UPES

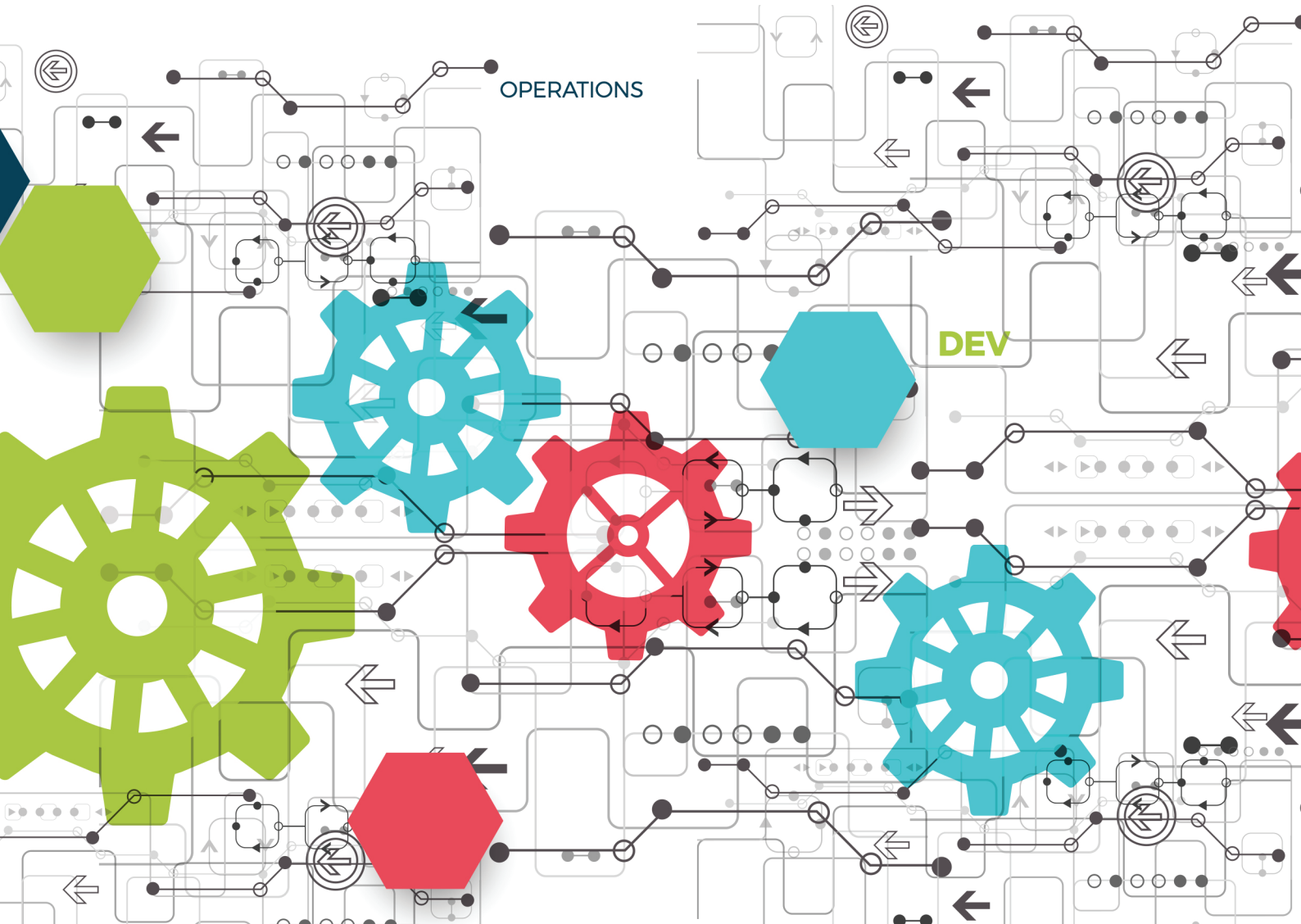
Xebia

YEAR 3

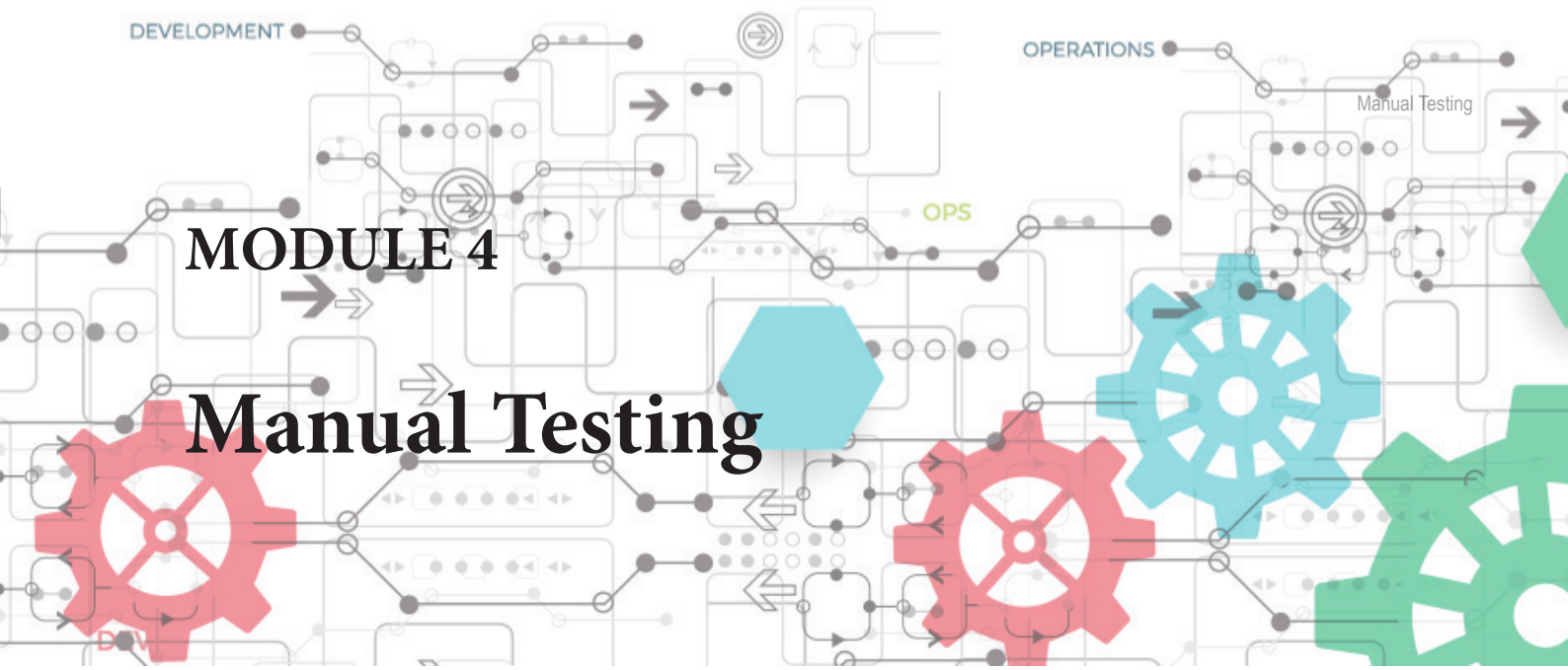
B.Tech Computer Science and Engineering in DevOps

Test Automation

MODULE 4
Manual Testing



Contents

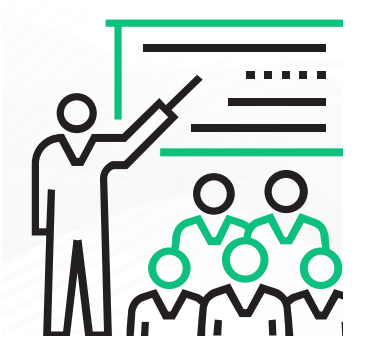


MODULE 4

Manual Testing

You will learn about the ‘Manual Testing’ and its best approaches

Module Objectives



At the end of this module, you will be able to learn:

- Define Manual Testing
- Define Automation Testing
- Understand Unit Testing
- Explain Integration Testing
- Describe Smoke-Sanity Testing

You will be informed about the module objectives.

At the end of this module, you will be able to learn:

- Define Manual Testing
- Define Automation Testing
- Understand Unit Testing
- Explain Integration Testing
- Describe Smoke-Sanity Testing

Module Topics

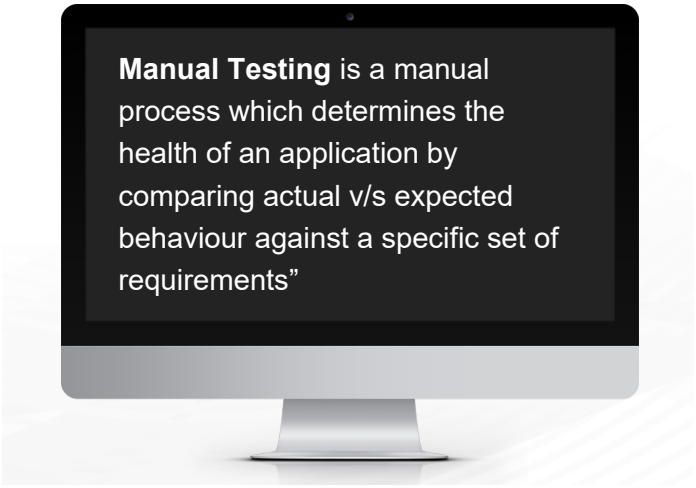
Let us take a quick look at the topics that we will cover in this module:

- Manual Testing.
- Automation Testing.
- Unit Testing.
- Integration Testing.
- Smoke-Sanity Testing.

You will learn about the following topics in this module:

- Manual Testing.
- Automation Testing.
- Unit Testing.
- Integration Testing.
- Smoke-Sanity Testing.

1.4.1 Manual Testing



As the name suggests ‘Manual’ which means testers will be ‘Manual Testers’ and the testing they will perform is ‘Manual Testing’. Indirectly it means manual testers will have knowledge of the application and they will conduct the manual testing without the use or help of any Automation Testing tool.

Manual Testing has been the most primitive form of testing. Testing started with Manual Testing approach only. And any new application must undergo a thorough Manual Testing before checking the feasibility of Automation Testing

Manual Testing approach opened gates to other testing approaches. Like all other approaches this also means that the application should behave in accordance with the requirements and any deviation is reported.

Manual Testing includes a lot of Testing Artifacts which are used from the beginning till the

end of the Testing Life Cycle. Sometimes they are hard to maintain, are boring and time consuming. But all these boring bits have their special place

Simply put ***"Manual Testing is a manual process which determines the health of an application by comparing actual v/s expected behaviour against a specific set of requirements"***

1.4.1 Manual Testing – How to Approach?

Listed below are the steps for approaching manual testing :

Only following a Requirement Document is not enough for a good manual testing project. There has to be proper review and sign-off with the latest version of the Requirement Document to be read and followed by the testers. Proper communication channels should be opened so that testers can ask doubts or raise queries in case they find certain ambiguity in the requirement document. Asking questions and raising doubts should be highly encouraged. Testers should also be allowed to give suggestions to improve the document. Once the finalized document is with the testers, testing can begin.

All the Testing Artifacts should be prepared. The Test Strategy document along with the Test Plan which defines the Scope is of highest importance. With Agile methodologies followed in almost every Software Company today, the good old practice of Test Strategy and Test Plan is getting sidelined but the same action is creating a lot of conflict and last moment defect creep-in for a lot of Projects. hence, it is always advisable to prepare a good solid Test Strategy which defines the Scope precisely. The Requirement Traceability Matrix (RTM) and the TCER (Test Conditions Expected Results) and other documents should be formulated depending on organization approved template formats.

Test Cases writing is probably the heart of the Manual Testing process. A good, simple, full coverage of scenarios Test Case is very important. The linguistics should be simple and well understood, the intent should be clearly defined and it should cover all aspects of functionality. Coverage should not be compromised.

Review, Inspection and Walkthrough are generally skipped in almost all software companies because of time crunch and high paced deliverable environments and methodologies. But a review from a senior always helps in identifying problems beforehand which will save lot of time later in the Project. Junior Testers or less experienced testers should always get their work reviewed from senior testers.

Test Execution should be done in accordance to the test case prepared. Sometimes, manual testers think they know in and out of the application and they can test the application on their own and they do not need to execute test cases. This leads to certain defect leakage later. So test execution should be done precisely with respect to the test cases that have been designed.

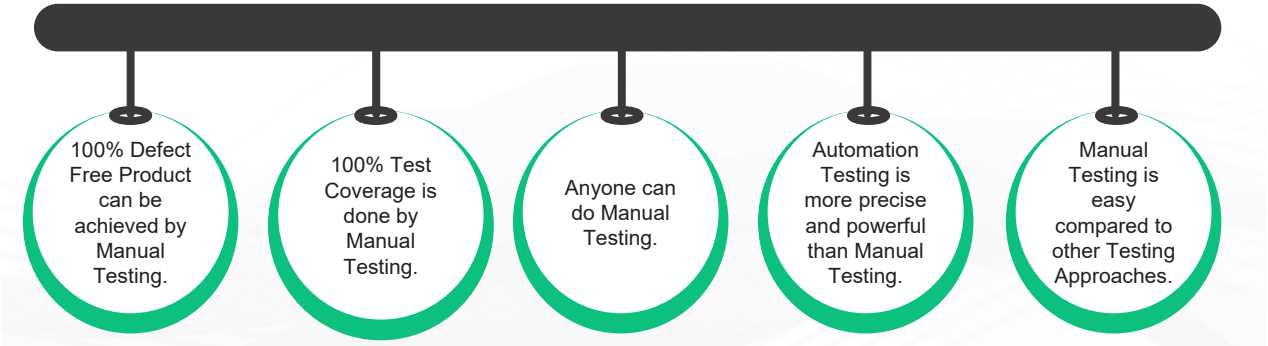
Defect/Bug reporting – this is another very important branch of manual testing. Finding a bug is not the only responsibility of a manual tester but reporting the same in correct channels is also very important and this should be done with precision as sometimes developers and other depending teams tend to deny responsibility if the bug reported is not understood or is not reproducible.

Following up with the status quo of the defect whether it is still in open state or it is fixed or it has some discrepancy is also important. Closer vigil needs to be kept. In case, the defect has been fixed and it has been re-assigned to the tester then the tester should test the defect in the requisite environment and report its current status immediately.

Test Summary – These reports are good practices to actually give a depiction of (how+what+when+by whom) things happened during the course of manual testing.

1.4.1 Manual Testing – Myth and fallacy

Manual testing is generally treated as most easy job. Let's look at some myths of manual testing.



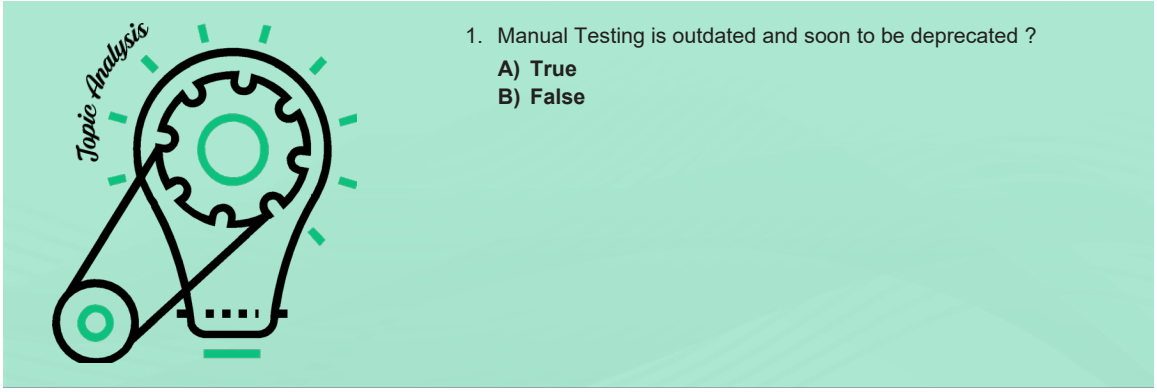
No application can be 100% defect free. Take for any product which we use in our day to day life. Even Google and Facebook and Amazon have defects. They are always evolving. Hence, no one can say that manual testing will cover and identify all the defects. It is a fallacy. Same is true about coverage. Try as hard as may, some or the other scenario will get skipped or will not come in terms of thought process of testing team. Manual Testers do tend to work hard and do their best to have maximum test coverage but somehow labeling that 100% test coverage is possible is a pure myth.

Anyone can do Manual Testing as if it is a thankless job. This is really not true. Manual Testing needs skills and not one but multiple skill sets which we shall discuss later.

Automation Testing is powerful – Agreed but it cannot be said that Automation Testing is more powerful than Manual Testing. As manual testing can uncover scenarios and defects which Automation Testing might not be able to because of coding constraints or some logic which will be hard to design.

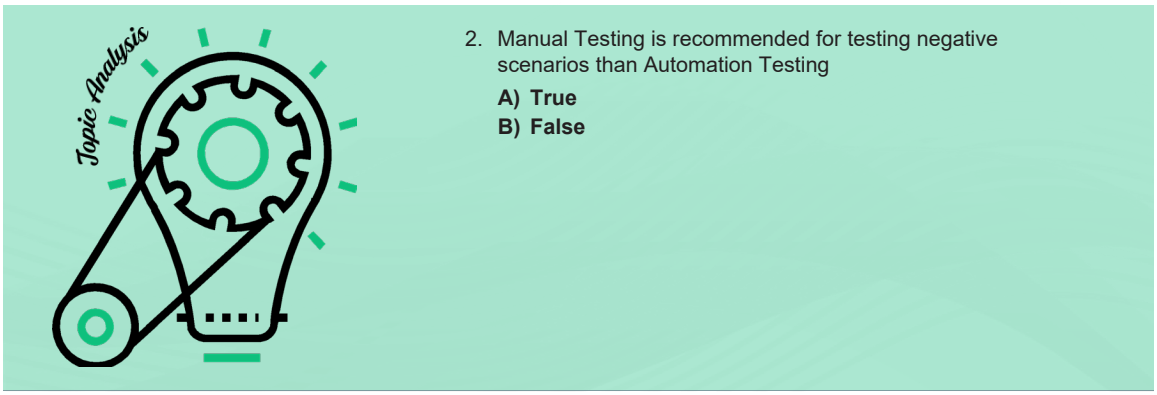
Manual Testing is not easy. It requires skills, thorough understanding and certain functional and technical approaches to get the job done. And testers are more at risk compared to developers as they need to authenticate the product.

What did you learn so far ?



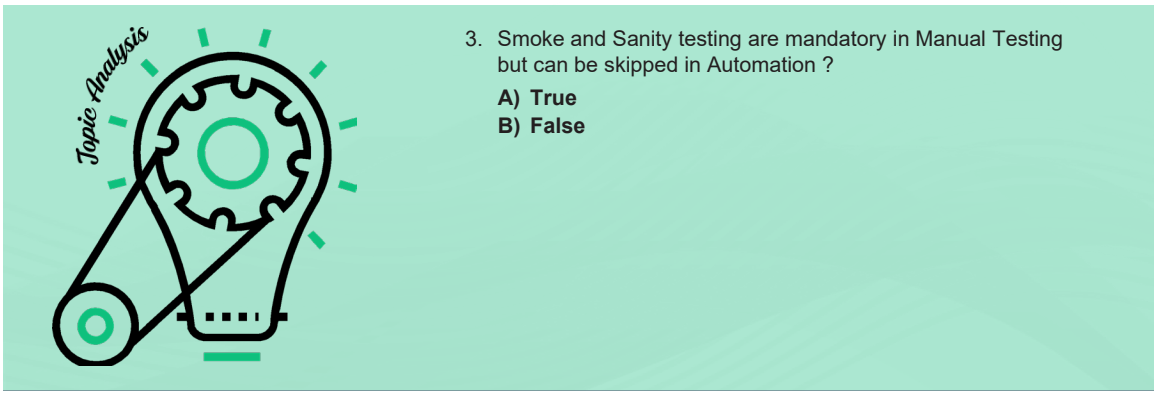
1. Manual Testing is outdated and soon to be deprecated ?
   A) True
   B) False

Answer: 1 : B



2. Manual Testing is recommended for testing negative scenarios than Automation Testing
   A) True
   B) False

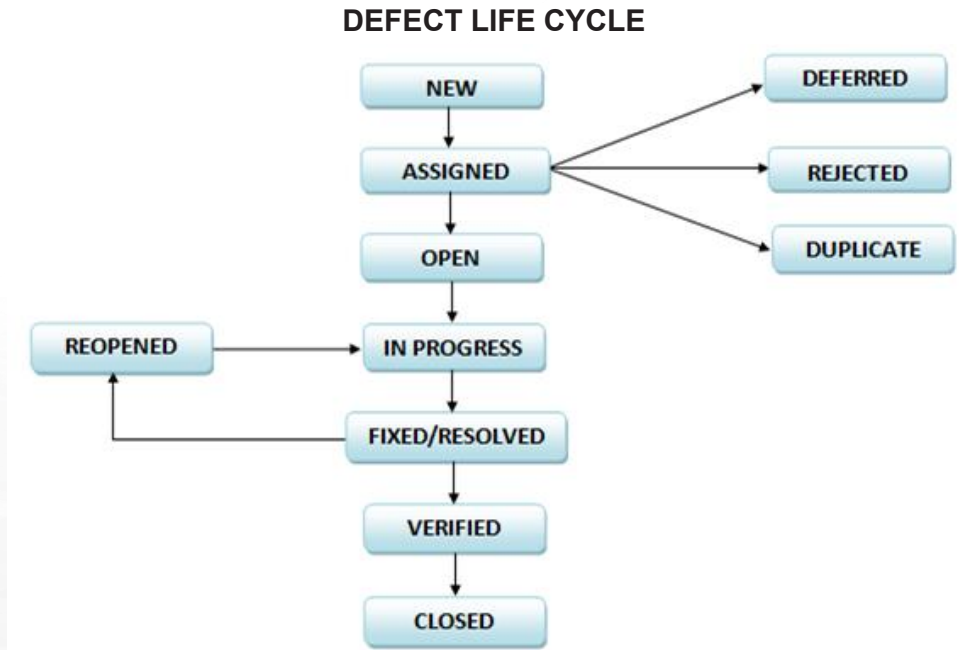
Answer: 2 : A



3. Smoke and Sanity testing are mandatory in Manual Testing but can be skipped in Automation ?
   A) True
   B) False

Answer: 3 : B

1.4.1 Manual Testing – Defect Life Cycle



The above image shows the workflow.Ok let's say for example you are a tester and you found a defect. And this defect has never been found before.

NEW – The first state of the defect.

ASSIGNED – Now you will assign this to a certain developer or development team. So the status of the defect will become assigned from new.

OPEN – When the developer has acknowledged the defect, the status of the defect now becomes open.

INPROGRESS–Whenthedeveloperstartsworkingonthedefect.ThestatebecomesInProgress.
FIXED/RESOLVED – Developer marks this defect as fixed once he has done necessary modifications in the code and has fixed the bug.

PENDING RETEST - After fixing the defect, the developer assigns the defect to the tester for retesting the defect at their end and till the tester works on retesting the defect, the state of the defect remains in 'Pending Retest'.

RETEST - At this point, the tester starts the task of working on the retesting of the defect to verify if the defect is fixed accurately by the developer as per the requirements or not.

REOPENED - If any issue still persists in the defect then it will be assigned to the developer again for testing and the status of the defect gets changed to 'Reopen'.

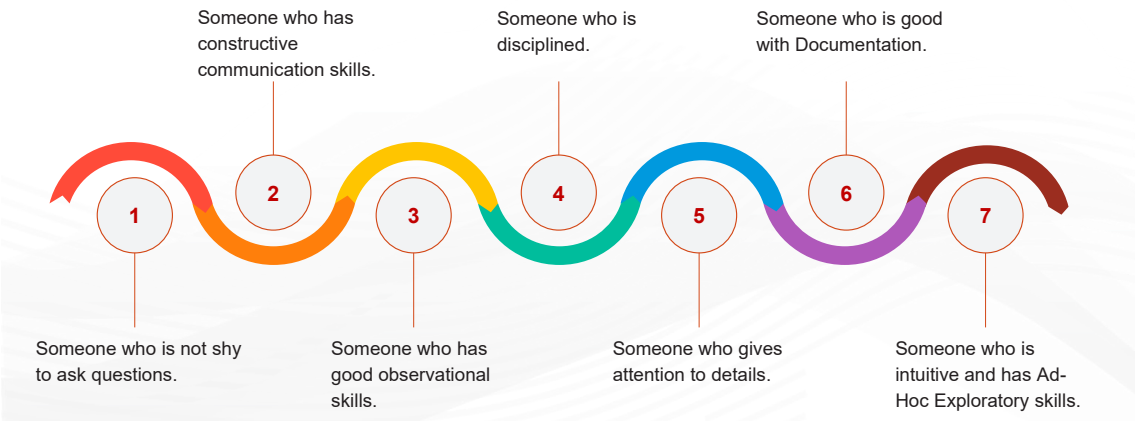
VERIFIED - If the tester does not find any issue in the defect after being assigned to the developer for retesting and the defect has been fixed accurately then the status of the defect gets assigned to 'Verified'.

CLOSED - When the defect does not exist any longer then the tester changes the status of the defect to 'Closed'.

There are chances that the developer might reject the defect if he finds it invalid.Sometimes some defects cannot be fixed in this iteration. It is kept under consideration to be fixed in the next release and is marked as **Deferred**.When the developers are not able to understand how to reproduce the defect and they mark it as **Could not Reproduce.**

1.4.1 Manual Testing – Qualities of a good Manual Tester

Some qualities of good manual testers:



What separates a good manual tester vs an average manual tester is not only dependent on skill factor but it also depends on the thought process and personal skills to enhance one own self to become a better software tester. The one aspect that is common for anyone is asking questions. When you ask questions – that means you are keen and you want to learn and see how things are and are more interested.

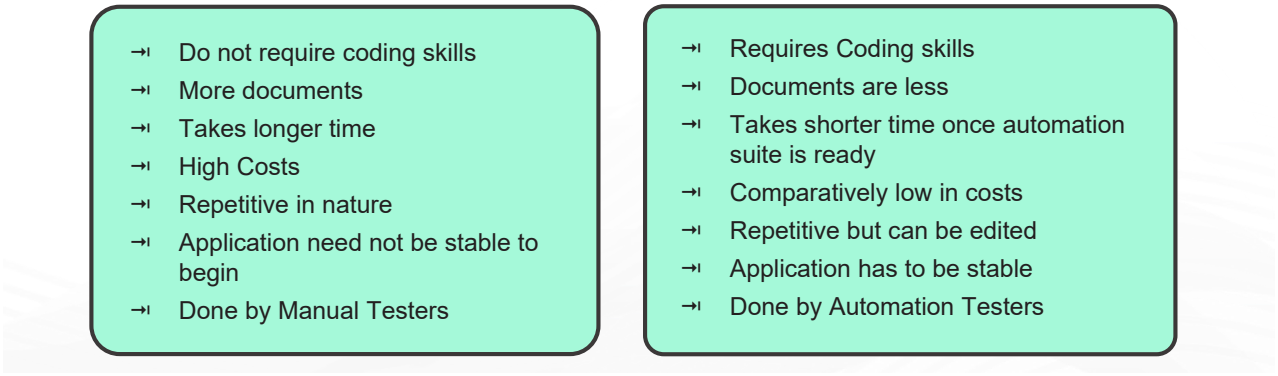
Testers should be able to do Constructive Communication. For testers it just becomes even more important as an average communication leads to extra intrusion of other people who will try to cover up for the lack of communication or err in communication. It gives good language in test documents and would be well appreciated by clients if they are of a Foreign Origin.

Observation is very important whether you are manually testing a web application or a mobile app. Yes it needs little experience to actually bring the observational skill to use but that eye is definitely going to help a manual tester to actually catch hold of any defect or deviation which might get overlooked otherwise

Disciplined approach and attention to details are very important aspects for a Manual Tester as manual testing involves quite a lot of maintenance and creation of documents. Sometimes the quantity of the same could be overwhelming. So a better organized tester is a better tester.

As mentioned in the above point, an organized tester (in this case with documentation) is prone to find answers to questions with lesser hassle. A good manual tester definitely needs to possess this quality of out of the blue doing something to find certain deviation even though all sorts of approvals have been given for the release of that product. It could be anything. The approach could be negative too but anything within the scope of testing should always be encouraged.

1.4.1 Manual Testing vs Automation Testing



In 2020, we are moving towards the Robotic frame of life where in everything is going to get automated or at least the step towards it, so why Software Testing stay behind. Cometh the need of the hour, even the Manual Testing approach is slowly moving towards the automation route.

But is the transition going to be that easy. What happens to Manual Testing then ? Will it be deprecated ? There are lot of questions to be answered and some of them quite genuine in that regard. Let's understand few facts what led to all this.

Software Applications with time have become quite complicated, Robust and the vastness also has exponentially increased. To handle the vastness, manual testing is not going to be an effective solution any more. Yes Manual Testing will have a big and important role to play in the first iteration but then the subsequent iterations will have to be assessed with something which will be time saving, more cost effective and less human intervention. Hence Automate it.

If we actually go by the book, yes Automation Testers need to hone their coding skills. Just looking into the Business Document and Functional Document and writing test cases will not suffice. Automation testers have to code in any of the language binding depending upon the automation testing tool which they are using

Automation Testing does help cut down costs and also saves a lot of time. Plus integration with CI/CD tools like JENKINS gives quite a peace of mind to let the test cases run automatically with just a timer set up.

The only major drawback is that if the application is unstable, then Automation Testing might not be effective rather it will be a complete waste of time.

1.4.1 Manual Testing - Types

The most common types of testing are listed below:

1. BlackBox Testing
2. WhiteBox Testing
3. GreyBox Testing
4. Unit Testing
5. Integration Testing
6. System Testing
7. Acceptance Testing
8. Operational Readiness Testing

The foremost thing to understand here is if the end goal is to determine defects, then why have so many forms and types. It is not that simple as just word makes it seem like. There is lot more to it and it is quite vast. Every stage of development in tandem with testing will have a certain way of looking into things. What Whitebox testing can determine, blackbox might not and what Acceptance testing can reveal a Blackbox testing might not. So every type has its own unique importance and that is what we will understand in further discussion

1.4.1 Manual Testing - Types

BlackBox Testing

This technique' focus is mostly on the external layer of the software such as technical specifications, design, and client's requirements to design test cases. The technique enables testers to develop test cases that provide full test coverage.

Black Box testing technique does not need the tester to have in depth coding knowledge of the application. What matters most here is the knowledge of possible inputs, logical operation and expected output. Having these 3 in the kitty, the Tester or Black Box Tester is good to go.

Let's discuss little bit about the various Black Box Testing Techniques:

1. **Boundary Value Analysis:-**

   Like any application logic becomes vulnerable and susceptible at boundaries even in

software applications. A Developer has more chances of introducing a faulty logic which might not be completely wrong but might be somewhat wrong at the border lines. Testing any application with the minimum value and maximum value parameter and transition or switch the next parameter should be done with this technique. There could be many real life examples like "***Checking February calendar for leap year, non leap year and checking the value in March", "checking for bulk items which might give a discount scheme from item 10th to 19th [so item no 19th, 20th and 21st will come under BVA] and then another discount scheme from item 20th to 29th [so item no 29th, 30th and 31st will come under BVA] "***

Similarly there could be multiple exams. So basic understanding is the minimum value, just below minimum value, maximum value and just above exact value – these areas needs to be tested.

2. Equivalence Partitioning:-

In this technique, the entire range of input data is divided into different partitions. All possible test cases are considered and divided into logical set of data. One test value is picked during each execution. So we divide the total input data into sets of data and then test them to find vulnerabilities. The only de-merit of this technique is that "it is not too helpful when fewer test cases should cover maximum requirements".

3. Decision Table Testing:-

In this technique, test cases are designed on the basis of the decision tables that are formulated using different combinations of inputs and their corresponding outputs based on various conditions and scenarios adhering to different business rules. It is used in both testing and requirements analysis. It manages complex business logics in simplified tabular format which gives a simplistic explanation. One advantage of using decision tables is that they make it possible to detect combinations of conditions that would otherwise not have been found and therefore, not tested or developed. The requirements become much clearer and you often realize that some requirements are illogical, something that is hard to see when the requirements are only expressed in text.

A disadvantage of the technique is that a decision table is not equivalent to complete test cases containing step-by-step instructions of what to do in what order. When this level of detail is required, the decision table has to be further detailed into test cases.

4. State Transition Testing:-

It is a Black Box Testing technique. The tester will give certain inputs [keep in mind the inputs can differ from positive to negative]. The tester determines the behavior of the application under certain permissible finite limits of inputs. And notices the system's response the moment the finite number of attempts gets exhausted.

Let's understand this via an example. You have an ATM Debit Card and you go to the ATM Machine.

Negative Attempts:-

1st Attempt – wrong pin and system denies access and warns you 2 more attempts left

2nd Attempt – wrong pin and system denies access and warns you 1 more attempt left

3rd Attempt – wrong pin and system denies access, blocks card and gives message “card has been blocked due to suspicious activity”

Positive Attempt:-

1st Attempt – right PIN and system acknowledges access

So here is that the state of the system changes with every attempt. This form of testing in real life scenario is called State Transition Testing

5. **Use Case Testing**:-

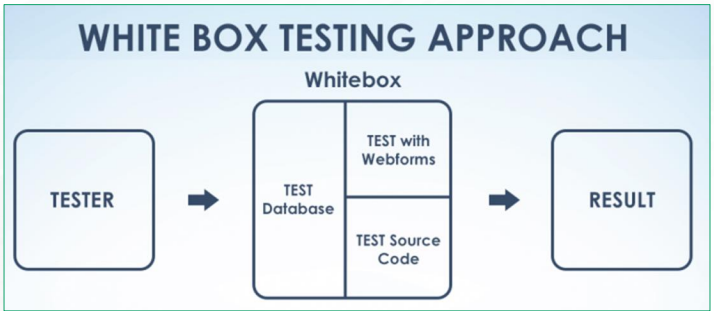
It is a descriptive form to address the functionality of the system at a user level. Specific negative and positive scenarios are not segregated. Just the intent of testing is directly mentioned in one line

For example : Login into the application using valid credentials and validating system response

Enter invalid credentials and attempting login and validating system response.

WhiteBox Testing

The structure-based or white-box technique design test cases based on the internal structure of the software. This technique exhaustively tests the developed code. Developers who have complete information of the software code, its internal structure, and design help to design the test cases.



White Box testing technique does need the tester to have in depth coding knowledge of the application. So generally developers act as Testers here. Unit Testing is an Example of White Box Testing which is done by developers

Let’s discuss little bit about the various White Box Testing Techniques

- **Statement Testing & Coverage**

  This technique involves execution of all the executable statements in the source code at least once. The percentage of the executable statements is calculated as per the given requirement. This is the least preferred metric for checking test coverage.

- **Decision Testing Coverage**

  This technique is also known as branch coverage is a testing method in which each one of the possible branches from each decision point is executed at least once to ensure all reachable code is executed. This helps to validate all the branches in the code. This helps to ensure that no branch leads to unexpected behavior of the application.

- **Condition Testing**

  Condition testing also is known as Predicate coverage testing, each Boolean expression is predicted as TRUE or FALSE. All the testing outcomes are at least tested once. This type of testing involves 100% coverage of the code. The test cases are designed as such that the condition outcomes are easily executed.

- **Multiple Condition Testing**

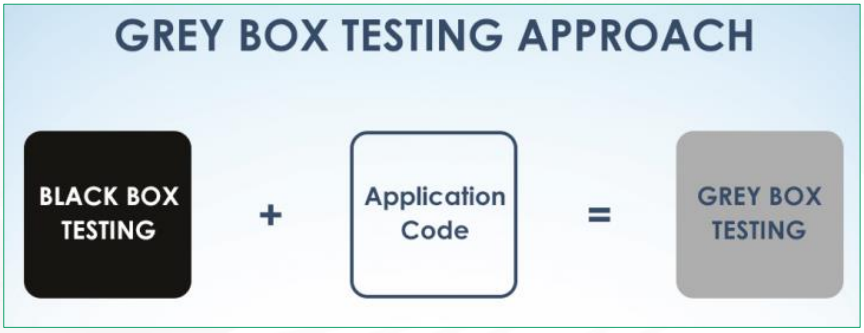
  The purpose of Multiple condition testing is to test the different combination of conditions to get 100% coverage. To ensure complete coverage, two or more test scripts are required which requires more efforts.

- **All Path Testing**

  In this technique, the source code of a program is leveraged to find every executable path. This helps to determine all the faults within a particular code.

GreyBox Testing

Unlike blackbox testing where the knowledge of business logic is not required and also complete knowledge in case of whitebox testing, greybox testing stands somewhere in the middle of both the approaches.



It is also done by Developers and experienced testers. It increases testing coverage by testing the layers of a system which is complex in nature. The most advantageous aspect of greybox testing can sometimes become quite a mess.

Anything with testing has a very basic aspect and that aspect in ‘input’. In the case of greybox testing, the developers will have their inputs and the testers will have their inputs and sometimes there is a clash as developers try to emphasize as their inputs being more accurate than testers.

But there are quite few advantages too. It reduces the overhead of long process of testing functional and non-functional types. It gives enough free time for a developer to fix defects.

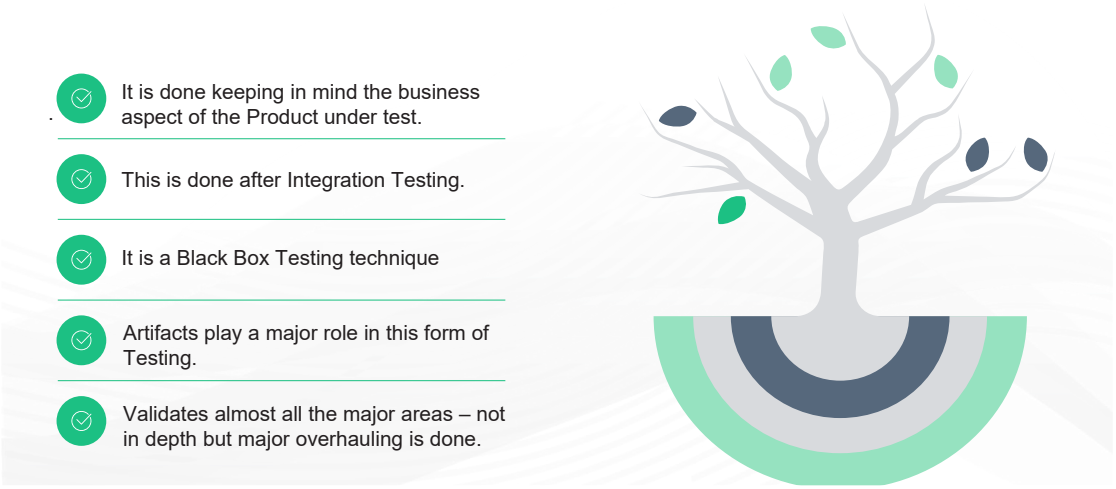
Some of the techniques which are used to perform Greybox testing are:-

- **Matrix Testing:** This testing technique involves defining all the variables that exist in their programs.
- **Regression Testing:** Testing to ensure the older version is working fine when the newer version has been introduced in the software.

- **Orthogonal Array Testing or OAT:** 100% code coverage is not possible and anyways developers tend to keep too busy to even ensure it happens or not. So in such scenarios developers write certain test cases which ensure maximum code coverage in a very smart way. This is OAT.
- **Pattern Testing:** Old artifacts are very important to determine old patterns of a certain application. They give lot of information about vulnerable areas, critical areas, areas which need intuitive testing techniques and areas which are generally stable and areas which are always stable. Sometimes what happens is in spite of reforming the source code, a system behaves very stubbornly because of the Architectural design complexity. So testing all these patterns is called Pattern Testing.

1.4.1 Manual Testing – System Testing

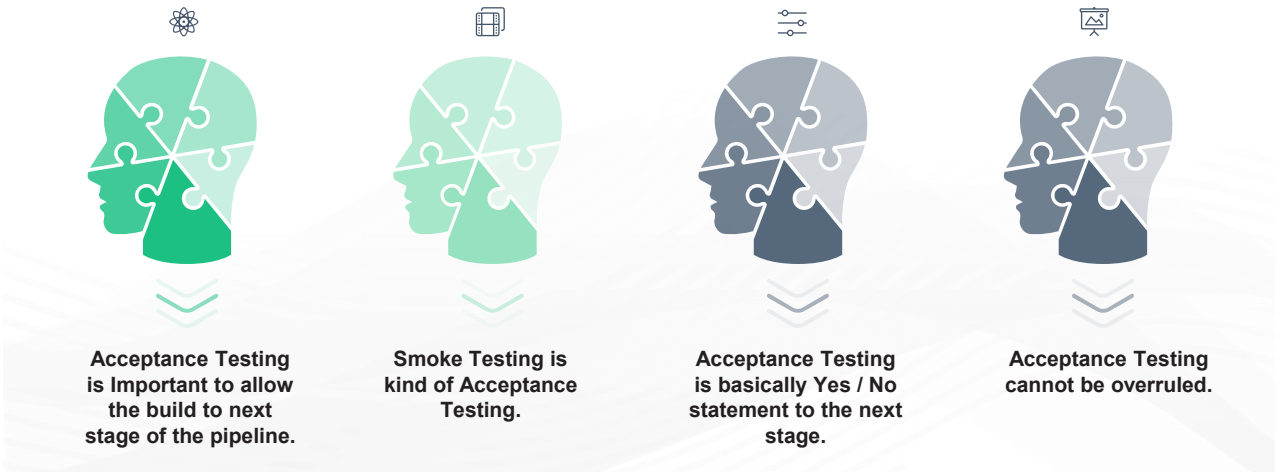
System testing is testing System as a single entity.



System testing is mostly done to ensure that all the previous testing efforts do not go wasted with what actually has come out as the intended product. Whether it serves the purpose of delivering to the client what the client needed.

Let's say you are launching a mobile phone in the market and all its individual components are working fine. Integrating them too is working fine. That is technical operations working fine. What about the functionality of usage. Does it work optimally or beyond? Can we improve it further? All these questions are answered in System Testing.

1.4.1 Manual Testing – Acceptance Testing

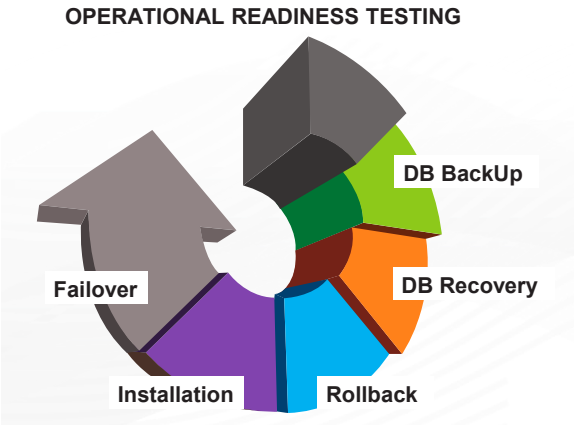


Acceptance testing can have various forms, even Smoke Testing is a form of Acceptance Testing. So anything which has an Entry Criteria and has an Exit Criteria is technically Acceptance Testing.

UAT is the most famous Acceptance testing in Software Testing. It is actually User Acceptance Testing done mostly by Business Analysts in the client environment to make sure all the functionalities are working fine.

1.4.1 Manual Testing – Operational Readiness Testing

Final stage of testing just before release



Operational Readiness Test is performed at final stage of testing when all other testing activities are performed and build is ready for live deployment.

The areas to focus are:-

DB BackUp – To make sure that we have the option to get the backup of any data in case there is a failure, a regular timer should be set to get this done.

DB Recovery – In case of loss of data or network failure, DB should be robust enough to get us the data, even the lost data and we should be able to restore it.

Installation - Software installation and configuration test is performed to check the software developed is getting installed successfully to the system. Each steps of installation instruction should be described and there should not be any difficulty or issue to install the software.

Rollback - Rollback test should be performed once any new deployment is done and after deployment application is not working as expected.

Failover - Failover testing verifies that application proceeds as normal when a redundant component fails.

What Have We Learnt So Far ?

1. Which Testing is performed first?
   - **A) Black box testing**
   - **B) White box testing**
   - **C) Dynamic testing**
   - **D) Regression testing**

Answer: 1 : (b)

2. A tester is executing a test to evaluate and it complies with the user requirement for a certain field be populated by using a dropdown box containing a list of values, at that time tester is performing _____
   - **A) White-box Testing**
   - **B) Black-box Testing**
   - **C) Load Testing**
   - **D) Regression Testing**

Answer: 2 : (b)

3. ORT comes after integration testing - Statement is
   - **A) True**
   - **B) False**

Answer: 3 :(A)

1.4.2 Automation Testing

Automation Testing

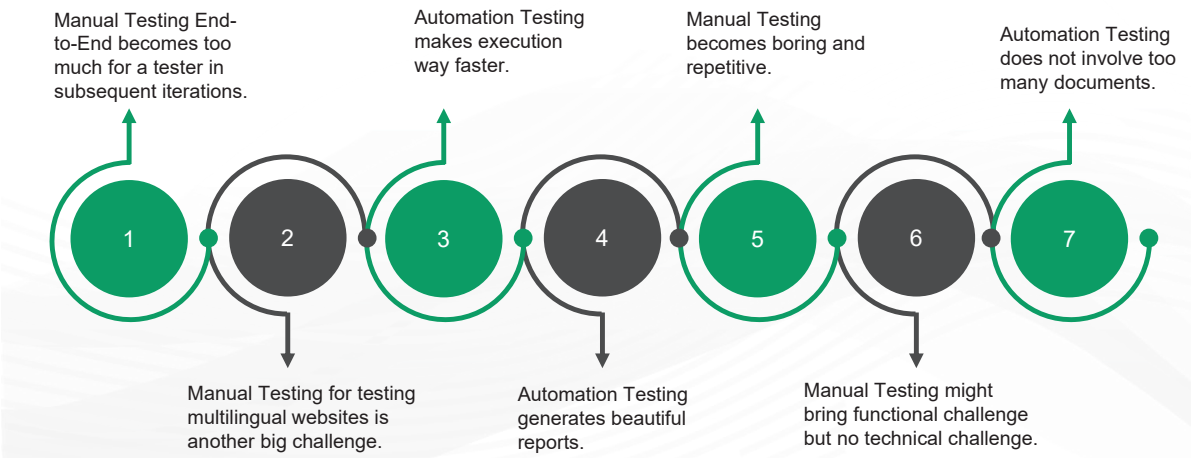
Automation testing is a Software testing technique to test and compare the actual outcome with the expected outcome. This can be achieved by writing test scripts or using any automation testing tool. Test automation is used to automate repetitive tasks and other testing tasks which are difficult to perform manually.

Some points about automation testing:

- It is performed by automation testers.
- It is done with usage of a test execution tool.
- Application needs to be stable for automation testing.
- It is very cost effective at regression and retesting stages.
- Automation Testing requires skilled workforce.
- Initial costs of setting up automation testing are high.

1.4.2 Automation Testing

Why do Automation Testing ?



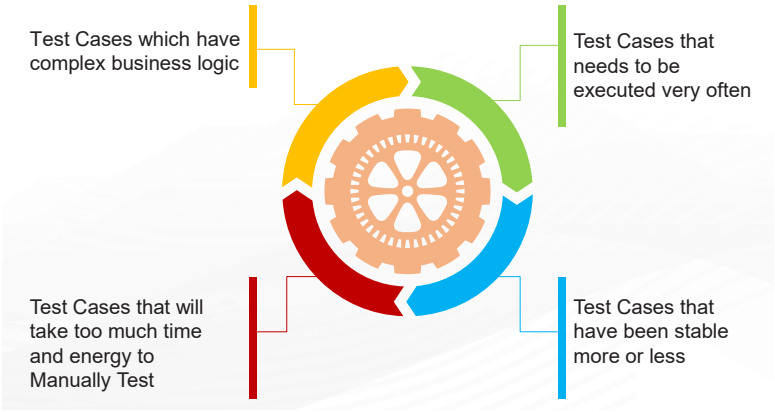
A human mind thinks and works differently. Imagine you eating the same food every day. The 1st day you would find it tasty and you can continue eating the same for 3 or 4 days. 5th day onwards you will feel the need of change and after 10 days you will start hating the same food even though it is as tasty as it was on the 1st day. That is just human nature. Change is permanent and in terms of software testing too, this logic is applicable.

A manual tester loses the zeal, anxiousness and energy to test the same application in subsequent regression builds. Hence, the efficiency goes down and defects creep in. Manual Testing can get monotonous and boring. Here, Automation Testing comes to rescue. What you are bored of doing Manually – Automate It! As simple as that.

It saves time and money and gets nice reports and there are some new challenges to face but all these are secondary. What matters most is the efficiency of Testing. If Automation Testing can give us better and faster results than Manual testing then we should definitely implement it.

1.4.2 Automation Testing

What to Automate ?



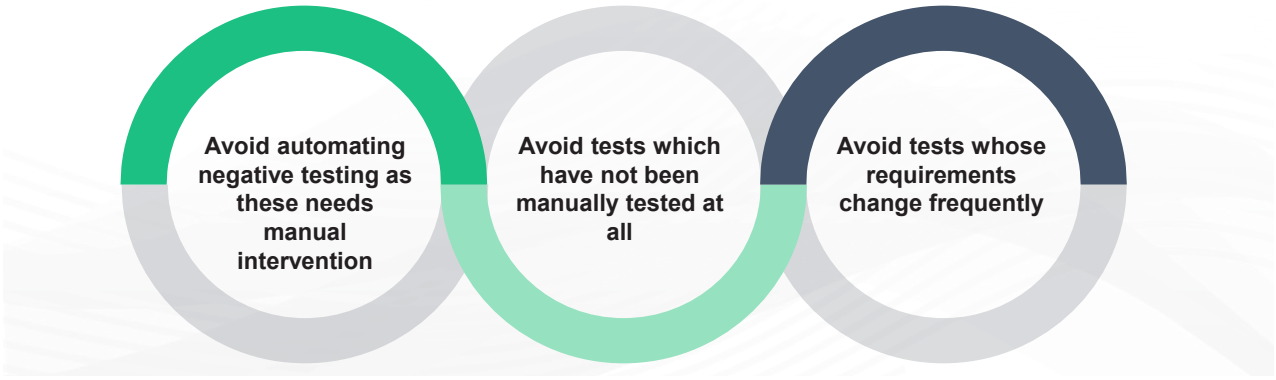
Let's say there is a Test Case Template and there are certain test cases which you have marked as complex as these involve some complex business logic which have dependencies on multiple modules. Though these are stable but manually takes a lot of time to test. So automate these areas and run them in Automation Test Suite.

There would be certain test cases which need to be executed every time there is a new release, irrespective of how many changes have been implemented in this new release but they have to be, and sometimes, testing the same thing again and again can get boring and also manual testers tend to skip which is dangerous. So automate these areas.

Test Cases that have been stable since the beginning or in last few iterations, automate these areas too.

1.4.2 Automation Testing

What not to Automate ?



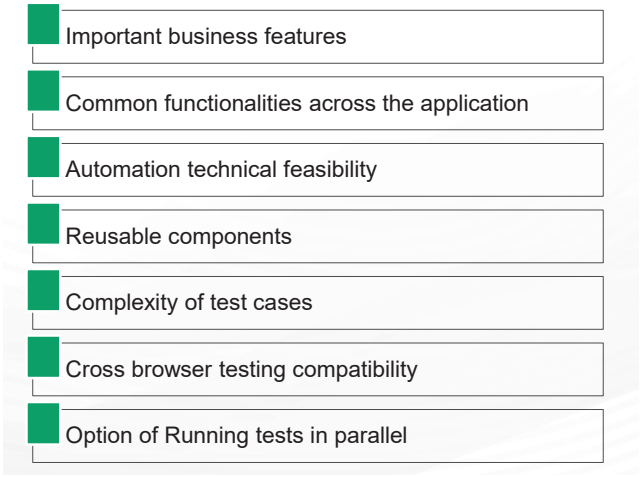
Negative testing will have multiple inputs and multiple out of the box scenarios which may be valid and may not be valid. So automating all of them might not be possible. These are better off traded with a manual approach.

Test cases which have not been tested manually before should not be automated as Manual Testing is the stepping stone. Automation Testing is more like making our life easier and helping us to increase test coverage as it will focus on areas which takes iterative approach and more time.

Avoid test cases whose requirements are changing very frequently. It can be done but then again it will have to be tested manually first then necessary changes have to be implemented in the automation code and then execution will be done and then again it will consume more time.

1.4.2 Automation Testing

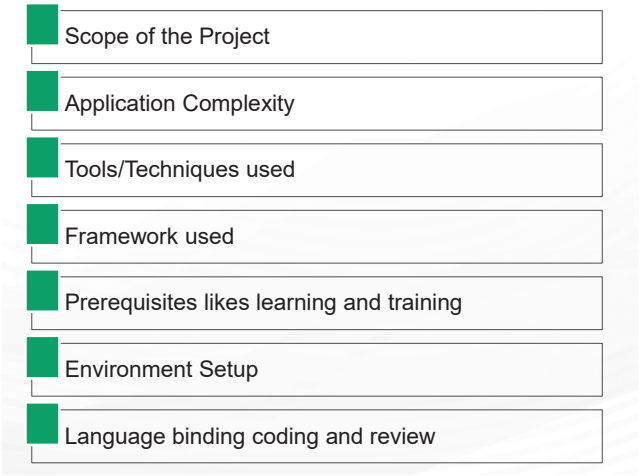
Scope of Automation:



In the course of discussion we have given a lot of emphasis on what makes the base of Automation Testing. So the Scope also is in close proximities with the same understanding. Everything cannot be automated risk free. Important Business features which needs constant vigil are the ones which should be automated. The ones which have commonalities across the application and doing so manually might be time consuming. So automating those will be quite fruitful. Manual testers cannot automate without having the functional knowledge of the application and neither can they without having the technical know-how about what approach are they going to follow to automate the same. There has to be certain features which will be common across, lots of reusability properties and other dependencies. Complexity plays a major role. Certain complexities become too complex to automate and certain complexities beg to automate. So understanding the complexities is also important.

There is a new thing in software projects these days – Browser compatibility testing and running tests in multiple browsers at the same instance and validating and comparing the performance outputs. So all these need Automation to pitch in as manually it will be difficult to carry out a large piece of work.

Estimation Automation [Selenium] :



Scope of the Project – Scoping is a big part of testing and identifying the right approach. Let's say you want to break the whole project into small modules, then how you do that by using work breakdown structure or functional point method or 3 point estimation method is totally dependent on the feasibility of how the application will behave. Identification of right resources is also important as manual testers cannot perform this operation.

Application Complexity – Again this is quite a debatable topic. Sometimes the webpage to be automated could be complex with lots of hidden elements and the interaction with those elements could be a challenge but this will not be revealed before actually automating it. So the smart move is to keep some buffer time in case such a thing happens.

Tools/Techniques Used – Selenium is not 100% a complete tool. It has a lot of 3rd party support for various purposes and combining all these different tools and applying different techniques to make sure that these tools are operating in proper sync as they are expected to, needs a thorough understanding of how much time it might take.

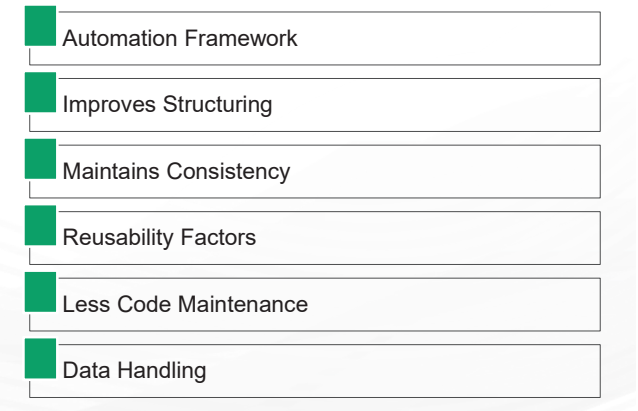
Framework used – In this particular project, we used Page Object Model and Data Driven. Let's say there is a project which is hybrid in nature. Then accordingly your test architect shall design the layout of the plan. So framework to be implemented – this gives varied timeframes for varied implementations.

Prerequisites like learning and training – Learning Selenium will not happen overnight. It will need a lot of practice and coding skills and other efforts. Hence, in case the automation project is purely on Selenium, then these things should be kept in mind if the manual testers are getting upgraded to Selenium Automation Testers.

Environment Setup – Some of the environment setups required may be

- Setting up the code in the test environment.
- Setting up code in production environment.
- Writing scripts for triggering the automated tests.
- Developing the logic for reporting.
- Establishing the frequency of running the scripts and developing logic for its implementation.
- Creating text/excel files for entering the test data and test cases.
- Creating property files for tracking the environments and credentials.

Language Binding Coding and Review – Any coding language needs time to get a proper hold upon. So rushing into Selenium without learning about Java or any other language binding which Selenium supports is like jumping into water without knowing how to swim. This can take time in case the testers have no prior knowledge. Hence, accordingly testing estimation needs to be made.



A framework design needs experience and expertise.

The most important aspect is to structure everything in the most compact form which is also easy to understand and re-usable in the future. It should have some level of consistency and should not give improper results with changes in data values.The framework should be designed in such a fashion that need not ask testers to keep changing the code base frequently. Minute changes would be enough to make it up and running.It should be compatible enough to handle small to large amount of data and work flawlessly.

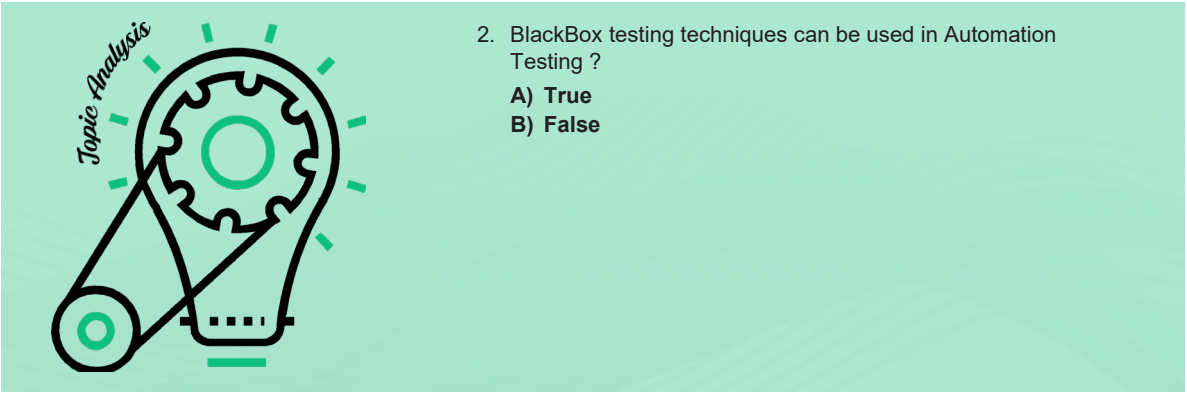
Some popular Automation Frameworks are :-

- Keyword Driven
- Data Driven
- Hybrid
- Page Object Model
- Cucumber BDD

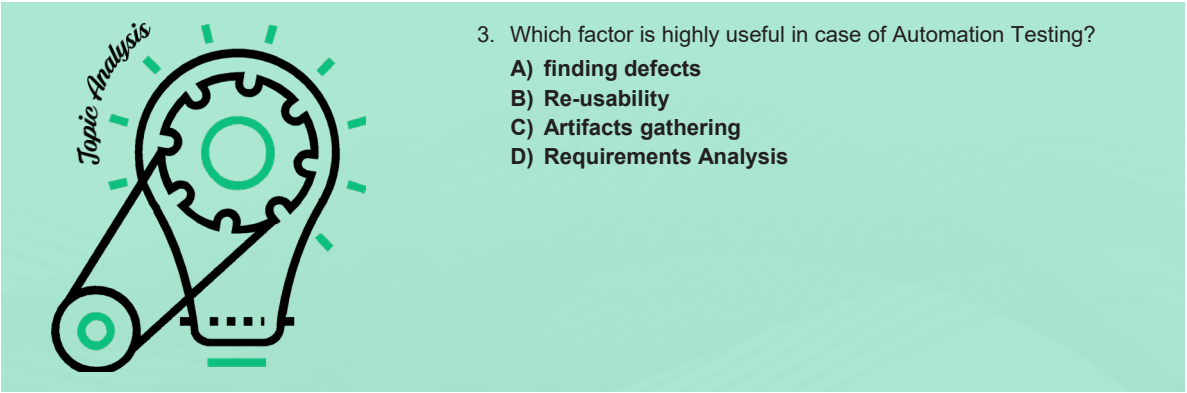
What did we learn so far ?

1. Automation Testing is used to mostly run
   - **A) Regression Test Suites**
   - **B) Smoke, Sanity, Regression Test Suites**
   - **C) Smoke, Sanity, Regression, Functional Test Suites**
   - **D) B and C**
   - **E) A and C**
   - **F) None of the above**

Answer: 1: A



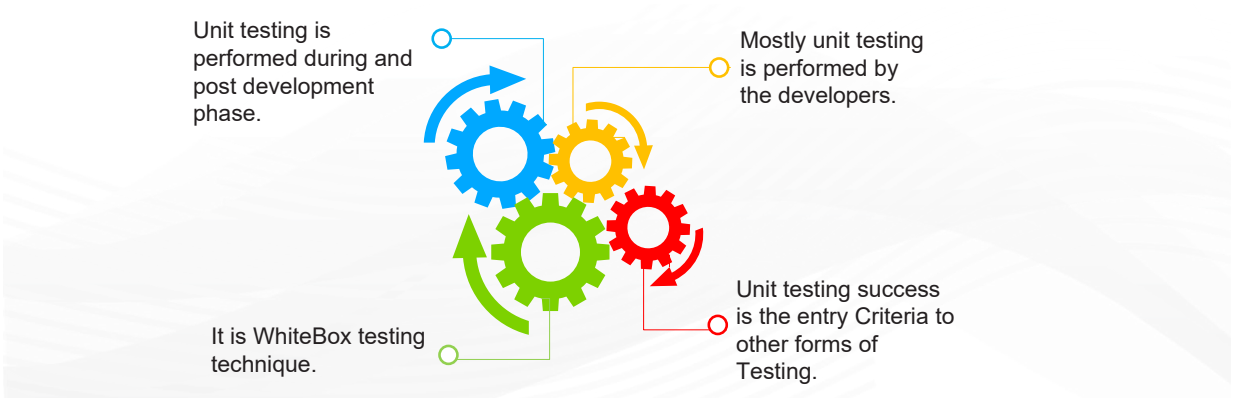
Answer: 2 : A



Answer: 3 : B

1.4.3 Unit Testing

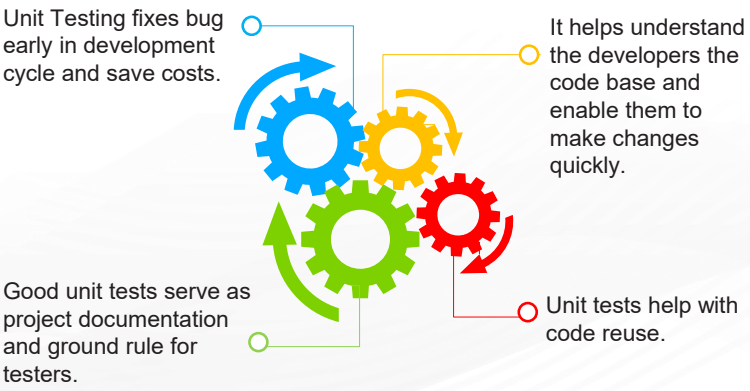
Unit Testing is also known as Component Testing as Individual Units/Components are tested during this phase.



Developers test but they test if their logic is working fine or not. They do not integrate other modules and test how their logic works when integrated with other modules. They will test a single unit or component to see if it works fine. This is called Unit Testing.

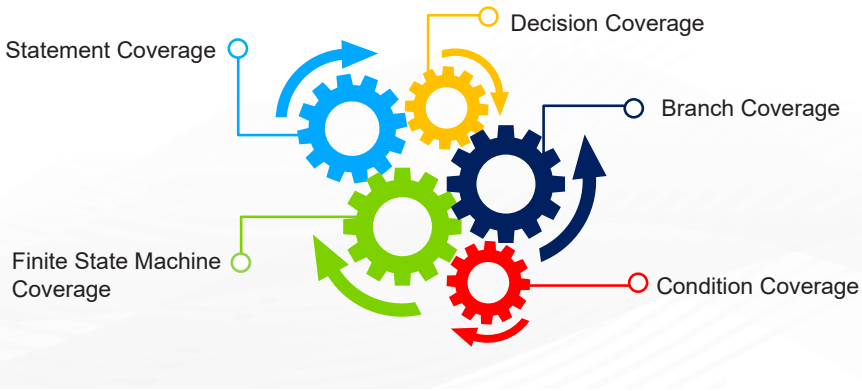
Unit Testing once pass, will let the application go into actual rounds of testing.

Why do Unit Testing



Developers are overloaded. In our industry for every 3 developers, there is 1 tester. Now this ratio might sound odd. In the first statement when I said “Developers are overloaded” but truth is development is completely a different activity and the resource : resource ratio does not matter. Since they are overburdened, they tend to skip unit testing at deep levels and have a mind-set that the testers will find the defect then it will be fixed. Hence, developers attempt to save time by doing minimal unit testing. But it backfires, because skipping on unit testing leads to higher defect fixing costs during System Testing, Integration Testing and even Beta Testing after the application is completed. Proper unit testing done during the development stage saves both time and money in the end.

Unit Testing Techniques are as follows:



Code coverage depicts the degree of which the source code of the program has been tested. The code which actually defines the business logic.

It is one form of white box testing which finds the areas of the program that may or may not have been executed by test cases[yes Unit Testing involves writing test cases.

Some form of test coverage also increases. Again, it purely depends on the developers who are really keen on this thing, they might not even do it.

Some test cases even give relevant information about the program under run. But the basic reasons why unit testing should be done in a nutshell is that it makes the code qualitative even though definition sounds quantitative.

1.4.3 Unit Testing Techniques

Statement Coverage

- Execution of all the executable statements in the source code at least once.
- Statement coverage is used to derive scenario based upon the structure of the code under test.

WhiteBox testing defines that a whitebox tester should concentrate on how the business logic of the software works. In other words, the tester will be concentrating on the internal working of source code. The source code of any software application will have classes, functions, loops, exceptions, print statements, exceptions, etc.Based on the input to the program, some of the code statements may not be executed.

The goal of Statement coverage is to cover all the possible lines and statements in the code.

Decision Coverage

- Aims to ensure that each one of the possible branch from each decision point is executed at least once and thereby ensuring that all reachable code is executed.
- It Reports true or false for each Boolean expression.
- 100% coverage is not possible for every decision.
- Combinations are more important as multiple scenarios have to be tested,
- Very Helpful in high critical scenarios

Let's say for example you are a Java Developer who is working on a large Project which has multiple branches. And for each branch you have created a package and each package has multiple classes and integration of all these branches completes the code. So as per this testing you need to test at least the most important business logic from each branch. Like test a particular package with certain class in it and execute and see how it responds.

What you can do is identify which class is extending which other class. What properties does it inherit and what it does not. So the code format decides what has to be undertaken and what can be skipped. This is called Decision Coverage.

1.4.3 Unit Testing

Branch Coverage

- Every item from code module is tested.
- Every possible branch from each decision condition is executed at least once.

Let's say you made a Java class which has approximately 10 test cases and you are using TestNG and of the 10 tests you have 6 important functionalities and the rest 4 are dependent methods. So what will you do as tester or let's use a word SDET [Software Development Engineer in Test] - you will test each and every @Test annotation. And there will be some code which you have written. When you run the code and Test annotation number 4 fails.

You will check the reasons of failure and do necessary changes in Test annotation number 4. Likewise, you will continue till all your 10 test cases are passing or they are finding deviations.

Similarly a developer has a lot of branches which might have lot of business logic which leads to a certain decision condition. So the developer tests all the business logic conditions at least once before proceeding. This is called Branch Coverage - to check all the business logic within each branch at least once

Condition Coverage

- Reveals how the variables or subexpressions in the conditional statement are evaluated.
- Expressions with logical operands are only considered.
- Offers better sensitivity to the control flow than decision coverage.
- Does not give assurance of full coverage.

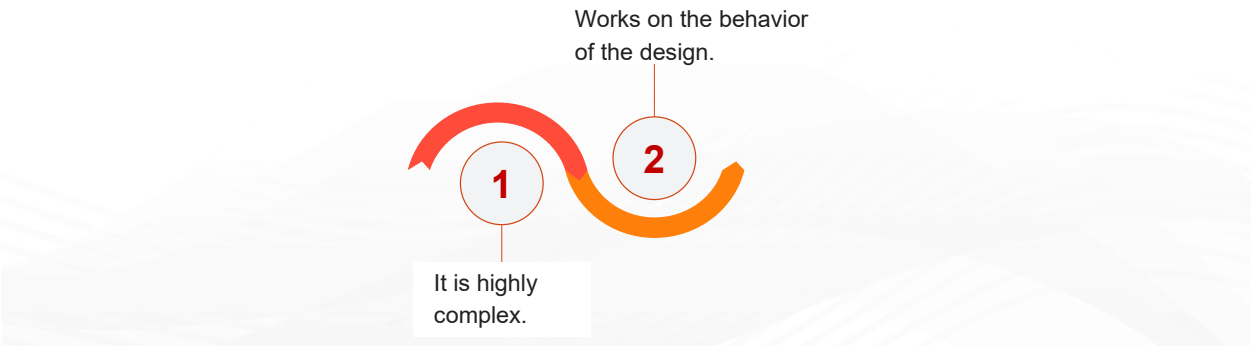
Condition coverage is also known as Predicate Coverage in which each one of the Boolean expression have been evaluated to both TRUE and FALSE.

All the testing outcomes are at least tested once. This type of testing involves 100% coverage of the code. The test cases are designed so that the condition outcomes are easily executed. This can be documented.

It takes certain amount of time but it is a practice that is skipped mostly as it involves in-depth knowledge. Getting inside functionality of each executable is not an easy task.

1.4.3 Unit Testing

Finite State Machine Coverage:



Finite state machine coverage is certainly the most complex type of code coverage method. This is because it works on the behavior of the design. In this coverage method, you need to look for how many time-specific states are visited, transited. It also checks how many sequences are included in a finite state machine.

Let's say you are working on the integration of payment division with a 3rd party software. Or say you have shifted from an old vendor to a new vendor. And the new vendor did not acknowledge old vendor's source code and complete new implementations have been done.

Hence, for this major transition it is but obvious that the system will behave differently with respect to this new vendor. In this case the compatibility testing also comes in picture. And the deviations which are recorded - are they smooth or too many errors and bottleneck issues

are bothering the operations. Every transition is recorded. At what time intervals all these transitions are taking place.

What have we learnt so far ?

1. While using white-box testing methods, the software engineer can derive test cases that
   i) guarantee that all independent paths with in a module have been exercised at least once.
   ii) exercise all logical decisions on their True and False sides.
   iii) execute all loops at their boundaries and within their operational bounds.

   A) i and ii only
   B) ii and iii only
   C) i and iii only
   D) All i, ii, and iii

Answer: 1 : D

2. ……, also called behavioral testing which focuses on the functional requirements of the software.

   A) White-box testing
   B) Control structure testing
   C) Black-box testing
   D) Gray-box testing

Answer: 2 : C

3. …….. tests are designed to validate functional requirements without regard to the internal working of program.

   A) White-box test
   B) Control structure test
   C) Black-box test
   D) Gray-box test

Answer: 3 : C

1.4.4 Integration Testing

Key points about Integration Testing:

- Individual units or components are combined and tested,
- Deviations are recorded when tested individually vs combined.
- Interface between the modules is the prime focus of test.

A single unit might have a logic which works fine when tested individually. The challenge comes when it is integrated with another unit and then complications creep in. Sometimes change in requirements leads to fast pace development work which might skip unit testing as meeting deliverable timelines becomes a priority. So these activities lead to certain issues which can be broadly divided to data formatting, problem with 3rd party software interference.

No matter how well each unit works, when it comes to integrating multiple modules, their response needs to be validated.

Advantages

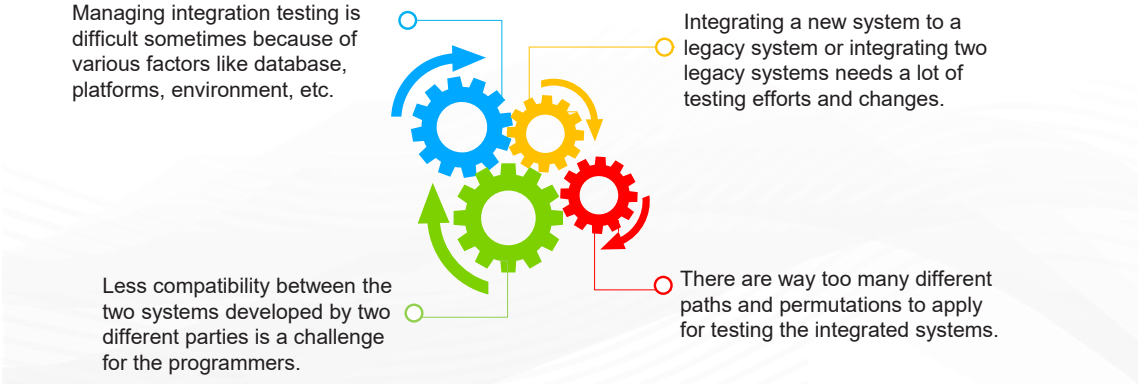
Advantages of Integration Testing

- Interfaces which are quite vulnerable undergo testing during this phase.
- Early detection of errors prevents a lot of re-work and last moment burden.
- The coverage of Integration Testing is large.
- Integration Testing increases the reliability of the system.

Imagine a worst case scenario, where Unit Testing has been totally negated and all Units are also vulnerable but we have to submit the deliverable and time is not our best friend. So we have to test individually each unit and then integrate them and test it. In case, we ignore the integration then imagine what happens when there is a total debacle at client's end.

Hence integration testing is very important as it helps to achieve a lot of coverage with lot of reliability and makes the whole system more efficient and also helps in determining errors not only at surface level but also in-depth.

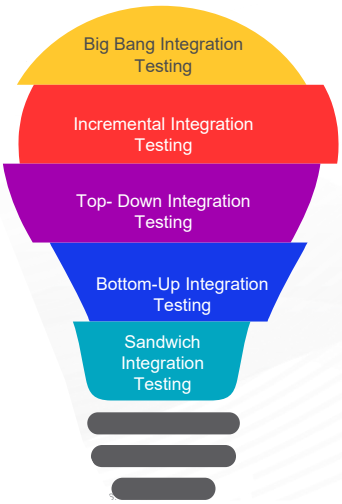
Challenges in Integration Testing



 |

Challenges are everywhere but does not mean we do not implement the activity.

The various types of Integration Testing are:

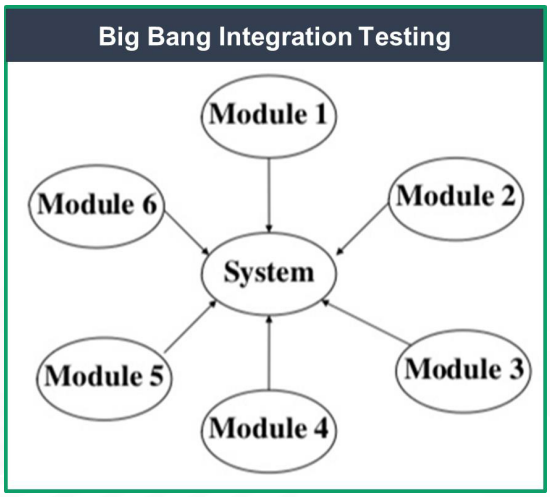


Stub : is a piece of code used to stand in for some other programming functionality.

Drivers : dummy code, which is used when the sub modules are ready but the main module is still not ready.

Big Bang Integration Testing

During Big Bang integration testing all the units are linked at once, resulting in a complete system.



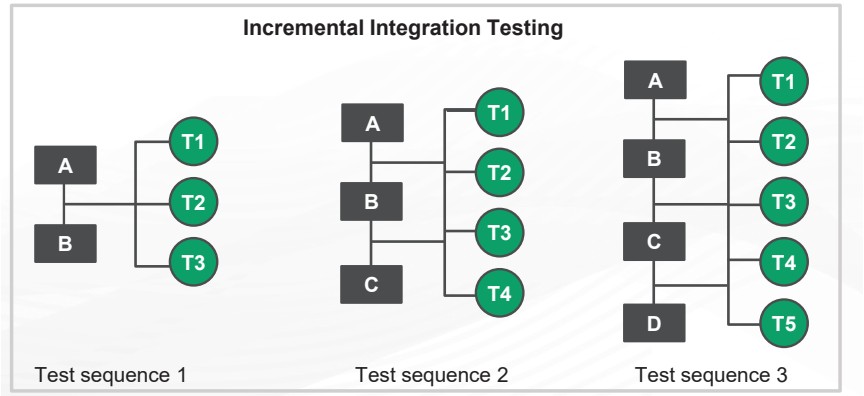
What happens here is all the modules [in the image 1, 2, ,3, 4, 5 and 6] are developed individually. Then they are tested individually

Next step is they are integrated together. All of them and they are tested. Sounds like madness. Isn't it ? But this is effective for small systems.

So this is not highly recommended because of the flaws it comes with:-

- Identifying a defect is difficult even though it is present.
- A lot of delay before testing as image each module will be developed, then it will enter testing

Incremental Integration Testing: Incremental Testing is performed by connecting two or more modules together that are logically related.

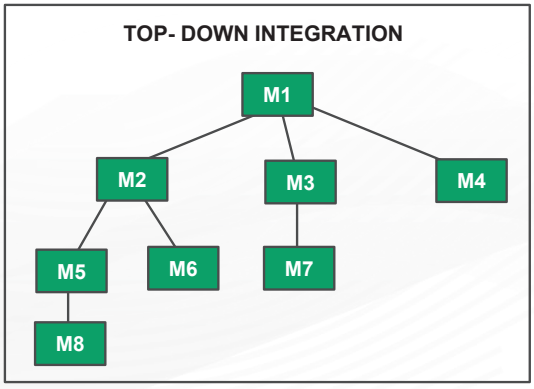


Incremental Testing is performed by connecting two or more modules together that are logically related. In the figure in Test Sequence 1, A and B are logically related.

Later more modules are added and tested for proper functionality. In Test Sequence 2, A and B and C are logically related.

This is done until all the modules are integrated and tested successfully.

Top- Down Integration Testing: The top-down approach starts by testing the top-most modules and gradually moving down to the lowest set of modules one-by-one



- The top-down approach starts by testing the top-most modules and gradually moving down to the lowest set of modules one-by-one.
- Testing takes place from top to down following the control flow of the software system.
- Stubs are widely used as there could be ambiguity in levels of modules being tested and developed.

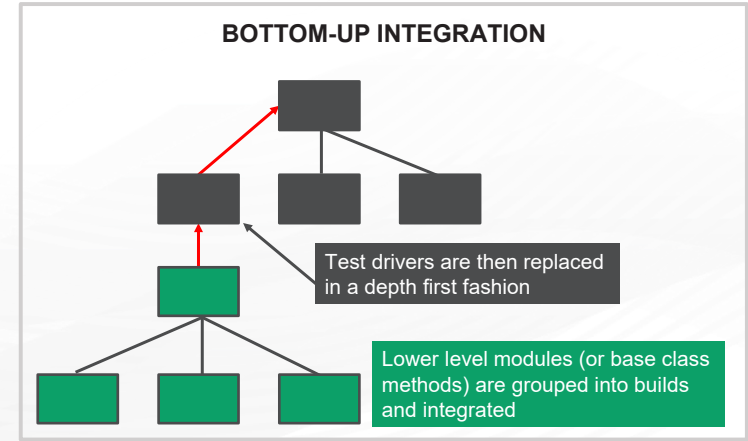
Advantages :

- Can easily identify the fault.
- The system is consistent and efficient.
- The stubs can be written in lesser time compared to drivers.
- Critical modules are tested on priority.
- Major design flaws are detected as early as possible.

Disadvantages :

- Requires several stubs .
- Poor support for early release.
- Basic functionality is tested at the end of the cycle.

Bottom-Up Integration Testing: The bottom-up approach starts with testing the lowest units of the application and gradually moving up one-by-one.

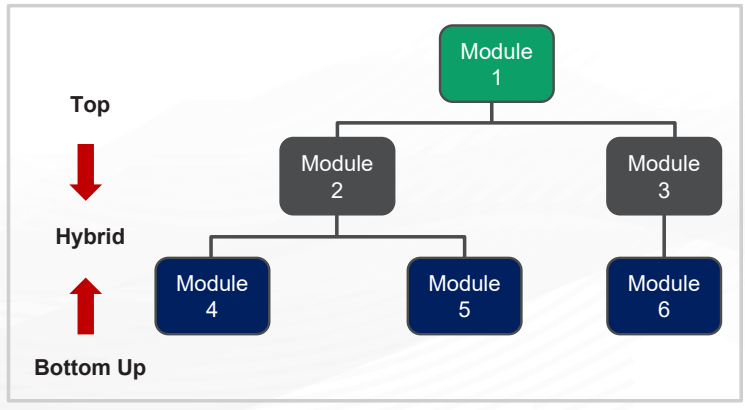


- The bottom-up approach starts with testing the lowest units of the application and gradually moving up one-by-one.
- Testing takes place from the bottom of the control flow to upwards.
- If higher level modules might not have been developed by the time lower modules are tested. Simulation is done - The functionality of missing modules by using drivers.
- These drivers perform a range of tasks such as invoking module under test, pass test data or receive output data.

Advantages:-

- **Disadvantages:** Here development & testing can be done together so the product will be efficient.
- Test conditions are much easy to create.
- Requires several drivers.
- Data flow is tested very late .
- Need for drivers complicates test data management.
- Poor support for early release.
- Key interfaces defects are detected late.

Sandwich Integration Testing: The Top-Down and Bottom-Up testing techniques can be performed in parallel or one after the other.



- Also known as mixed integration testing or Hybrid integration testing.
- There are 3 layers - Main target layer in the middle, another layer above the target layer, and the last layer below the target layer.
- The top-down approach is used on the layer from the top to the middle layer.
- The bottom-up approach is used on the layer from the bottom to middle. Big bang approach is used for modules in the middle.

Advantages

- Top-Down and Bottom-Up testing techniques can be performed in parallel or one after the other
- Very useful for large enterprises and huge projects that further have several subprojects

Disadvantages

- The cost requirement is very high
- Cannot be used for smaller systems with huge interdependence between the modules
- Different skill sets are required for testers at different levels

1.4.5 Smoke-Sanity Testing

Smoke Testing

- ⇥ Is initial health check up of software builds to check eligibility to move to next round of testing.
- ⇥ It is generally performed on critical functionalities.
- ⇥ Smokes tests can be both manual and automated.
- ⇥ It works as entry criteria to Sanity Testing.

- Smoke Testing is not exhaustive in nature. It targets a certain set of Tests. It does not involve a lot of tests. Certain tests which define the entry criteria for next levels
- Smoke Testing is a group of tests that are executed to verify if the basic functionalities of that particular build are working fine or not. In case, they are working fine then.
- it certifies the build to be moved to the next rounds of testing and in case it fails - it is reverted and the development teams fix it.

- It is the first thing that needs to be done as soon as build is handed over by the development team.
- Development team gets to immediately work on this as Build rejection criteria is determined by the Smoke Test Cases. Hence, no time is lost trying to follow long procedures.
- Carried out via a set of dedicated smoke test cases which can be manual or automated in nature.

Sanity Testing

- ⇥ Sanity Testing is done to check the new functionality/bugs have been fixed.
- ⇥ Sanity testing is usually performed by testers.
- ⇥ Does not have a dedicated set of test cases.
- ⇥ It is not surface level like Smoke Testing

Sanity Testing as a test execution which is done to touch each implementation and its impact but not thoroughly or in-depth, it may include functional, UI, version, etc. testing depending on the implementation and its impact.

If sanity test fails, the build is rejected to save the time and costs involved in a more rigorous testing.

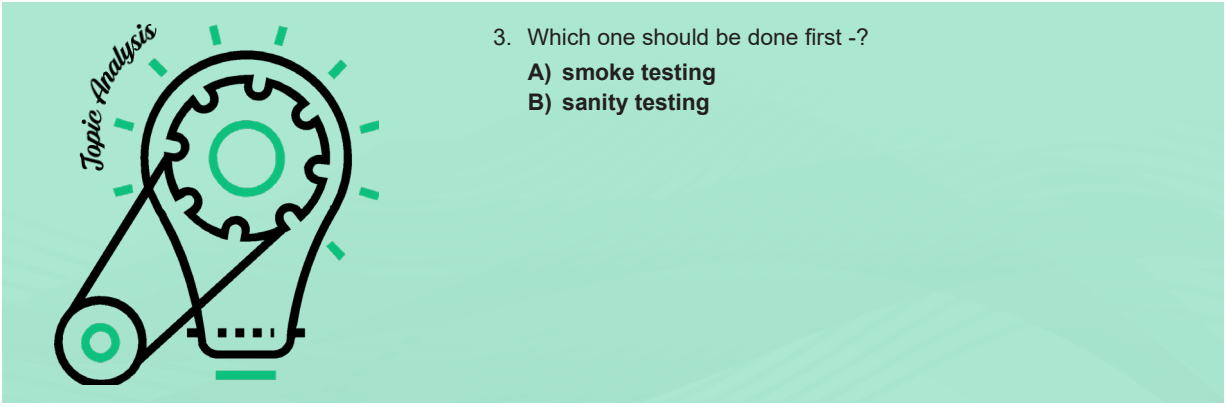
What have we learnt so far ?

1. Smoke Testing is the job of Developers only
   A) True
   B) False

Answer: 1: (b)

2. Which is a Build Verification Testing -
   A) smoke
   B) sanity

Answer: 2 : (a)



Answer: 3 : (a)

In a nutshell, we learnt:

1. Manual Testing.
2. Automation Testing.
3. Unit Testing.
4. Integration Testing.
5. Smoke-Sanity Testing.

Now, you have reached the end of the module, In this module, you have learned:

- Manual Testing.
- Automation Testing.
- Unit Testing.
- Integration Testing.
- Smoke-Sanity Testing.

 |

Release Notes

B. TECH CSE with Specialization in DevOps

Semester Six -Year 03

Release Components.
Facilitator Guide, Facilitator Course Presentations, Student Guide, Mock exams and relevant lab guide.

Current Release Version.
1.0.0

Current Release Date.
19 Jan 2020

Course Description.
Xebia, has been recognized as a leader in DevOps by Gartner and Forrester and this course is created by Xebia to equip students with set of practices, methodologies and tools that emphasizes the collaboration and communication of both software developers and other information-technology (IT) professionals while automating the process of software delivery and infrastructure changes.

The language used in this course is US English. Our sources of reference for grammar, syntax, and mechanics are from The Chicago Manual of Style, The American Heritage Dictionary, and the Microsoft Manual of Style for Technical Publications.

Bugs reported	Not applicable for version 1.0.0
Next planned release	Version 2.0.0 Jan 2021