y, in which one of the functionality is to trigger confirmation, acceptance, and dispatched emails. It also needs to be tested to ensure that the change in the code not affected them. Regressing testing does not depend on any programming language like [Java](https://www.javatpoint.com/java-tutorial), [C++](https://www.javatpoint.com/cpp-tutorial), [C#](https://www.javatpoint.com/c-sharp-tutorial), etc. This method is used to test the product for modifications or any updates done. It ensures that any change in a product does not affect the existing module of the product. Verify that the bugs fixed and the newly added features not created any problem in the previous working version of the Software.

## **When can we perform Regression Testing?**

We do regression testing whenever the production code is modified.

We can perform regression testing in the following scenario, these are:

**1. When new functionality added to the application.**

**Example:**

A website has a login functionality which allows users to log in only with Email. Now providing a new feature to do login using Facebook.

**2. When there is a Change Requirement.**

**Example:**

Remember password removed from the login page which is applicable previously.

**3. When the defect fixed**

**Example:**

Assume login button is not working in a login page and a tester reports a bug stating that the login button is broken. Once the bug fixed by developers, tester tests it to make sure Login Button is working as per the expected result. Simultaneously, tester tests other functionality which is related to the login button.

**4. When there is a performance issue fix**

**Example:**

Loading of a home page takes 5 seconds, reducing the load time to 2 seconds.

**5. When there is an environment change**

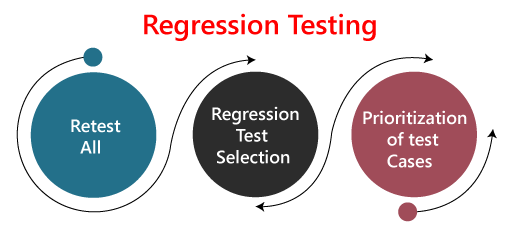
**Example:**

When we update the database from MySQL to Oracle.

## **How to perform Regression Testing?**

The need for regression testing comes when software maintenance includes enhancements, error corrections, optimization, and deletion of existing features. These modifications may affect system functionality. Regression Testing becomes necessary in this case.

Regression testing can be performed using the following techniques:



**1. Re-test All:**

Re-Test is one of the approaches to do regression testing. In this approach, all the test case suits should be re-executed. Here we can define re-test as when a test fails, and we determine the cause of the failure is a software fault. The fault is reported, we can expect a new version of the software in which defect fixed. In this case, we will need to execute the test again to confirm that the fault fixed. This is known as re-testing. Some will refer to this as confirmation testing.

The re-test is very expensive, as it requires enormous time and resources.

**2. Regression test Selection:**

* n this technique, a selected test-case suit will execute rather than an entire test-case suit.
* The selected test case suits divided in two cases
  1. Reusable Test cases.
  2. Obsolete Test cases.
* Reusable test cases can use in succeeding regression cycle.
* Obsolete test cases can't use in succeeding regression cycle.

**3. Prioritization of test cases:**

Prioritize the test case depending on business impact, critical and frequently functionality used. Selection of test cases will reduce the regression test suite.

## **What are the Regression Testing tools?**

Regression Testing is a vital part of the QA process; while performing the regression we may face the below challenges:

* **Time Consuming**

Regression Testing consumes a lot of time to complete. Regression testing involves existing tests again, so testers are not excited to re-run the test.

* **Complex**  
  Regression Testing is complex as well when there is a need to update any product; lists of the test are also increasing.
* **Communicating business rule**

Regression Testing ensures the existing product features are still in working order. Communication about regression testing with a non-technical leader can be a difficult task. The executive wants to see the product move forward and making a considerable time investment in regression testing to ensure existing functionality working can be hard.

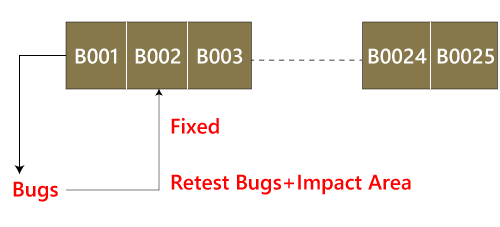
* **Identify Impact Area**
* **Test Cases Increases Release by Release**
* **Less Resources**
* **No Accuracy**
* **Repetitive Task**
* **Monotonous Job**

## **Regression testing process**

The regression testing process can be performed across the **builds** and the **releases**.

### **Regression testing across the builds**

Whenever the bug fixed, we retest the Bug, and if there is any dependent module, we go for a Regression Testing.



**For example**, How we perform the regression testing if we have different builds as **Build 1, Build 2, and Build 3**, which having different scenarios.

**Build1**

* Firstly the client will provide the business needs.
* Then the development team starts developing the features.
* After that, the testing team will start writing the test cases; for example, they write 900 test cases for the release#1 of the product.
* And then, they will start implementing the test cases.
* Once the product is released, the customer performs one round of acceptance testing.
* And in the end, the product is moved to the production server.

**Build2**

* Now, the customer asks for 3-4 extra (new) features to be added and also provides the requirements for the new features.
* The development team starts developing new features.
* After that, the testing team will start writing the test case for the new features, and they write about 150 new test cases. Therefore, the total number of the test case written is 1050 for both the releases.
* Now the testing team starts testing the new features using 150 new test cases.
* Once it is done, they will begin testing the old features with the help of 900 test cases to verify that adding the new feature has damaged the old features or not.
* Here, testing the old features is known as **Regression Testing**.
* Once all the features (New and Old) have been tested, the product is handed over to the customer, and then the customer will do the acceptance testing.
* Once the acceptance testing is done, the product is moved to the production server.

**Build3**

* After the second release, the customer wants to remove one of the features like Sales.
* Then he/she will delete all the test cases which are belonging to the sales module (about 120 test cases).
* And then, test the other feature for verifying that if all the other features are working fine after removing the sales module test cases, and this process is done under the regression testing.

**Note:**

* Testing the stable features to ensure that it is broken because of the changes. Here changes imply that the **modification, addition, bug fixing, or the deletion**.
* Re-execution of the same test cases in the different builds or releases is to ensure that changes (modification, addition, bug fixing, or the deletion) are not introducing bugs in stable features.

### **Regression testing across the release**

The regression testing process starts whenever there is a new Release for same project because the new feature may affect the old elements in the previous releases.

To understand the regression testing process, we will follow the below steps:

**Step1**

There is no regression testing in **Release#1** because there is no modification happen in the Release#1 as the release is new itself.

**Step2**

The concept of Regression testing starts from **Release#2** when the customer gives some **new requirements**.

**Step3**

After getting the new requirements (modifying features) first, they (the developers and test engineers) will understand the needs before going to the **impact analysis**.

**Step4**

After understanding the new requirements, we will perform one round of **impact analysis** to avoid the major risk, but here the question arises who will do the Impact analysis?

**Step5**

The impact analysis is done by the **customer** based on their **business knowledge**, the **developer** based on their **coding knowledge**, and most importantly, it is done by the **test engineer** because they have the **product knowledge**.

**Step6**

Once we are done with the **impact area**, then the developer will prepare the **impact area (document)**, and the **customer** will also prepare the **impact area document** so that we can achieve the **maximum coverage of impact analysis**.

**Step7**

After completing the impact analysis, the developer, the customer, and the test engineer will send the **Reports#** of the impact area documents to the **Test Lead**. And in the meantime, the test engineer and the developer are busy working on the new test case.

**Step8**

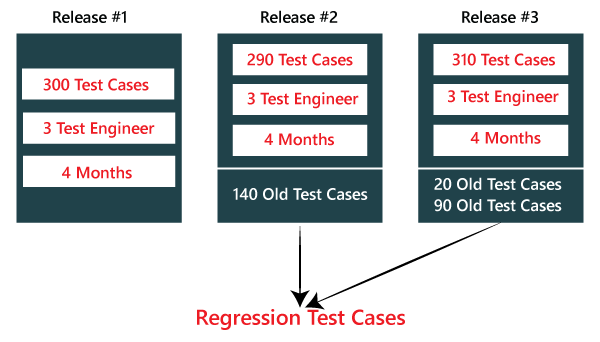
Once the Test lead gets the Reports#, he/she will **consolidate** the reports and stored in the **test case requirement repository** for the release#1.

**tep9**

After that, the Test Lead will take the help of RTM and pick the necessary **regression test case** from the **test case repository**, and those files will be placed in the **Regression Test Suite**.

**Note:**

* The test lead will store the regression test case in the regression test suite for no further confusion.
* **Regression test suite**: Here, we will save all the impact area test documents.
* **Regression Test Cases**: These are the test cases of the old releases text document which need to be re-executed as we can see in the below image:

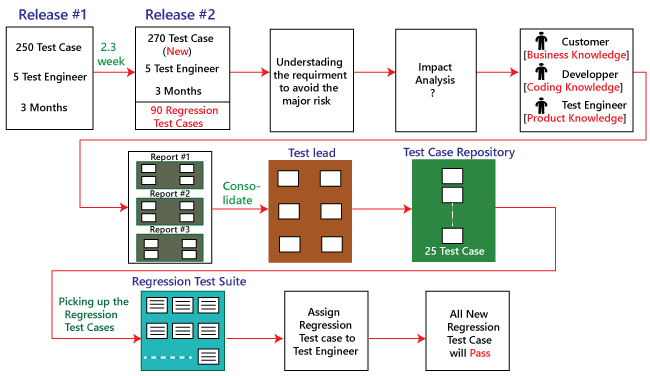


**Step10**

After that, when the test engineer has done working on the new test cases, the test lead will **assign the regression test case** to the test engineer.

**Step11**

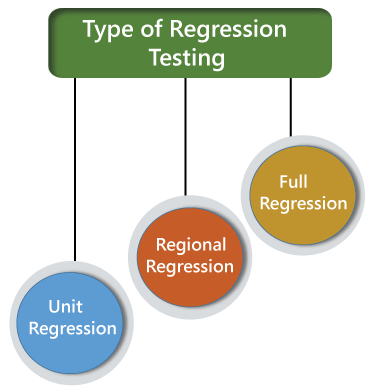
When all the regression test cases and the new features are **stable and pass**, then check the **impact area using the test case** until it is durable for old features plus the new features, and then it will be handed over to the customer.



## **Types of Regression Testing**

The different types of Regression Testing are as follows:

* Unit Regression Testing [URT]
* Regional Regression Testing[RRT]
* Full or Complete Regression Testing [FRT]

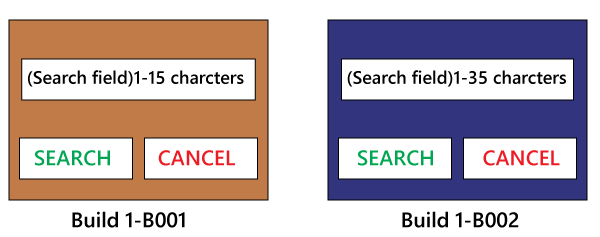


### **Unit Regression Testing [URT]**

In this, we are going to test only the changed unit, not the impact area, because it may affect the components of the same module.

**Example1**

In the below application, and in the first build, the developer develops the **Search** button that accepts **1-15 characters**. Then the test engineer tests the Search button with the help of the **test case design technique**.

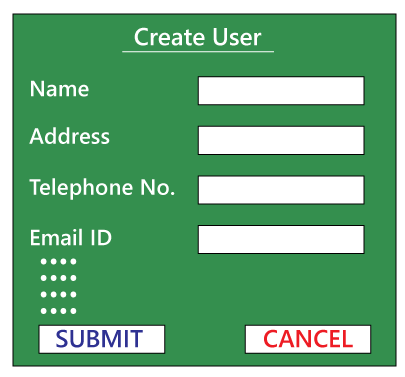


Now, the client does some modification in the requirement and also requests that the **Search button** can accept the **1-35 characters**. The test engineer will test only the Search button to verify that it takes 1-35 characters and does not check any further feature of the first build.

**Example2**

Here, we have **Build B001**, and a defect is identified, and the report is delivered to the developer. The developer will fix the bug and sends along with some new features which are developed in the second **Build B002**. After that, the test engineer will test only after the defect is fixed.

* The test engineer will identify that clicking on the **Submit** button goes to the blank page.
* And it is a defect, and it is sent to the developer for fixing it.
* When the new build comes along with the bug fixes, the test engineer will test only the Submit button.
* And here, we are not going to check other features of the first build and move to test the new features and sent in the second build.
* We are sure that fixing the **Submit** button is not going to affect the other features, so we test only the fixed bug.



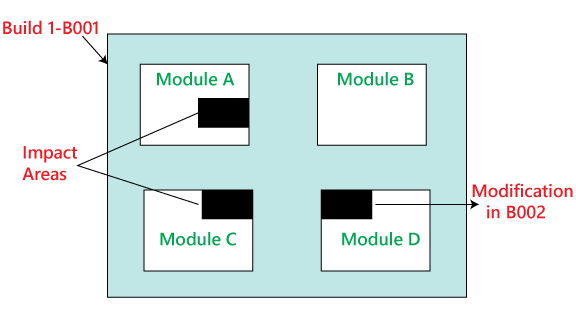
Therefore, we can say that by testing only the changed feature is called the **Unit Regression Testing**.

### **Regional Regression testing [RRT]**

In this, we are going to test the modification along with the impact area or regions, are called the **Regional Regression testing**. Here, we are testing the impact area because if there are dependable modules, it will affect the other modules also.

**For example:**

In the below image as we can see that we have four different modules, such as **Module A, Module B, Module C, and Module D**, which are provided by the developers for the testing during the first build. Now, the test engineer will identify the bugs in **Module D**. The bug report is sent to the developers, and the development team fixes those defects and sends the second build.



In the second build, the previous defects are fixed. Now the test engineer understands that the bug fixing in Module D has impacted some features in **Module A and Module C**. Hence, the test engineer first tests the Module D where the bug has been fixed and then checks the impact areas in **Module A and Module C**. Therefore, this testing is known as **Regional regression testing.**

While performing the regional regression testing, we may face the below problem:

**Problem:**

In the first build, the client sends some modification in requirement and also wants to add new features in the product. The needs are sent to both the teams, i.e., development and testing.

After getting the requirements, the development team starts doing the modification and also develops the new features based on the needs.

Now, the test lead sends mail to the clients and asks them that all are the impact areas that will be affected after the necessary modification have been done. Therefore, the customer will get an idea, which all features are needed to be tested again. And he/she will also send a mail to the development team to know which all areas in the application will be affected as a result of the changes and additions of new features.

And similarly, the customer sends a mail to the testing team for a list of impact areas. Hence, the test lead will collect the impact list from the client, development team, and the testing team as well.

This **Impact list** is sent to all the test engineers who look at the list and check if their features are modified and if yes, then they do **regional regression testing**. The impact areas and modified areas are all tested by the respective engineers. Every test engineer tests only their features that could have been affected as a result of the modification.

The problem with this above approach is that the test lead may not get the whole idea of the impact areas because the development team and the client may not have so much time to revert his/her mails.

**Solution**

To resolve the above problem, we will follow the below process:

When a new build comes along with the latest features and bug fixes, the testing team will arrange the meeting where they will talk about if their features are affecting because of the above modification. Therefore, they will do one round of **Impact Analysis** and generate the **Impact List**. In this particular list, the test engineer tries to enclose the maximum probable impact areas, which also decreases the chance of getting the defects.

When a new build comes, the testing team will follow the below procedure:

* They will do smoke testing to check the basic functionality of an application.
* Then they will test new features.
* After that, they will check the changed features.
* Once they are done with checking the changed features, the test engineer will re-test the bugs.
* And then they will check the impact area by performing the regional regression testing.

### **Disadvantage of using Unit and Regional Regression testing**

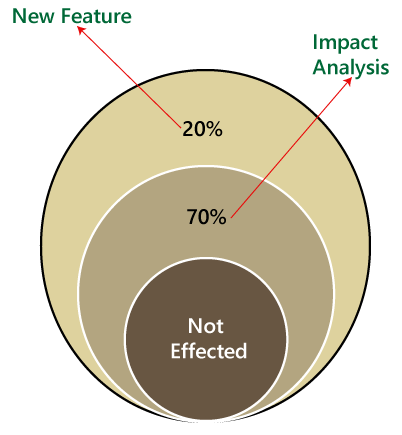
Following are some of the drawbacks of using unit and Regional regression testing:

* We may miss some impact area.
* It is possible that we may identify the wrong impact area.

### **Full Regression testing [FRT]**

During the second and the third release of the product, the client asks for adding 3-4 new features, and also some defects need to be fixed from the previous release. Then the testing team will do the Impact Analysis and identify that the above modification will lead us to test the entire product.

Therefore, we can say that testing the **modified features** and **all the remaining (old) features** is called the **Full Regression testing**.



### **When we perform Full Regression testing?**

We will perform the FRT when we have the following conditions:

* When the modification is happening in the source file of the product. **For example**, JVM is the root file of the JAVA application, and if any change is going to happen in JVM, then the entire JAVA program will be tested.
* When we have to perform n-number of changes.

**Note:**

The regional regression testing is the ideal approach of regression testing, but the issue is, we may miss lots of defects while performing the Regional Regression testing.

And here we are going to solve this issue with the help of the following approach:

* When the application is given for the testing, the test engineer will test the first 10-14 cycle, and will do the **RRT**.
* Then for the 15th cycle, we do FRT. And again, for the next 10-15 cycle, we do **Regional regression testing**, and for the 31th cycle, we do the **full regression testing**, and we will continue like this.
* But for the last ten cycle of the release, we will perform only **complete regression testing**.

Therefore, if we follow the above approach, we can get more defects.

**The drawback of doing regression testing manually repeatedly:**

* Productivity will decrease.
* It is a difficult job to do.
* There is no consistency in test execution.
* And the test execution time is also increased.

Hence, we will go for the automation to get over with these issues; when we have n-number of the regression test cycle, we will go for the **automation regression testing process**.

### **Automated Regression testing process**

Generally we go for the automation whenever there are multiple releases or multiple regression cycle or there is the repetitive task.

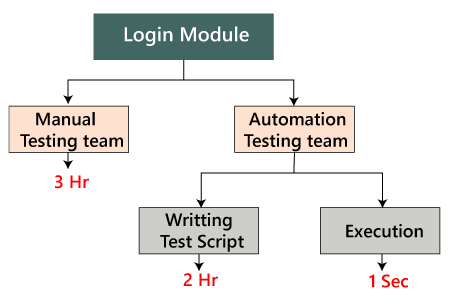
The automation regression testing process can be done in the following steps:

**Note1:**

The process of testing the application by using some tools is known as automation testing.

Suppose if we take one sample example of a **Login module**, then how we can perform the regression testing.

Here, the Login can be done in two ways, which are as follows:



**Manually:** In this, we will perform regression only one and twice.

**Automation:** In this, we will do the automation multiple times as we have to write the test scripts and do the execution.

|  |  |
| --- | --- |
| **Issues** | **Handle by** |
| New features | Manual test engineer |
| Regressing testing features | Automation test engineer |
| Remaining ( 110 feature + Release#1) | Manual test engineer |

**Step1**

When the new release starts, we don't go for the automation because there is no concept of regression testing and regression test case as we understood this in the above process.

**Step2**

When the new release and the enhancement starts, we have two teams, i.e., manual team and the automation team.

**Step3**

The manual team will go through the requirements and also identify the impact area and hand over the **requirement test suite** to the automation team.

**Step4**

Now, the manual team starts working on the new features, and the automation team will start developing the test script and also start automating the [test case](https://www.javatpoint.com/test-case), which means that the regression test cases will be converted into the test script.

**Step5**

Before they (automation team) start automating the test case, they will also analyze which all cases can be automated or not.

**Step6**

Based on the analysis, they will start the automation i.e., converting every regression test cases into the test script.

**Step7**

During this process, they will take help of the **Regression cases** because they don't have product knowledge as well as the **tool** and the **application**.

**Step8**

Once the test script is ready, they will start the execution of these scripts on the new application [old feature]. Since, the test script is written with the help of the regression feature or the old feature.

**Step9**

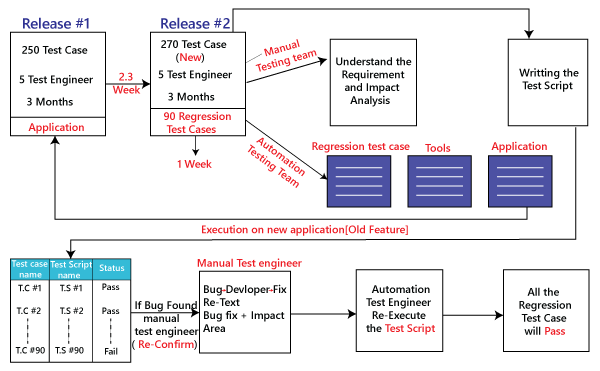
Once the execution is completed, we get a different status like **Pass/fail**.

**Step10**

If the status is failed, which means it needs to be re-confirmed manually, and if the Bug exists, then it will report to the concerned developer. When the developer fixes that bug, the Bug needs to be re-tested along with the Impact area by the manual test engineer, and also the script needs to be re-executed by the automation test engineer.

**Step11**

This process goes on until all the new features, and the regression feature will be passed.



### **Benefits of doing regression testing by the automation testing:**

* **Accuracy** always exists because the task is done by the tools and tools never get bored or tired.
* The test script can be re-used across multiple releases.
* **Batch execution** is possible using the automation i.e.; all the written test scripts can be executed parallel or simultaneously.
* Even though the number of regression test case increase release per release, and we don't have to increase the automation resource since some regression case are already automated from the previous release.
* It is a **time-saving process** because the execution is always faster than the manual method.

## **How to select test cases for regression testing?**

It was found from industry inspection. The several defects reported by the customer were due to last-minute bug fixes. These creating side effects and hence selecting the Test Case for regression testing is an art, not an easy task.

Regression test can be done by:

* A test case which has frequent defects
* Functionalities which are more visible to users.
* Test cases verify the core features of the product.
* All integration test cases
* All complex test cases
* Boundary value test cases
* A sample of successful test cases
* Failure of test cases

## **Regression testing tools**

If Software undergoes frequent changes, regression testing costs also increase. In those cases, manual execution of test cases increases test execution time as well as costs. In that case, automation testing is the best choice. The duration of automation depends on the number of test cases that remain reusable for successive regression cycles.

**Following are the essential tools used for regression testing:**

**Selenium**

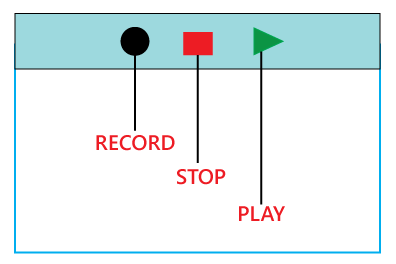
Selenium is an open-source tool. This tool used for automated testing of a web application. For browser-based regression testing, selenium used. Selenium used for UI level regression test for web-based application.

**Ranorex Studio**

All in one regression test automation for desktop, web, and mobile apps with built-in Selenium Web Driver. Ranorex Studio includes full IDE plus tools for codeless automation.

**Quick Test Professional (QTP)**

QTP is an automated testing tool used for Regression and Functional Testing. It is a Data-Driven, keyword-based tool. It used VBScript language for automation. If we open the QTP tool, we see the three buttons which are **Record, Play and Stop**. These buttons help to record every click and action performed on the computer system. It records the actions and play it back.



**Rational Functional Tester (RTF)**

Rational functional tester is a Java tool used to automate the test cases of software applications. RTF used for automating regression test cases, and it also integrates with the rational functional tester.

## **What are the Regression Testing and Configuration Management?**

Configuration Management in the regression testing becomes imperative in Agile Environments, where a code is continuously modified. To ensure a valid regression test, we must follow the steps:

* Changes are not allowed in the code during the regression testing phase.
* A regression test case must be unaffected developer changes.
* The database used for regression testing must be isolated; changes are not allowed in the database.

## **What are the differences between Retesting and Regression Testing?**

**Re-testing Testing** means testing the functionality or bug again to ensure the code fixed. If not set, defects need not be re-opened. If fixed, the defect closed.

Re-testing is a type of testing which performed to check the test-cases that were unsuccessful in the final execution are successfully pass after the defects repaired.

**Regression Testing** means testing the software application when it undergoes a code change to ensure that new code has not affected other parts of the Software.

Regression testing is a type of testing executed to check whether a code has not changed the existing functionality of the application.

**Differences between the Re-testing and Regression Testing are as follows:**

|  |  |
| --- | --- |
| **Re-testing** | **Regression Testing** |
| Re-testing is performed to ensure that the test cases that are failed in the final execution are passing after the defects fixed. | Regression Testing is done to confirm whether the code change has not affected the existing features. |
| Re-Testing works on defect fixes. | The purpose of regression testing is to ensure that the code changes adversely not affect the existing functionality. |
| Defect verification is the part of the Retesting. | Regression testing does not include defect verification |
| The priority of Retesting is higher than Regression Testing, so it is done before the Regression Testing. | Based on the project type and availability of resources, regression testing can be parallel to Retesting. |
| Re-Test is a planned Testing. | Regression testing is a generic Testing. |
| We cannot automate the test-cases for Retesting. | We can do automation for regression testing; manual testing could be expensive and time-consuming. |
| Re-testing is for failed test-cases. | Regression testing is for passed Test-cases. |
| Re-testing make sure that the original fault is corrected. | Regression testing checks for unexpected side effect. |
| Retesting executes defects with the same data and the same environment with different input with a new build. | Regression testing is when there is a modification or changes become mandatory in an existing project. |
| Re-testing cannot do before start testing. | Regression testing can obtain test cases from the functional specification, user tutorials and manuals, and defects reports in regards to the corrected problem. |

## **What are the advantages of Regression Testing?**

Advantages of Regression Testing are:

* Regression Testing increases the product's quality.
* It ensures that any bug fix or changes do not impact the existing functionality of the product.
* Automation tools can be used for regression testing.
* It makes sure the issues fixed do not occur again.

## **What are the disadvantages of Regression Testing?**

There are several advantages of Regression Testing though there are disadvantages as well.

* Regression Testing should be done for small changes in the code because even a slight change in the code can create issues in the existing functionality.
* If in case automation is not used in the project for testing, it will time consuming and tedious task to execute the test again and again.

## **Conclusion**

Regression Testing is one of the essential aspects as it helps to deliver a quality product which saves organizations time and money. It helps to provide a quality product by making sure that any change in the code does not affect the existing functionality.

For automating the regression test cases, there are several automation tools available. A tool should have the ability to update the **test suite** as the regression test suit needs to be updated frequently.

* **2.11 Non-Functional Testing Types :-**
* **2.11.1 Performance (Load & Stress) Testing :-**

**Performance Testing** is a software testing process used for testing the speed, response time, stability, reliability, scalability and resource usage of a software application under particular workload. The main purpose of performance testing is to identify and eliminate the performance bottlenecks in the software application. It is a subset of performance engineering and also known as “Perf Testing”.

The focus of Performance Testing is checking a software program’s

* Speed – Determines whether the application responds quickly
* Scalability – Determines maximum user load the software application can handle.
* Stability – Determines if the application is stable under varying loads

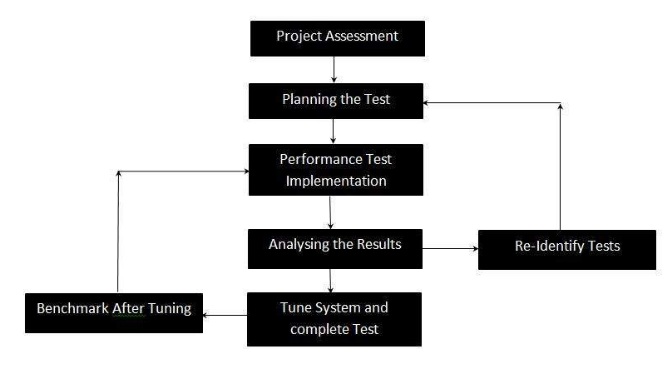
Performance testing, a non-functional testing technique performed to determine the system parameters in terms of responsiveness and stability under various workload. Performance testing measures the quality attributes of the system, such as scalability, reliability and resource usage.

## “Checking the behavior of an application by applying some load is known as performance testing.”

## **Performance Testing Techniques:**

* **Load testing -**It is the simplest form of testing conducted to understand the behavior of the system under a specific load. Load testing will result in measuring important business critical transactions and load on the database, application server, etc., are also monitored.
* **Stress testing -**It is performed to find the upper limit capacity of the system and also to determine how the system performs if the current load goes well above the expected maximum.
* **Soak testing -**Soak Testing also known as endurance testing, is performed to determine the system parameters under continuous expected load. During soak tests the parameters such as memory utilization is monitored to detect memory leaks or other performance issues. The main aim is to discover the system's performance under sustained use.
* **Spike testing -**Spike testing is performed by increasing the number of users suddenly by a very large amount and measuring the performance of the system. The main aim is to determine whether the system will be able to sustain the workload.

## **Performance Testing Process:**



1. **Identify your testing environment –** Know your physical test environment, production environment and what testing tools are available. Understand details of the hardware, software and network configurations used during testing before you begin the testing process. It will help testers create more efficient tests.  It will also help identify possible challenges that testers may encounter during the performance testing procedures.
2. **Identify the performance acceptance criteria –** This includes goals and constraints for throughput, response times and resource allocation.  It is also necessary to identify project success criteria outside of these goals and constraints. Testers should be empowered to set performance criteria and goals because often the project specifications will not include a wide enough variety of performance benchmarks. Sometimes there may be none at all. When possible finding a similar application to compare to is a good way to set performance goals.
3. **Plan & design performance tests –** Determine how usage is likely to vary amongst end users and identify key scenarios to test for all possible use cases. It is necessary to simulate a variety of end users, plan performance test data and outline what metrics will be gathered.
4. **Configuring the test environment –**Prepare the testing environment before execution. Also, arrange tools and other resources.
5. **Implement test design –** Create the performance tests according to your test design.
6. **Run the tests –** Execute and monitor the tests.
7. **Analyze, tune and retest** – Consolidate, analyze and share test results. Then fine tune and test again to see if there is an improvement or decrease in performance. Since improvements generally grow smaller with each retest, stop when bottlenecking is caused by the CPU. Then you may have the consider option of increasing CPU power.

The performance testing cannot be done manually since:

* We need a lot of resources, and it became a costlier approach.
* And the accuracy cannot maintain when we track response time manually.

The Performance testing process will be completed in the following steps:

* Identify performance scenarios
* Plan and design performance test script
* Configure the test environment & distribute the load
* Execute test scripts
* Result
* Analysis result
* Identify the Bottleneck
* Re-run test



If we perform a **positive flow** of the performance testing process, it could follow the below process:

### **Identify performance scenarios:-**

Firstly, we will identify the performance scenarios based on these below factors:

**Most commonly scenarios:** It means that we can find the performance scenarios based on the scenarios, which commonly used like in the **Gmail application;** we will perform **login, inbox, send items, and compose a mail and logout**.

**Most critical scenarios:** Critical scenarios mean regularly used and important for the business-like in Gmail application **login, compose, inbox, and logout**.

**Huge data transaction:** If we have huge data means that n-number of the users using the application at the same time.

Once we identify the performance scenarios, we will move to the next step.

### **Plan and design performance test script:-**

In this step, we will install the tools in the Test Engineer Machine and access the test server and then we write some script according to the test scenarios and run the tool.

Once we are done with writing the script, we will go to the next step.

### **Configure the test environment & distribute the load:-**

After writing the test scripts, we will arrange the testing environment before the execution. And also, manage the tools, other resources and distribute the load according to the "Usage Pattern" or mention the duration and stability.

### **Execute test scripts:-**

Once we are done with distributing the load, we will execute, validate, and monitor the test scripts.

### **Result:-**

After executing the test scripts, we will get the test result. And check that the result meeting the goal in the given response time or not, and the response time could be maximum, average, and minimum.

If the response is not meeting the required time response, then we will go for the **negative flow** where will perform the below steps:

### **Analysis result**

First, we will analyze the test result whether it meets with the response time or not.

### **Identify the Bottleneck**

After that, we will identify the **bottleneck (bug or performance issue**). And the bottleneck could occur because of these aspects like the **problem in code, hardware issue (hard disk, RAM Processor), network issues,** and the **software issue (operating system)**. And after finding the bottleneck, we will perform **tuning (fix or adjustment)** to resolve this bottleneck.

### **Re-run test**

Once we fix the bottlenecks, re-run the test scripts and checks the result whether it meets the required goal or not.

## **The problem occurs in performance testing**

While performing performance testing on the application, some problems may occur, and these problems are also called the **performance issue**.

The performance issues are as follows:

* **Response time issue**
* **Scalability issue**
* **Bottleneck**
* **Speed issue**

### **Response time issue**

The response time means how quickly the server respond to the client's request. If the user's request does not complete in the given response time, it might have possible that the user may be lost his/her interest in the particular software or application. That's why the application or software should have a perfect response time for responding user's request quickly.

### **Scalability issue**

The scalability issues occur when the application cannot take the n-numbers of users and expected user requests at the same time. That's why we will do **upward scalability testing** (check the maximum capacity of the application) and **downward scalability testing** (when expected time is not matched with the actual time).

### **Bottleneck**

The Bottleneck is the informal name of bug, which occurs when the application is limited by a single component and creates a bad impact on the system performance.

The main causes of bottlenecking are **software issues (issue related to the operating system), hardware issues (issues related to the hard disk, RAM and the processor),** and **coding issue,** etc.

Following are the most common performance bottlenecks:

* Memory utilization
* Disk usage
* CPU utilization
* Operating System limitations
* Network utilization

### **Speed issues**

When we perform performance testing on the application, the application should be faster in speed to get the user's interest and attention because if the application's speed is slow, it may lose the user interest in the application.

## 

### **Load testing**

The load testing is used to check the performance of an application by applying some load which is either less than or equal to the desired load is known as load testing.

**Load testing –** checks the application’s ability to perform under anticipated user loads. The objective is to identify performance bottlenecks before the software application goes live.

**For example:** In the below image, **1000 users** are the **desired load**, which is given by the customer, and **3/second** is the **goal** which we want to achieve while performing a load testing.

### **Stress Testing**

The stress testing is testing, which checks the behavior of an application by applying load greater than the desired load.

**Stress testing –** involves testing an application under extreme workloads to see how it handles high traffic or data processing. The objective is to identify the breaking point of an application.

**For example:** If we took the above example and increased the desired load 1000 to 1100 users, and the goal is 4/second. While performing the stress testing in this scenario, it will pass because the load is greater (100 up) than the actual desired load.

### **Scalability Testing**

Checking the performance of an application by increasing or decreasing the load in particular scales (no of a user) is known as **scalability testing**. Upward scalability and downward scalability testing are called scalability testing.

The objective of scalability testing is to determine the software application’s effectiveness in “scaling up” to support an increase in user load. It helps plan capacity addition to your software system.

Scalability testing is divided into two parts which are as follows:

* **Upward scalability testing**
* **Downward scalability testing**

**Upward scalability testing**

It is testing where we **increase the number of users on a particular scale** until we get a crash point. We will use upward scalability testing to find the maximum capacity of an application.

**Downward scalability testing**

The downward scalability testing is used when the load testing is not passed, then start **decreasing the no. of users in a particular interval** until the goal is achieved. So that it is easy to identify the bottleneck (bug).

### **Stability Testing**

Checking the performance of an application by **applying the load for a particular duration of time** is known as **Stability Testing**.

**Endurance testing –** is done to make sure the software can handle the expected load over a long period of time.

**Spike testing –** tests the software’s reaction to sudden large spikes in the load generated by users.

**Volume testing** – Under Volume Testing large no. of. Data is populated in a database and the overall software system’s behaviour is monitored. The objective is to check software application’s performance under varying database volumes.

**Volume testing** is testing, which helps us to check the behavior of an application by inserting a massive volume of the load in terms of data is known as volume testing, and here, we will concentrate on the number of data rates than the number of users.

**Soak Testing** In this type of testing, we will check the behavior of an application on the environment, which is unsupportive for a long duration of time is known as soak testing.

## **Why do Performance Testing?**

Features and Functionality supported by a software system is not the only concern. A software application’s performance like its response time, reliability, resource usage and scalability do matter. The goal of Performance Testing is not to find bugs but to eliminate performance bottlenecks.

Performance Testing is done to provide stakeholders with information about their application regarding speed, stability, and scalability. More importantly, Performance Testing uncovers what needs to be improved before the product goes to market. Without Performance Testing, software is likely to suffer from issues such as: running slow while several users use it simultaneously, inconsistencies across different operating systems and poor usability.

Performance testing will determine whether their software meets speed, scalability and stability requirements under expected workloads. Applications sent to market with poor performance metrics due to nonexistent or poor performance testing are likely to gain a bad reputation and fail to meet expected sales goals.

Also, mission-critical applications like space launch programs or life-saving medical equipment should be performance tested to ensure that they run for a long period without deviations.

According to Dunn & Bradstreet, 59% of Fortune 500 companies experience an estimated 1.6 hours of downtime every week. Considering the average Fortune 500 company with a minimum of 10,000 employees is paying $56 per hour, the labor part of downtime costs for such an organization would be $896,000 weekly, translating into more than $46 million per year.

Only a 5-minute downtime of Google.com (19-Aug-13) is estimated to cost the search giant as much as $545,000.

It’s estimated that companies lost sales worth $1100 per second due to a recent Amazon Web Service Outage.

Hence, performance testing is important.

## **Common Performance Problems**

* **Long Load time –** Load time is normally the initial time it takes an application to start. This should generally be kept to a minimum. While some applications are impossible to make load in under a minute, Load time should be kept under a few seconds if possible.
* **Poor response time –** Response time is the time it takes from when a user inputs data into the application until the application outputs a response to that input. Generally, this should be very quick. Again if a user has to wait too long, they lose interest.
* **Poor scalability –** A software product suffers from poor scalability when it cannot handle the expected number of users or when it does not accommodate a wide enough range of users. [Load Testing](https://www.guru99.com/load-testing-tutorial.html) should be done to be certain the application can handle the anticipated number of users.
* **Bottlenecking –** Bottlenecks are obstructions in a system which degrade overall system performance. Bottlenecking is when either coding errors or hardware issues cause a decrease of throughput under certain loads. Bottlenecking is often caused by one faulty section of code. The key to fixing a bottlenecking issue is to find the section of code that is causing the slowdown and try to fix it there. Bottlenecking is generally fixed by either fixing poor running processes or adding additional Hardware. Some **common performance bottlenecks** are
  + CPU utilization
  + Memory utilization
  + Network utilization
  + Operating System limitations
  + Disk usage

## **Performance testing example-**

Let us take one example where we will **test the behavior of an application where the desired load is either less than 1000 or equal to 1000 users**.

In the below image, we can see that the **100 up** users are increased continuously to check the **maximum load**, which is also called **upward scalability testing**.

* **Scenario1:** When we have the 1000 users as desired load, and the 2.7/sec is goal time, these scenarios will pass while performing the load test because in load testing, we will concentrate on the no. of users, and as per the requirement it is equal to 1000 user.
* **Scenario2:** In the next scenario, we will increase the desired load by 100 users, and goal time will go up to 3.5\sec. This scenario will pass if we perform stress testing because here, the actual load is greater than (1100) the desired load (1000).
* **Scenario3:** In this, if we increase the desired load three times as  
  **1200 → 3.5\sec:** [it is not less than or equal to the desired load that's why it will **Fail**]  
  **1300 → 4\sec:** [it is not less than or equal to the desired load. i.e., **Fail**]  
  **1400 → Crashed**

## 

## Some More Example Performance Test Cases

* Verify response time is not more than 4 secs when 1000 users access the website simultaneously.
* Verify response time of the Application Under Load is within an acceptable range when the network connectivity is slow
* Check the maximum number of users that the application can handle before it crashes.
* Check database execution time when 500 records are read/written simultaneously.
* Check CPU and memory usage of the application and the database server under peak load conditions
* Verify response time of the application under low, normal, moderate and heavy load conditions.

## Performance Testing Metrics: Parameters Monitored

* **Processor Usage –** an amount of time processor spends executing non-idle threads.
* **Memory use –** amount of physical memory available to processes on a computer.
* **Disk time –**amount of time disk is busy executing a read or write request.
* **Bandwidth –** shows the bits per second used by a network interface.
* **Private bytes –** number of bytes a process has allocated that can’t be shared amongst other processes. These are used to measure memory leaks and usage.
* **Committed memory –** amount of virtual memory used.
* **Memory pages/second –** number of pages written to or read from the disk in order to resolve hard page faults. Hard page faults are when code not from the current working set is called up from elsewhere and retrieved from a disk.
* **Page faults/second –** the overall rate in which fault pages are processed by the processor. This again occurs when a process requires code from outside its working set.
* **CPU interrupts per second –** is the avg. number of hardware interrupts a processor is receiving and processing each second.
* **Disk queue length –** is the avg. no. of read and write requests queued for the selected disk during a sample interval.
* **Network output queue length –** length of the output packet queue in packets. Anything more than two means a delay and bottlenecking needs to be stopped.
* **Network bytes total per second –** rate which bytes are sent and received on the interface including framing characters.
* **Response time –** time from when a user enters a request until the first character of the response is received.
* **Throughput –** rate a computer or network receives requests per second.
* **Amount of connection pooling –** the number of user requests that are met by pooled connections. The more requests met by connections in the pool, the better the performance will be.
* **Maximum active sessions –** the maximum number of sessions that can be active at once.
* **Hit ratios –** This has to do with the number of[SQL](https://www.guru99.com/sql.html)statements that are handled by cached data instead of expensive I/O operations. This is a good place to start for solving bottlenecking issues.
* **Hits per second –** the no. of hits on a web server during each second of a load test.
* **Rollback segment –** the amount of data that can rollback at any point in time.
* **Database locks –** locking of tables and databases needs to be monitored and carefully tuned.
* **Top waits –** are monitored to determine what wait times can be cut down when dealing with the how fast data is retrieved from memory
* **Thread counts –** An applications health can be measured by the no. of threads that are running and currently active.
* **Garbage collection –** It has to do with returning unused memory back to the system. Garbage collection needs to be monitored for efficiency.

## **2.11.2 Usability Testing :-**

**Usability Testing** also known as User Experience(UX) Testing, is a testing method for measuring how easy and user-friendly a software application is. A small set of target end-users, use software application to expose usability defects. Usability testing mainly focuses on user’s ease of using application, flexibility of application to handle controls and ability of application to meet its objectives.

There are many software applications/websites, which miserably fail, once launched, due to following reasons –

* Where do I click next?
* Which page needs to be navigated?
* Which Icon or Jargon represents what?
* Error messages are not consistent or effectively displayed
* Session time not sufficient.

Usability Testing is a significant [type of **software testing**](https://www.javatpoint.com/types-of-software-testing) technique, which is comes under the [**non-functional testing**](https://www.javatpoint.com/non-functional-testing)

It is primarily used in user-centered interaction design on order to check the usability or ease of using a software product. The implementation of usability testing requires an understanding of the application, as it is extensive testing.

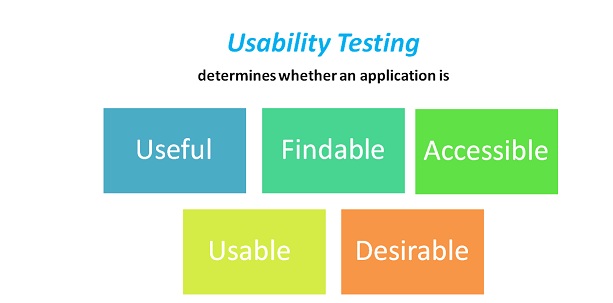
Generally, usability testing is performed from an end-user viewpoint to verify if the system is efficiently working or not.

*“Checking the user-friendliness, efficiency, and accuracy of the application is known as Usability Testing. “*

The primary purpose of executing the usability testing is to check that the application should be easy to use for the end-user who is meant to use it, whereas sustaining the client's specified functional and business requirements.

When we use usability testing, it makes sure that the developed software is straightforward while using the system without facing any problem and makes end-user life easier.

In other words, we can say that Usability testing is one of the distinct testing techniques that identify the defect in the end-user communication of software product. And that's why it is also known as **User Experience** (UX) Testing.

****

The goal of this testing is to satisfy users and it mainly concentrates on the following parameters of a system:

**The effectiveness of the system**

* Is the system is easy to learn?
* Is the system useful and adds value to the target audience?
* Are Content, Color, Icons, Images used are aesthetically pleasing?

**Efficiency**

* Little navigation should be required to reach the desired screen or webpage, and scrollbars should be used infrequently.
* Uniformity in the **format** of screen/pages in your application/website.
* Option to search within your software application or website.

**Accuracy**

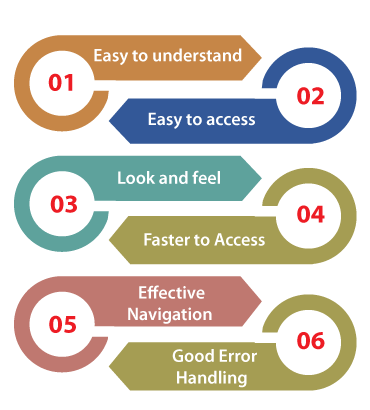
* No out dated or incorrect data like contact information/address should be present.
* No broken links should be present.

**User Friendliness**

* Controls used should be self-explanatory and must not require training to operate
* Help should be provided for the users to understand the application/website
* Alignment with the above goals helps in effective usability testing

In Usability Testing, the user-friendliness can be described with the help of the following characteristics:

* Easy to understand
* Easy to access
* Look and feel
* Faster to Access
* Effective Navigation
* Good Error Handling



Let us see them one by one for our better understanding:

**Easy to understand**

* All the features of software or applications must be visible to the end-users.
* Easy to Access
* A user-friendly application should be accessible by everyone.
* Easy to Access
* The look and feel of the application should be excellent and attractive to get the user's interest.
* The GUI of the software should be good because if the GUI is not well, the user may be lost his/her interest while using the application or the software.
* The quality of the product is up to the mark as given by the client.

**Faster to Access**

* The software should be faster while accessing, which means that the application's response time is quick.
* If the response time is slow, it might happen that the user got irritated. We have to ensure that our application will be loaded within 3 to 6 seconds of the response time.

**Effective Navigation**

* Effective navigation is the most significant aspect of the software. Some of the following aspects for effective navigation:
* Good Internal Linking
* Informative header and footer
* Good search feature

**Good Error Handling**

* Handling error at a coding level makes sure that the software or the application is bug-free and robust.
* By showing the correct error message will help to enhance the user experience and usability of the application.

**Why do we need to perform Usability Testing?**

We need usability testing because usability testing is to build a system with great user experience. Usability is not only used for [software development](https://www.javatpoint.com/software-development-life-cycle)

or website development, but it is also used for product designing.

And Customers must be comfortable with your application with the following parameters.

* The flow of an Application should be good
* Navigation steps should be clear
* Content should be simple
* The layout should be clear
* Response time

And we can also test the different features in usability testing given as follows:

* How easy it is using the application
* How easy to learn application

## **Features of Usability Testing**

The implementation of usability testing helps us to increase the end-user experience of the particular application and software. With the help of usability testing, the software development team can quickly detect several usability errors in the system and fix them quickly.

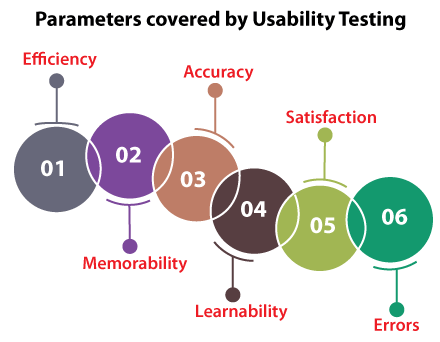
Some other vital features of usability testing are as follows:

* It is an essential type of **non-functional testing** technique, which comes under the **black-box testing** technique in **software testing**.
* Usability testing is performed throughout the **system** and **acceptance testing** levels.
* Generally, usability testing is implemented in the early stage of **the Software Development Life Cycle** (SDLC).
* The execution of usability testing offers more visibility on the prospects of the end-users.
* The usability testing makes sure that the software product meets its planned purpose.
* It also helps us to find many usability errors in the specified software product.
* Usability testing mainly tests the user-friendliness, usefulness, traceability, usability, and desirability of the final product.
* It offers direct input on how real-users use the software/application.
* The usability testing includes the systematic executions of the product's usability under a measured environment.

## **Parameters covered by Usability Testing**

In order to test the **quality, usability, user-friendline**ss, and other significant factors of the software, usability testing plays an important role. And it also helps us in order to supports the organizations for delivering a more extensive services to their target audience.

However, the impact of usability testing is inadequate to these aspects, and also covered the following various constraints or parameters that help us enhance the software's productivity.



1. **Efficiency**
2. **Memorability**
3. **Accuracy**
4. **Learnability**
5. **Satisfaction**
6. **Errors**

Let's see them separately in order to enhance our knowledge of usability testing:

### **Efficiency**

The first constraint covered by the execution of usability testing is **Efficiency.** Here, the efficiency parameter explains the end-user who is an expert and can take the minimum amount of time to execute his/her fundamental or, we can say, undeveloped task.

### **Memorability**

The second constrain that is covered by the implementation of usability testing is **Memorability.** The Memorability of an application can be beneficial or not beneficial. But, the question arises, how can we have decided the Memorability of an application is good or bad?

The below points will give the perfect answer to the above arise the question:

* When we are not asking for an application for some time and returning to the application or trying to do the simple task without any help, we can say that the **Memorability of an application is beneficial.**
* Or, if we cannot execute a simple task without any help after a duration, we can say that the **Memorability of an application is not beneficial.**

### **Accuracy**

The next parameter covered by performing the Usability testing is **Accuracy**. The usability testing ensures that no inappropriate/irrelevant data or information exists in the product. And also, able to discover the broken links in the particular product that helps us develop the accuracy of the final product.

### **Learnability**

Another constraint that is encompassed by usability testing is **Learnability.** In this constraint, the end-user takes a minimum amount of time to learn the fundamental task.

### **Satisfaction**

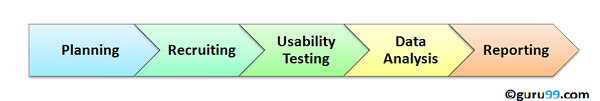
The execution of usability testing ensures the **customer's satisfaction** as we know that the satisfied customer can easily or freely use the application.

### **Errors**

The last and most important parameter covered by the usability testing is **Errors detection**. At this point, we try to help the end-users fix those errors they made earlier and accomplish their tasks all over again.

## How to do Usability Testing:

Usability testing process consists of the following phases



**Planning**:- During this phase the goals of usability test are determined. Having volunteers sit in front of your application and recording their actions is not a goal. You need to determine critical functionalities and objectives of the system. You need to assign tasks to your testers, which exercise these critical functionalities. During this phase, the usability testing method, number & demographics of usability testers, test report formats are also determined

**Recruiting**: During this phase, you recruit the desired number of testers as per your usability test plan. Finding testers who match your demographic (age, sex etc.) and professional ( education, job etc.) profile can take time.

**Usability** **Testing**: During this phase, usability tests are actually executed.

**Data Analysis**: Data from usability tests is thoroughly analyzed to derive meaningful inferences and give actionable recommendations to improve the overall usability of your product.

**Reporting**: Findings of the usability test is shared with all concerned stakeholders which can include designer, developer, client, and CEO

## Methods of Usability Testing

There are two methods available to do usability testing –

1. Laboratory Usability Testing
2. Remote Usability Testing

**Laboratory Usability Testing:**. This testing is conducted in a separate lab room in presence of the observers. The testers are assigned tasks to execute. The role of the observer is to monitor the behavior of the testers and report the outcome of testing. The observer remains silent during the course of testing. In this testing, both observers and testers are present in a same physical location.

**Remote Usability Testing**: Under this testing observers and testers are remotely located. Testers access the System Under Test, remotely and perform assigned tasks. Tester’s voice , screen activity , testers facial expressions are recorded by an automated software. Observers analyze this data and report findings of the test.

## Usability Testing Advantages

As with anything in life, usability testing has its merits and de-merits. Let’s look at them

* It helps uncover usability issues before the product is marketed.
* It helps improve end-user satisfaction
* It makes your system highly effective and efficient
* It helps gather true feedback from your target audience who actually use your system during a usability test. You do not need to rely on “opinions” from random people.

## Usability Testing Disadvantages

* Cost is a major consideration in usability testing. It takes lots of resources to set up a Usability Test Lab. Recruiting and management of usability testers can also be expensive

## **2.11.3 Maintainability Testing :-**

The term maintainability corresponds to the ability to update or modify the system under test. This is a very important parameter as the system is subjected to changes throughout the software life cycle.

To make Maintainability Testing more effective, testers should include static analysis and reviews as these are hard to spot during dynamic testing while it is easily captured in code walkthrough and inspection.

“Maintainability means fixing, updating, servicing and to modify the system or update the software for performance improvements or for the correction of faults”

Software maintenance is required when the customer demands new features and new functions in the software. Sometimes maintenance is required when the hardware of the system is changed then the modification of software is needed.

Market conditions and organization changes are also the reasons for software modification. It also includes that when the issue is detected, immediately fix it before it becomes a big problem.

**Software maintainability consists of four types**.

# 1.CORRECTIVE

Corrective maintenance is defined as maintenance of bugs or errors. It means when the error is detected in the software then the corrective maintenance is required to fix it. These bugs or errors are responsible for the faults which may appear in the code, design or logic of the software. Sometimes the user asks for the enhancements of the software and not about fixing the bugs. Corrective maintenance requires the correction of existing [faults](https://t4tutorials.com/software-fault-tolerance-mcqs-questions-answers/) in the software. Sometimes a change in hardware also cause bugs or errors.

# 2.ADAPTIVE

Adaptive maintenance includes the environmental changes where your software is living. Changes to the hardware, operating system, software dependencies, and organizational business rules and policies are handled in adaptive maintenance. By these modifications to the environment, changes can occur in the other parts of the software. In changing circumstances adaptive maintenance is required to keep your software fresh or to increase the lifetime of the software.

# 3.PERFECTIVE

Perfective maintenance refers to the changes in features and requirements in your existing system. After sometime when user suggests for new features and new functionality of the software than adaptive maintenance is used. In adaptive maintenance, some features are removed from the software which features are not effective for the software. Adaptive maintenance involves 50-55% of the maintenance work.

# 4.PREVENTIVE

Perfective maintenance maximizes the [maintainability](https://t4tutorials.com/software-evolution-and-maintenance-mcqs-questions-answers/) or understanding of the software system. Documentation updating or code optimizing are involved in preventive maintenance. Preventive maintenance helps the software to become more scalable, stable, understandable, maintainable. This maintenance acts as medicine to prevent the problems. Restructuring the data and code of the software are implemented in preventive maintenance.

## **Maintainability Testing Checklist:**

* Verifying the development standards such as structured programming, standards for database approach, recognizable nomenclature and standards for the user interfaces
* Verify if the data processing split up into sub transactions?
* Verify if the input, the processing and the output have been implemented separately
* Verify if the programs have been parameterized under necessary conditions to promote reusability.
* Verify if the systems are distributed.
* Verify if the algorithms are optimized.

## **ADVANTAGES OF SOFTWARE MAINTAINABILITY**

* Performance improvements
* Various bug fixing
* Up to date with current trends
* No need to spend extra bucks
* Contributes positively for the reputation of companies

## **DISADVANTAGES OF SOFTWARE MAINTAINABILITY**

* More money upfront
* Over-maintenance
* More workers
* **2.11.4 Portability :-** It is a measure of how easily an application can be transferred from one computer environment to another. A computer software application is considered portable to a new environment if the effort required to adapt it to the new environment is within reasonable limits. The meaning of the abstract term 'reasonable' depends upon the nature of the application and is often difficult to express in quantifiable units.

The phrase *"to port" means to modify software and make it adaptable to work on a different computer system*. *For example, to port an application to Linux means to modify the program so that it can be run in a Linux environment.*  
Portability refers to the ability of an application to move across environments, not just across platforms. To clarify, a computer platform generally refers to the operating system and computer hardware only. A computer environment is much broader and may include the hardware, the operating system and the interfaces with other software, users and programmers.

Portability testing is a process of testing with ease with which the software or product can be moved from one environment to another. It is measured in terms of maximum amount of effort required to transfer from one system to another system.

The portability testing is performed regularly throughout the software development life cycle in an iterative and incremental manner.

## **Portability Testing attributes:**

Following are the attributes of the portability Testing:

* Adaptability
* Install ability
* Replace ability
* Co-existence

## **Portability Testing Checklists:**

* Verify if the application is able to fulfill the portability requirements.
* Determine the look and feel of the application in the various browser types and various browser versions.
* Report the defects to the development teams so that they can be associated and defects can be fixed.
* The failures during the portability testing can help to identify defects that were not detected during unit and integration testing.
* **2.11.5 Security :-**

**Security Testing** is a type of Software Testing that uncovers vulnerabilities, threats, risks in a software application and prevents malicious attacks from intruders. The purpose of Security Tests is to identify all possible loopholes and weaknesses of the software system which might result in a loss of information, revenue, repute at the hands of the employees or outsiders of the Organization.

Security testing is an integral part of software testing, which is used to discover the weaknesses, risks, or threats in the software application and also help us to stop the nasty attack from the outsiders and make sure the security of our software applications.

The primary objective of security testing is to find all the potential ambiguities and vulnerabilities of the application so that the software does not stop working. If we perform security testing, then it helps us to identify all the possible security threats and also help the programmer to fix those errors.

It is a testing procedure, which is used to define that the data will be safe and also continue the working process of the software.

## **Why Security Testing is Important?**

The main goal of Security Testing is to identify the threats in the system andmeasure its potential vulnerabilities, so the threats can be encountered and the system does not stop functioning or can not be exploited. It also helps in detecting all possible security risks in the system and helps developers to fix the problems through coding.

**Types of Security Testing:**

There are seven main types of security testing as per Open Source Security Testing methodology manual. They are explained as follows:

**Vulnerability Scanning**: This is done through automated software to scan a system against known vulnerability signatures.

**Security Scanning:** It involves identifying network and system weaknesses, and later provides solutions for reducing these risks. This scanning can be performed for both Manual and Automated scanning.

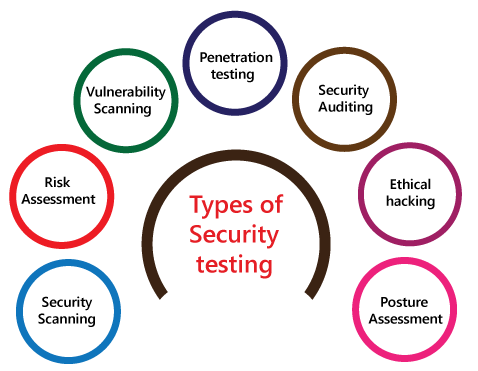
**Penetration testing**: This kind of testing simulates an attack from a malicious hacker. This testing involves analysis of a particular system to check for potential vulnerabilities to an external hacking attempt.

**Risk Assessment:** This testing involves analysis of security risks observed in the organization. Risks are classified as  Low, Medium and High. This testing recommends controls and measures to reduce the risk.

**Security Auditing:** This is an internal inspection of Applications and Operating systems for security flaws. An audit can also be done via line by line inspection of code

**Ethical hacking:** It’s hacking an Organization Software systems. Unlike malicious hackers, who steal for their own gains, the intent is to expose security flaws in the system.

**Posture Assessment:** This combines Security scanning,[Ethical Hacking](https://www.guru99.com/ethical-hacking-tutorials.html)and Risk Assessments to show an overall security posture of an organization.



**Principle Security Testing** is a testing technique to determine if an information system protects data and maintains functionality as intended. It also aims at verifying 6 basic principles as listed below:

* Confidentiality
* Integrity
* Authentication
* Authorization
* Availability
* Non-repudiation

## Security Testing

### **Availability**

In this, the data must be retained by an official person, and they also guarantee that the data and statement services will be ready to use whenever we need it.

### **Integrity**

In this, we will secure those data which have been changed by the unofficial person. The primary objective of integrity is to permit the receiver to control the data that is given by the system.

The integrity systems regularly use some of the similar fundamental approaches as confidentiality structures. Still, they generally include the data for the communication to create the source of an algorithmic check rather than encrypting all of the communication. And also verify that correct data is conveyed from one application to another.

### **Authorization**

It is the process of defining that a client is permitted to perform an action and also receive the services. The example of authorization is Access control.



### **Confidentiality**

It is a security process that protracts the leak of the data from the outsider's because it is the only way where we can make sure the security of our data.

### **Authentication**

The authentication process comprises confirming the individuality of a person, tracing the source of a product that is necessary to allow access to the private information or the system.



### **Non- repudiation**

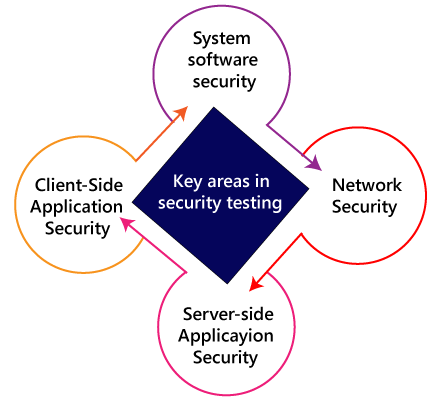
It is used as a reference to the digital security, and it a way of assurance that the sender of a message cannot disagree with having sent the message and that the recipient cannot repudiate having received the message.

The non-repudiation is used to ensure that a conveyed message has been sent and received by the person who claims to have sent and received the message.

### **Key Areas in Security Testing**

### While performing the security testing on the web application, we need to

### concentrate on the following areas to test the application:



### **System software security**

In this, we will evaluate the vulnerabilities of the application based on different software such as **Operating system, Database system**, etc.

### **Network security**

In this, we will check the weakness of the network structure, such as **policies and resources**.

### **Server-side application security**

We will do the server-side application security to ensure that the server encryption and its tools are sufficient to protect the software from any disturbance.

### **Client-side application security**

In this, we will make sure that any intruders cannot operate on any browser or any tool which is used by customers.

## **Security Testing - Techniques:**

* Injection
* Broken Authentication and Session Management
* Cross-Site Scripting (XSS)
* Insecure Direct Object References
* Security Misconfiguration
* Sensitive Data Exposure
* Missing Function Level Access Control
* Cross-Site Request Forgery (CSRF)
* Using Components with Known Vulnerabilities
* Un validated Redirects and Forwards
* **2.11.6 Localization & Internationalization :-**

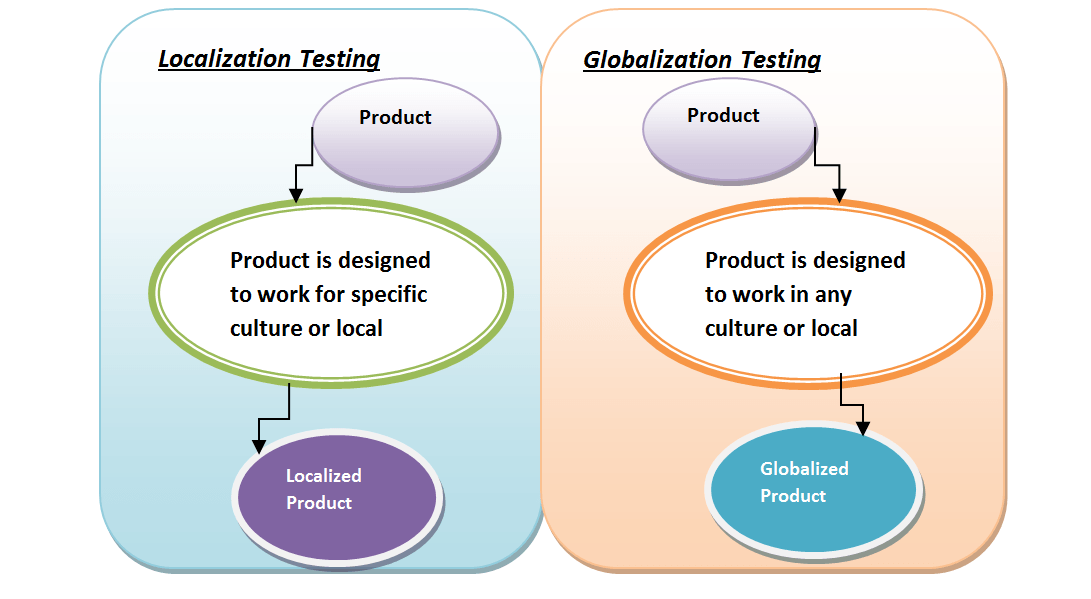
## **Localization Testing:-**

Localization testing is the software testing process for checking the localized version of a product for that particular culture or locale settings. The areas affected by localization testing are UI and content.

It is nothing but a format of [software testing](https://www.javatpoint.com/software-testing-tutorial). We test the particular application based on the country, region, etc. As we know, the Localized product only supports the precise kind of languages that are usable only in a specific region.

## **Globalization Testing**

**Globalization Testing** is a software testing method used to ensure that the software application can function in any culture or locale (language, territory or code page) by testing the software functionalities using each type of international input possible. The purpose of Globalization testing is to ensure that software can be used internationally or worldwide. It is also called Internationalization Testing.



It is another type of software testing used to test the software developed for multiple languages, is called [**globalization testing**](https://www.javatpoint.com/globalization-testing), and improving the application or software for various languages is known as **globalization**.

Globalization testing ensures that the application will support multiple languages and multiple features because, in current scenarios, we can see the enhancement in several technologies as the applications are planned to be used globally.

**Localization Testing:**

* Localization is a process of adapting internationalized software for a specific region or language by adding local specific components and translating text.
* Localization translates the product UI and occasionally changes some initial settings to make it suitable for another region. Localization testing checks the quality of a product's localization for a particular target culture/locale. This test is based on the results of globalization testing, which verifies the functional support for that particular culture/locale. Localization testing can be executed only on the localized version of a product.
* Localizability testing does not test for localization quality.
* The test effort during localization testing focuses on:
* Areas affected by localization, such as UI and content
* Culture/locale-specific, language-specific, and region-specific areas
* In addition, localization testing should include:
* Basic functionality tests
* Setup and upgrade tests run in the localized environment
* Plan application and hardware compatibility tests according to the product's target region.
* You can select any language version of Windows 2000 as a platform for the test. However, you must install the target language support.
* The localization testing of the user interface and linguistics should cover items such as:
* Validation of all application resources
* Verification of linguistic accuracy and resource attributes
* Typographical errors
* Consistency checking of printed documentation, online help, messages, interface resources, command-key sequences, etc.
* Confirmation of adherence to system, input, and display environment standards
* User interface usability
* Assessment of cultural appropriateness
* Checking for politically sensitive content
* When shipping a localized product, ensure that localized documentation (manuals, online help, context help, etc.) is included. Items to check include:
* The quality of the translation
* The completeness of the translation
* Terminology is used consistently in all documents and application UI.

**Internationalization Testing:**

* Internationalization is the process of designing a software application so that it can be adapted to various languages and regions without engineering changes.
* Internationalization is the adaptation of products for potential use virtually everywhere, while localization is the addition of special features for use in a specific locale.
* Internationalization is done once per product, while localization is done once for each combination of product and locale. The processes are complementary, and must be combined to lead to the objective of a system that works globally.
* ***What is Globalization (Internationalization) Testing?***
* Globalization definition:
* Globalization Testing is testing process to check whether software can perform properly in any locale or culture & functioning properly with all types of international inputs and steps to effectively make your product truly global.
* This type of testing validates whether the application is capable for using all over the world and to check whether the input accepts all the language texts.
* It is also called as “G11N“, because there as 11 characters in between G & N.
* It ensures that the product will handle international support without breaking functionality.
* Globalization testing mainly focuses on the functionality of the product with any culture/locale settings and every type of possible international input.
* It also helps uncover issues that may increase the costs of localization and future product support later on.
* Internationalization testing is the process of verifying the application under test to work uniformly across multiple regions and cultures.
* The main purpose of internationalization is to check if the code can handle all international support without breaking functionality that might cause data loss or data integrity issues.
* Globalization testing verifies if there is proper functionality of the product with any of the locale settings.
* Internationalization Checklists:
* Testing to check if the product works across settings.
* Verifying the installation using various settings.
* Verify if the product works across language settings and currency settings.

## **Globalization Testing Vs Localization Testing :-**

|  |  |
| --- | --- |
| **Globalization Testing** | **Localization Testing** |
| Globalization testing checks the proper functioning of the product, using every type of international input possible. It ensures that without breaking functionality the code can handle all international support. For example I18N, is the process of planning and implementing products and services so that they can easily be adapted to specific languages and culture. | Localizing testing is done to ensure the quality of a product for a particular target or locale. For example, for French users, the testing product is denoted as L10N. |
| In a globalized product, a code is separated from the messages or information. With the help of globalization, it enables software to be used with different languages without having to redesign the complete software. | This is not necessary for a Localized product |
| Globalization focuses your application’s capabilities on users as the generic user base. | Localization focuses on a subset of users in a given culture or locale. |
| Separation of testers from translators and engineers, ensuring a thorough and impartial approach. | It helps to reduce a time for testing since it’s done for just on locale |
| Formalized bug reporting | It reduces overall testing and support costs |
| Detect potential problems in application design that could inhibit globalization | Validation of all application resources |
| It ensures that without breaking functionality code can handle all international support | Verification of linguistic accuracy and resource attributes. Check Typographical errors |
| Compatibility tests of hardware and application according to the product’s target region | Confirmation of input and display environment standards, adherence to the system. Usability of User Interface |

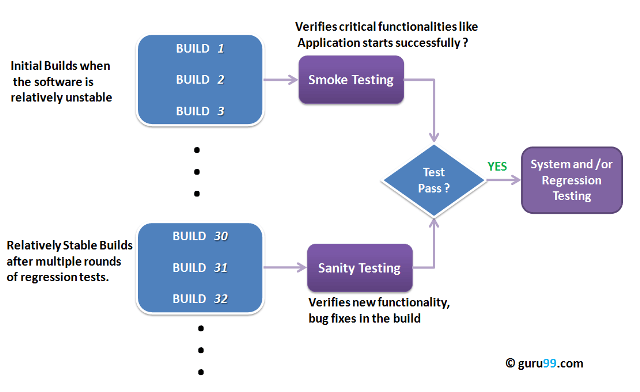
Using globalization testing and localization testing, we can test some of the most important software features, such as linguistic relevance, Cultural sensitivity, and the software's global and locale appeal. With the help of these components, the software can be used by the global and local audience and increase the popularity of the software.

The globalization testing and localization testing will help the testing team takes required procedures to enhance the performance, quality, functionality, and other significant fundamentals of the software product.

Lastly, we can say that if we do not perform globalization testing and localization testing techniques, the software engineers cannot develop a good quality software product, which meets the necessities of the end-user and market.

* Globalization focuses on the application’s world-wide capabilities whereas localization focuses on a subset of users in a given culture or locale.
* In a globalized product, the code is separated from the messages or information while it is not necessary for the localized product
* Globalization ensures that without breaking functionality, code can handle all international support, whereas Localization ensures verification of linguistic accuracy and resource attributes.
* Globalization detects potential problems in application whereas localization provides validation of all application resources.
* **2.12 :- Concept of Smoke testing and Sanity Testing :-**

The key differences between Smoke Testing and Sanity Testing can be learned with the help of the following diagram –



To appreciate the above diagram lets first understand –

**What is a Software Build?**

If you are developing a simple computer program which consists of only one source code file, you merely need to compile and link this one file, to produce an executable file. This process is very simple.  
Usually, this is not the case. A typical Software Project consists of hundreds or even thousands of source code files. Creating an executable program from these source files is a complicated and time-consuming task.  
You need to use “build” software to create an executable program and the process is called ” *Software Build*”

## **What is Release?**

In **software testing**, it is very common terminology used on an everyday basis. The **Release** is a final product or a project, which is delivered to the customer.

It involves the complete activities from the **Requirement, Designing, Development, and Testing phases** until it is handed over to the customer.

In other words, we can say that a release is an entirely developed application, while the build is the part of an application or the software.

## **Smoke Testing :-**

**Smoke Testing** is a software testing technique performed post software build to verify that the critical functionalities of software are working fine. It is executed before any detailed functional or regression tests are executed. The main purpose of smoke testing is to reject a software application with defects so that QA team does not waste time testing broken software application.

In [Smoke Testing](https://www.guru99.com/smoke-testing.html), the test cases chose to cover the most important functionality or component of the system. The objective is not to perform exhaustive testing, but to verify that the critical functionalities of the system are working fine.  
For Example, a typical smoke test would be – Verify that the application launches successfully, Check that the GUI is responsive … etc.

It is a type of testing that guarantees an application's basic and critical features are working fine before doing exhaustive testing or rigorous testing.

Smoke testing is also known as a subcategory of [acceptance testing](https://www.javatpoint.com/acceptance-testing) or Build Verification testing.

In other words, we can say that smoke testing is used to test all the functionality of the software product or check whether the build is broken or not.

In smoke testing, we only perform positive testing, which implies that we can enter only the valid data not invalid data.

Whenever a new build comes in, we always start with smoke testing because some changes might have broken a major feature of a new build.

## 

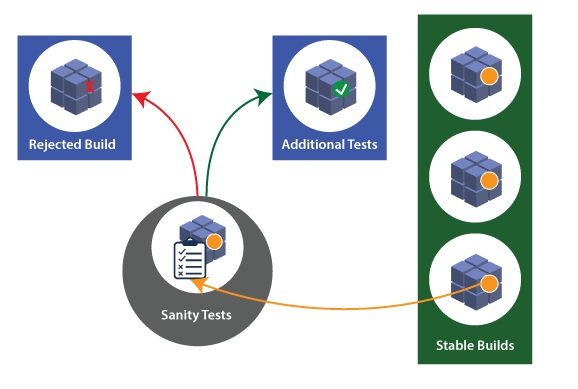
## **What is Sanity Testing?**

Sanity testing is a kind of Software Testing performed after receiving a software build, with minor changes in code, or functionality, to ascertain that the bugs have been fixed and no further issues are introduced due to these changes. The goal is to determine that the proposed functionality works roughly as expected. If sanity test fails, the build is rejected to save the time and costs involved in a more rigorous testing.

The objective is “not” to verify thoroughly the new functionality but to determine that the developer has applied some rationality (sanity) while producing the software. For instance, if your scientific calculator gives the result of 2 + 2 =5! Then, there is no point testing the advanced functionalities like sin 30 + cos 50.

It is performed to check whether the bugs have been fixed after the build. Generally, Sanity testing is performed on stable builds. It is also known as a variant of regression testing.

The initial aim of performing sanity testing is to determine that the planned features work roughly as expected. If the sanity test fails, the build is rejected to save the costs and time complex in more severe testing.

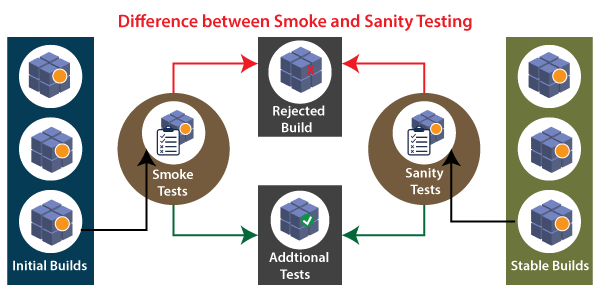
****

Whenever we talk about **smoke and sanity testing**, we know that these two types of testing are similar, but both smoke testing and sanity testing have their purposes and significance.

* Smoke Testing is scripted, which means it can be documented, whereas the sanity testing is unscripted, which implies that it cannot be documented.
* Smoke testing is considered shallow and wide testing, and on the other hand, sanity testing is considered narrow and deep testing.
* Smoke testing takes all important features and performs high-level testing, whereas sanity testing takes some very significant features and performs deep testing.
* Smoke testing is executed as soon as the build is installed, and on the other hand, sanity testing is implemented as soon as the bug fixes are done.

## **The key difference between Smoke Testing and Sanity Testing**

The below facts explain the differences between **smoke and sanity testing**:



The below comparison table enlightens the important differences between smoke testing and sanity testing in a quick manner:

|  |  |  |  |
| --- | --- | --- | --- |
| **S.No.** | **Comparison Basis** | **Smoke Testing** | **Sanity Testing** |
| 1 | Test coverage | It is a broad approach to testing where all parts of the application are tested. | It is a narrow approach to testing where specific parts of the application are tested. |
| 2 | Measures | It measures the stability of the system by performing rigorous testing. | It measures the rationality of the system by performing rigorous testing. |
| 3 | Technique | Smoke testing can be either manual or automated. | Sanity testing can be done without test cases or scripts. |
| 4 | Executed by | It is performed by both testers and developers. | It is performed by only testers. |
| 5 | Purpose | Testing is done without getting into deep but whenever needed tester has to go into deep. | Sanity testing does not need to go into deep of the application. |
| 6. | Performed at | Smoke testing is the first testing performed on the initial build. | Sanity testing is performed when the build is comparatively stable. |
| 7 | Documentation | Smoke testing is documented. | Sanity testing is not documented. |
| 8 | Used to | It is used to test End to End function of the application. | It is used to test only modified or defect fixed functions. |
| 9 | Subset | It is considered as a subset of acceptance testing. | It is considered as a subset of regression testing. |

|  |  |
| --- | --- |
| **Smoke Testing** | **Sanity Testing** |
| Smoke Testing is performed to ascertain that the critical functionalities of the program is working fine | Sanity Testing is done to check the new  functionality/bugs have been fixed |
| The objective of this testing is to verify the “stability” of the system in order to proceed with more rigorous testing | The objective of the testing is to verify the “rationality” of the system in order to proceed with more rigorous testing |
| This testing is performed by the developers or testers | Sanity testing in software testing is usually  performed by testers |
| Smoke testing is usually documented or scripted | Sanity testing is usually not documented  and is unscripted |
| Smoke testing is a subset of Acceptance testing | Sanity testing is a subset of [Regression Testing](https://www.guru99.com/regression-testing.html) |
| Smoke testing exercises the entire system from end to end | Sanity testing exercises only the particular  component of the entire system |
| Smoke testing is like General Health Check Up | Sanity Testing is like specialized health check up |