UPES

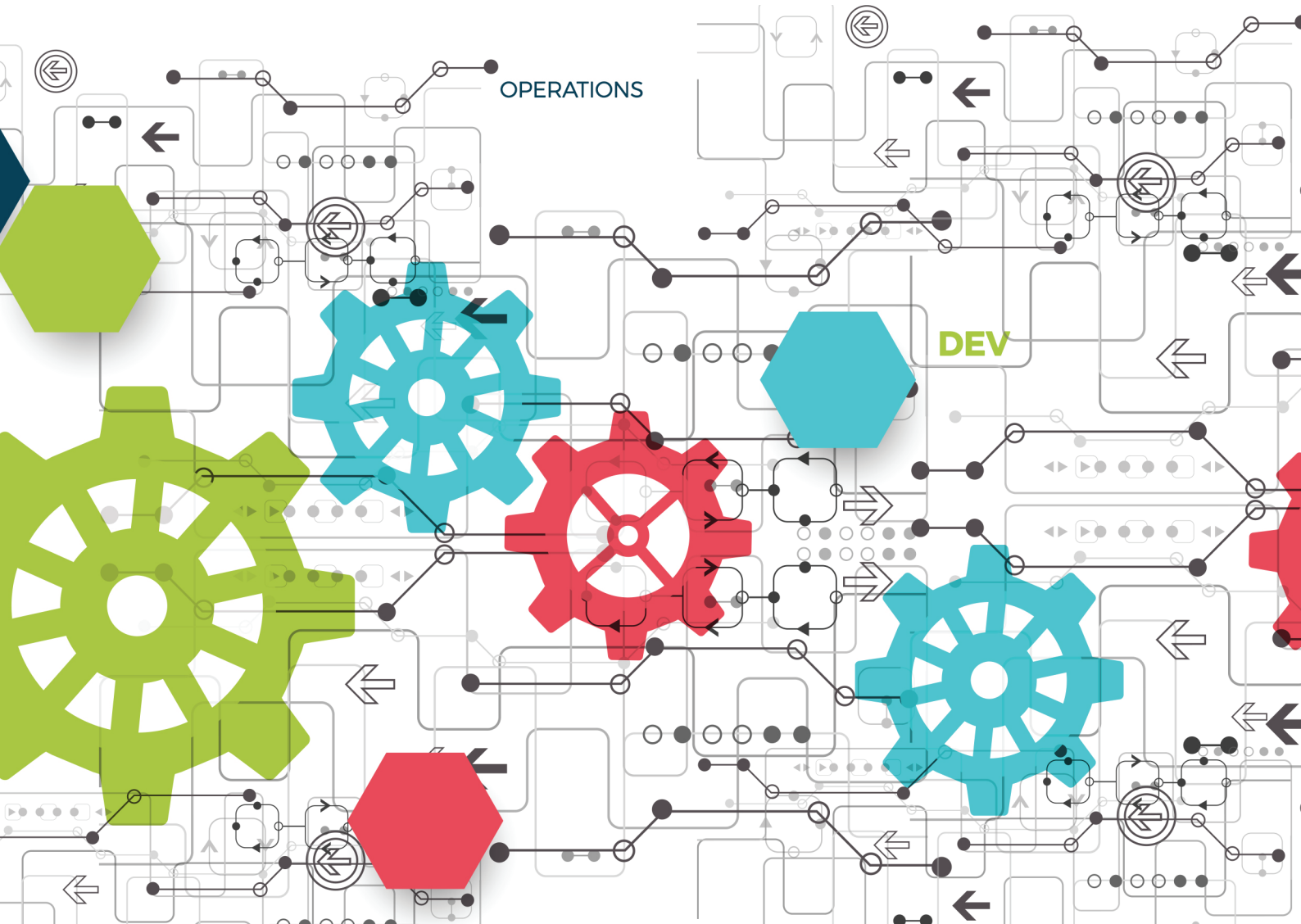
Xebia

YEAR 3

B.Tech Computer Science and Engineering in DevOps

Test Automation

MODULE 1
Introduction to Software Testing



Contents

MODULE 1

Introduction to Software Testing

You will learn about the ‘Software Testing’, best practices and Automation Testing

Module Objectives

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

- Define seven principles of Software testing
- Differentiate between SDLC v/s STLC
- Explain software testing life cycle
- Describe various types testing

You will be informed about the module objectives.

- At the end of this module, you will be able to learn:
- Define seven principles of Software testing
- Differentiate between SDLC v/s STLC
- Explain software testing life cycle
- Describe usability testing
- Describe functional testing
- Describe end to end testing
- Describe compatibility testing
- Explain GUI testing
- Explain API testing

Module Topics

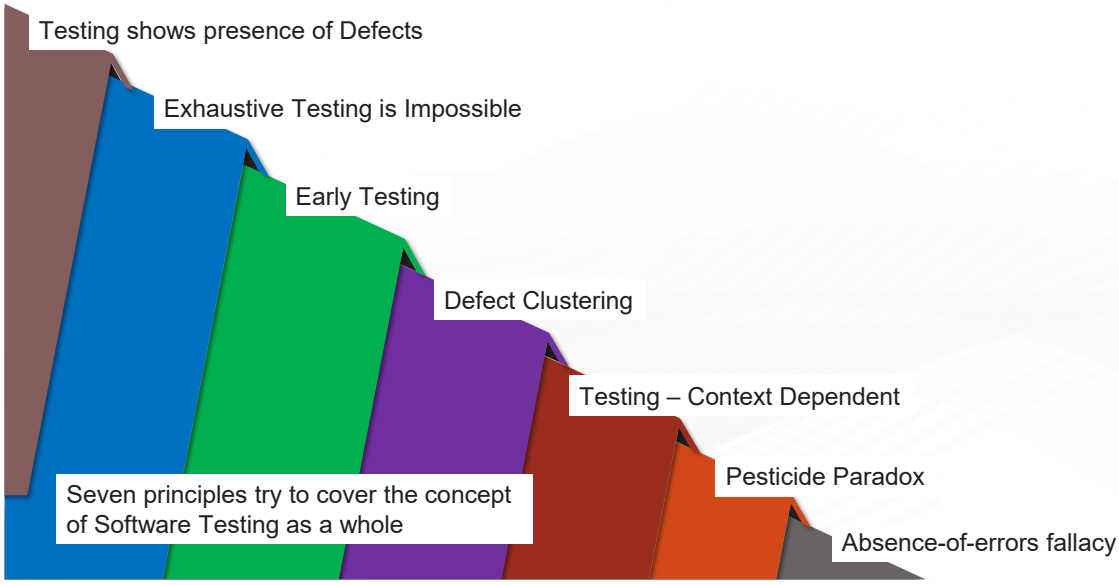
Let us take a quick look at the topics that we will cover in this module:

- Seven principles of Software Testing
- SDLC vs STLC
- Testing Life Cycle
- Usability Testing
- Functional Testing
- End to End Testing
- Compatibility Testing
- GUI testing
- API testing

You will learn about the following topics in this module:

- Seven principles of Software Testing
- SDLC vs STLC
- Testing Life Cycle
- Usability Testing
- Functional Testing
- End to End Testing
- Compatibility Testing
- GUI testing
- API testing

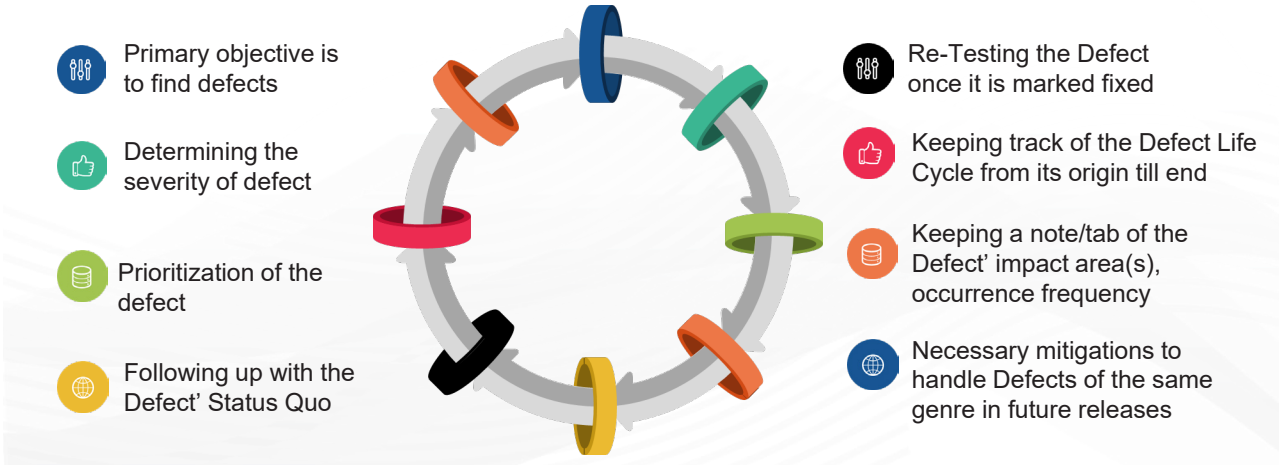
1.1 Seven principles of Software Testing



Software Testing as any other technology has evolved quite a lot. And it is growing at an exponential rate. But in a nutshell these 7 pointers define the core principles of Software Testing. Let's learn about them in detail.

1.1.1 Testing shows presence of Defects

Here are some of the reasons why testing plays important role in development life-cycle:



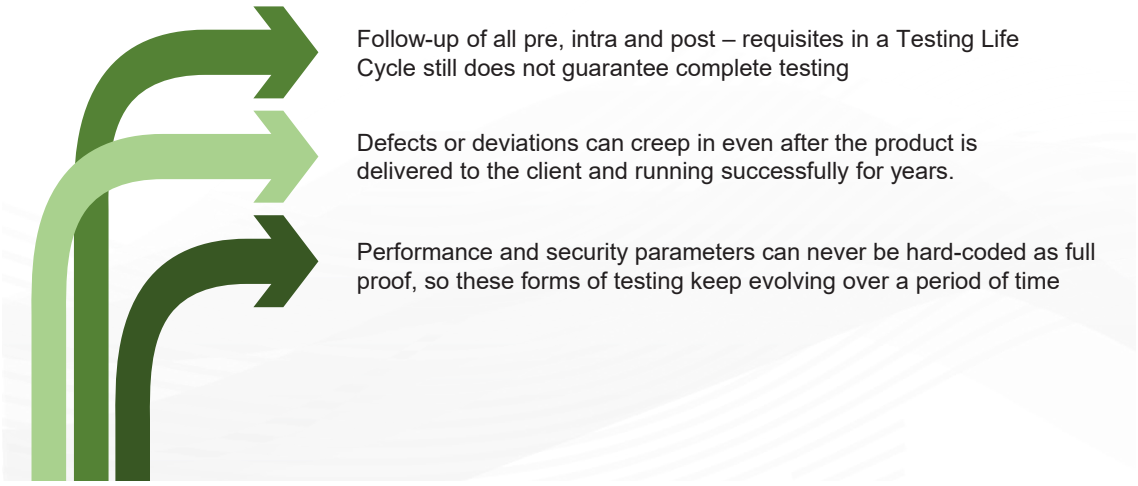
If we actually go with the basic definition of testing it says "*Operation carried out to find Defects*" . However, a more mature definition would be "*Operation carried out to find Defects along with their impact severity*" . But the modern definition of testing is "*It is a qualitative approach to determine the deviation of a system's behavior from its intended behavior resulting in formulation of quantitative data describing the severity of the impact and mitigations needed to resolute the impediments*".

So testing is way beyond just finding defects now. Gone are the days when testers were considered as Defect Finders. Software Testing in more reform terms is called Quality Control when it is related to the Product under test. And it is called Quality Assurance when it is

related to the complete Process as whole. So testing is finding defects but there is lot to testing that meets the general norm.

1.1.2 Exhaustive Testing is Impossible

Listed below are some reasons why test coverage can be 100% but testing cannot be 100%:



Exhaustive Testing as the name suggests is like trying to do maximum coverage of testing but 100% testing is not possible. There can always be a scenario which might exist and might not have been even comprehended. No tester can ever say “Testing stands 100% complete”. There is always room for improvement in testing and scope of testing. A certain testing approach might fail to identify any defect or deviation in behavior as intended but a certain other testing approach might get to do that. So comprehending various approaches for one particular event is quite a standard practice these days. In-spite of trying everything, sometimes still there are certain deviations which skip even the most knowledgeable minds. So complete testing is not possible.

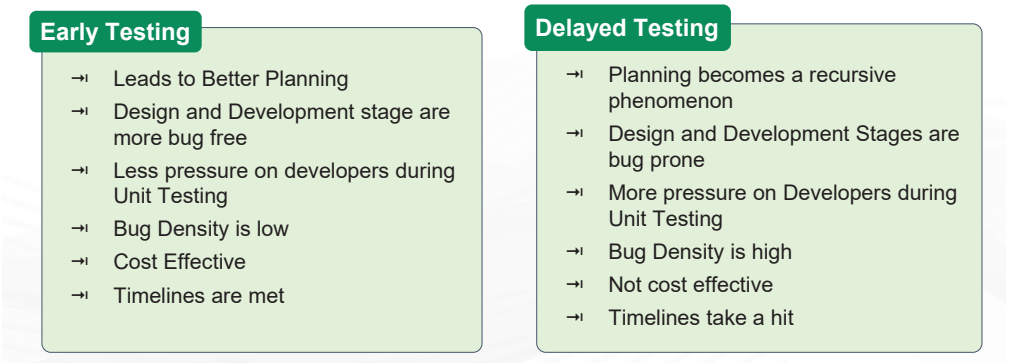
In today’s world of Continuous Integration / Continuous Development with shorter time frames and vivacious rebuilds and version changes, the probability of finding defects just has increased. The applications are more robust and more secure and defect free but softwares have different life cycles these days where in they carry a baggage of certain defects and deviations to the next iteration and that is quite a norm these days.

New Testing techniques have evolved, testing artifacts have evolved, testing tools which cover the aspects of Automation, Performance, Security have evolved but still there is always some gap. However small there always is. Might not be even visible to the most trained and experienced eye but still there is.

Software Testers, Testing Leads, Testing Managers, Testing Architects do their level best to get the software completely bug free but it is never 100%.

1.1.3 Early Testing

Reasons why early testing is much better than delayed testing



A decade and half or probably 2 decades ago, Testing Methodologies were not that advanced, and testing was considered a process which shall follow suit as development comes to an end but those processes failed and failed handsomely as there was too much re-work, application became cumbersome, too many iterations without proper testing carried recurring defects which kind of became too stubborn and affected the application's operation as a whole. Sometimes, the application's tell of the tale was so complicated to resolve, buggy applications were given a green from the production team and it resulted in serious failures, violation of contracts and preposterous business declines. Hence testing methodologies were deduced by industry experts which made testing to begin in tandem with development. The earlier you test, the secure you are. Simple.

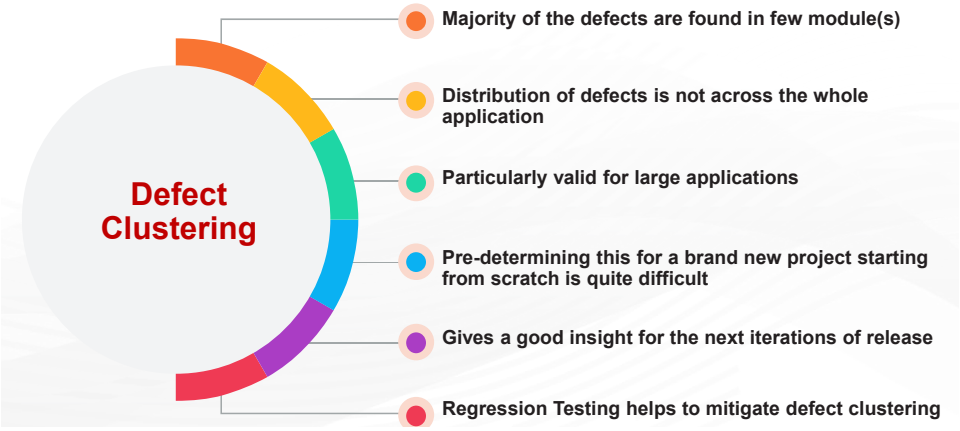
Software industry even up to mid 1990's was following the trend of one task complete, then doors shall open for the subsequent task. In simple terms, once requirement analysis got over, then only High Level Architectural Design was formulated. Once High-Level Design module got over, it opened doors for Low Level Design and subsequently Unit Testing, Integration Testing, System Testing,User Acceptance Testing, Operational Readiness Testing and so on.

The above scenario is concrete but it is concrete if every stage which precedes the other is completely bug free and non-ambiguous which we all know is impossible. For simple and basic application which did not involve too much Design and Coding, it worked fine. But when applications got larger, implementations got complicated and required extensive intelligent logic - then the above approach failed and it failed astoundingly. Statistically speaking software Industry suffers losses worth billions of dollars for such failures. The re-work as too much.

In 2020, even when testing was done from the very beginning of a product life cycle, still there were certain loophole which crept in one form or the other but it can be contained in a more organized manner. The re-work will be less, the cost incurred will be less and timelines can be still met if planning for buffer is good. So early testing will allow the fixation to happen within the same stage. And the defect will not be carried forwarded to the next stage.

1.1.4 Defect Clustering

Defect Clustering follows an extended idea of the Law of the Vital few which states that, for many events, roughly 80% of the effects come from 20% of the causes.

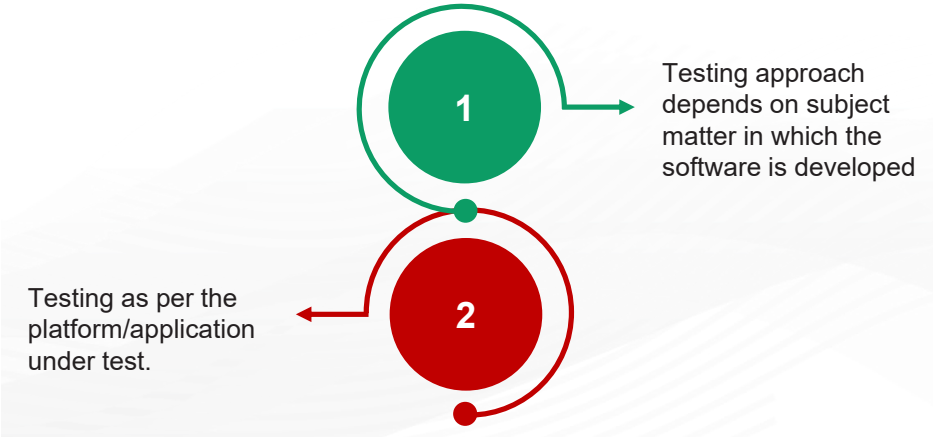


Defect Clustering can be true for applications of gargantuan proportions. With so many inter-dependent modules in any application these days, it is virtually impossible to say which particular module will have these many defects and when the next version comes out, that count will increase to certain number. So defect clustering at a certain module and its occurrence to be a predetermined is bit tricky.

Of course, there is a historical data but it is historical that means such an occurrence has been a thing of the past, so development and testing team will be more careful while working on that module or any upgrade of that module. But determining something like this for a Project which is brand new and beginning from scratch is very difficult.It shall however, give an insight to focus on areas with more precision and attention in the iterative releases.

1.1.5 Testing - Context Dependent

Testing is better if the tester understands the context:



Let's say for example you are an Automobile Engineer and all you know is about Engines, Gearboxes, Suspension Setup, Braking System, Air Conditioning and out of certain constraints in your engineering college you get a job in software industry and you land up becoming a Test Engineer.The project assigned to you is related to automobiles. Just imagine the speed

up and enthusiasm which will creep into you - the point being if you know something or may be know quite a bit about something or maybe you are an SME (Subject Matter Expert), then Testing around the knowledge becomes an easier and more enjoyable affair.

Similarly in Software Testing also, there are different domains to be tested, different testing approaches which are followed and different techniques related to those domains. To put it more simplistically, an E-Commerce application cannot be tested like a Flight Simulation application, or may be a Health Domain application cannot be treated like a Retail Market application.

The knowledge of the Context is very important. Learning the Functionality of the context and all scenarios to test might take weeks or in some cases even months.

So testing is Context Dependent. Certain set of skills to master that domain becomes imperative eventually with passage of time.

1.1.6 Pesticide Paradox

Pesticide Paradox can be defined as Software undergoing the same repetitive tests eventually builds up resistance to them :

Let's understand this with a lifetime example. *"You are a gym goer and you go to gym and do the same weight training for the same muscle groups over and over again and neither do you change the weights nor change the repetitions nor change any workout pattern. So what happens ?"*

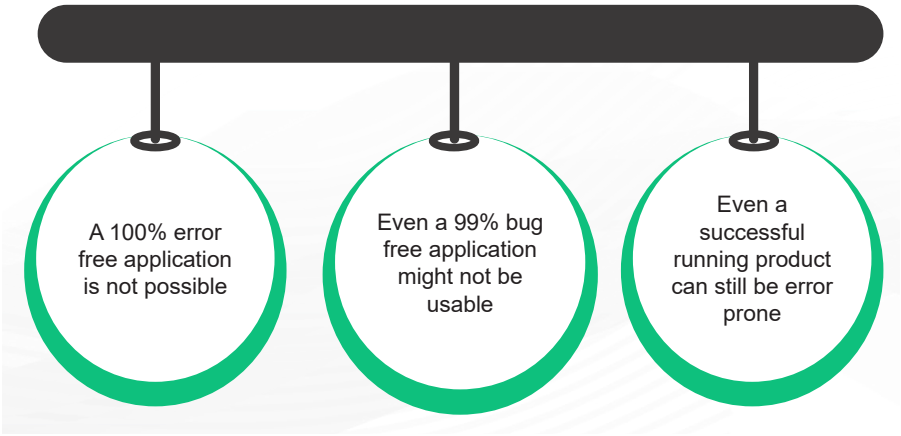
There is no muscle growth, there is no further change in your body's strength, stamina, muscle outlook. So technically your body has become immune to the same old fashioned routine. Similarly if any application under test undergoes the same kind of testing over and over again then finding defects will become redundant after some point of time. So a tester should keep evolving and use different testing techniques and try to break the system with an intent to find defect(s) or any deviation in behavior.

Let's say you have an application which has evolved over a period of time and initial releases there were lot of bugs and testers found those bugs and reported those bugs which got fixed and now with multiple releases of the same application, the Development team has done a commendable job to ensure the defect density is too low and on a daily test sanity run, no defectis encountered. So does this mean the application is totally error free ?

Answer is NO. Need to change the approach of testing. May be add more negative test cases, purposefully try to break the application using lot of negative scenarios. Get more understanding about the application by interaction with the architects and finding areas of vulnerability which are stable with current Test cases but they might break with introduction of something new.

We can also implement these changes in the Requirement Traceability Matrix, introduce a TCER [Test Conditions and Expected Results] sheet dedicated only to negative testing, for each and every functionality that can render a negative outcome – prepare some test scenarios Test the Security and Performance aspects of the application too.

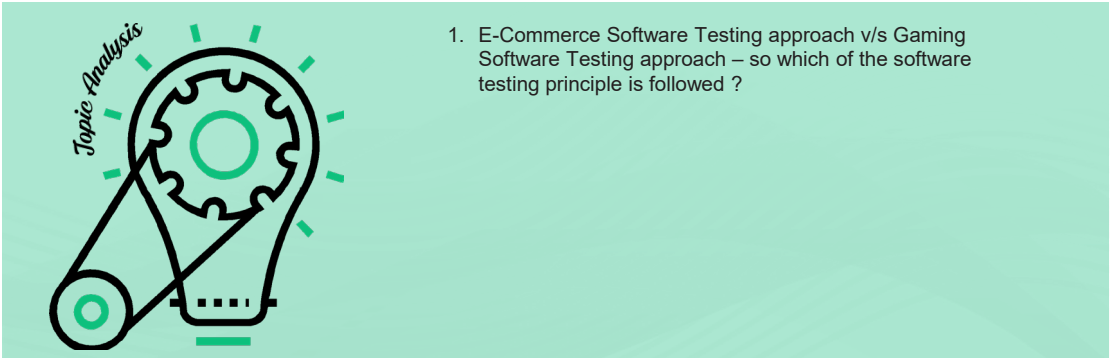
1.1.7 Absence of Errors - Fallacy



Absence-of-errors fallacy – Ok let's learn this with an example again. Let's say there is an application which has been released in the market and the market has accepted the same and millions are coming. Does this guarantee that the application is totally bug free ?? Answer is : NO and that too an astounding NO because nothing is 100% bug proof.

Some scenario – let it be security, performance or functional can come up which will induce or find a certain deviation in the application. So 100% error free application is next to impossible.

What did you grasp?



Answers: 1 : Context Dependent

2 Which of the following statement is true?

A) Exhaustive testing is impossible but practicable
B) Exhaustive testing is possible but impracticable
C) Exhaustive testing is impossible and impracticable
D) Exhaustive testing is possible and practicable

Answers: 2 : (b)

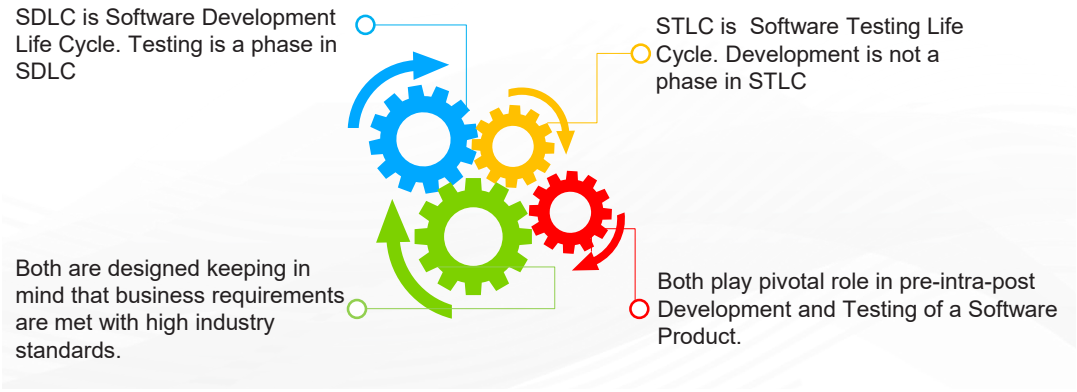
3 Software tester focuses more on complex part of the software. Which of the following testing principle implies this?

A) Testing shows presence of defects
B) Pesticide paradox
C) Testing is context dependent
D) Defect Clustering

Answers: Ans 3 : (d)

1.2 SDLC vs STLC

SDLC is different from STLC as STLC focuses completely on testing:



Let's understand word by word.

S - Software Application.

D - Development (as the word suggests a certain software following a certain Architectural Design is being developed)

L - Life (starting from day 1 till its closure date)

C - Cycle (the software can have multiple iterations, versions, updates and total end to end responses)

So what does SDLC do or if we ask the right question why is it even deduced. In simple terms, it is kind of a process to consolidate all the necessary phases and understand the individuality of each phase and integrate them then see how they work together. And then prepare the best systematic way to produce a high quality product keeping in mind the end user' requirements.

Reviews/Inspections/Walkthroughs are important practices before and after completion of every stage that provides optimum management control.SDLC plays a pivotal role in giving structured, editable. reusable documentation with different Version controls.Exit and Entry Criteria for each Stage is well defined.

S - Software Application

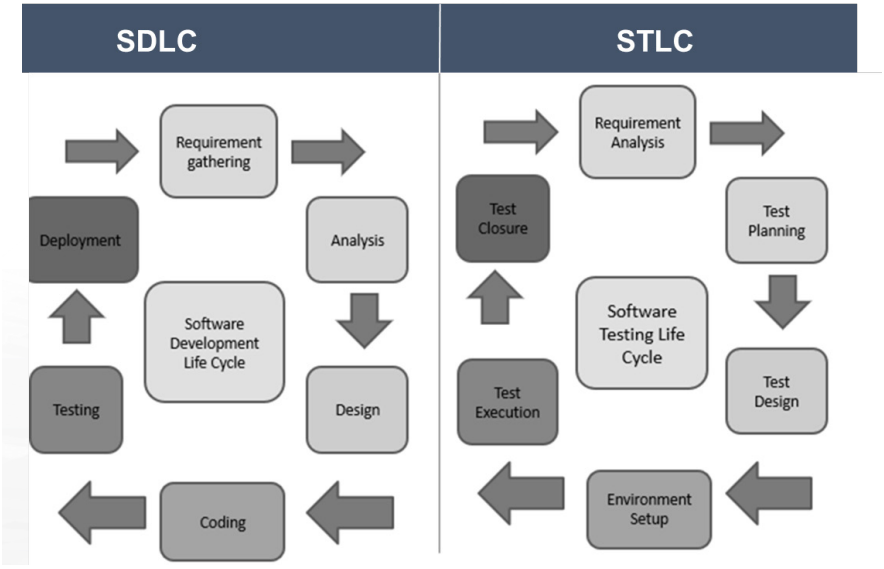
T - Testing (as the word suggests the software will be undergoing testing as it is developed - kind of in tandem)

L - Life (testing activity day 1 till end of testing)

C - Cycle (the software like its development phase will undergo a lot of phases of testing - so this defines all its phases and behavior)

STLC gives end to end ideas of what, how and when testing activities needs to be carried out. Estimation plays a pivotal role in STLC .Each module of the project is tested before the beginning of the another module.

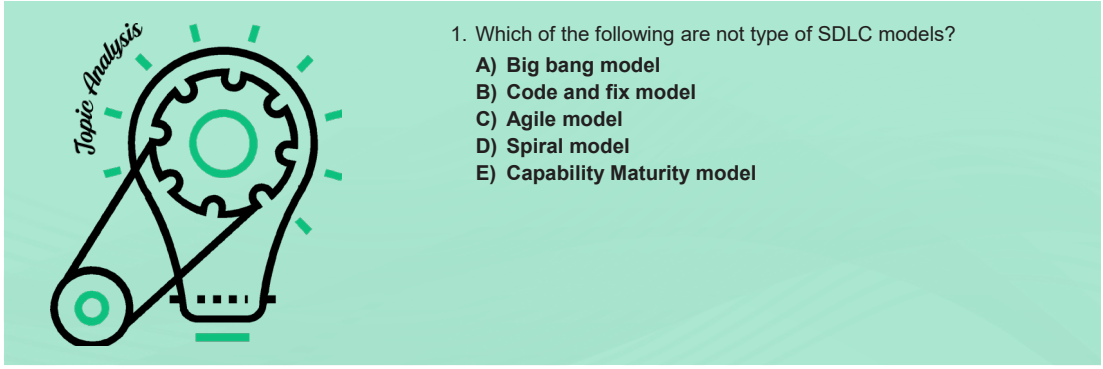
SDLC vs STLC



- SDLC is predominantly related to Software Development and has Testing as a High Level Design Phase whereas STLC is predominantly related to Software Testing and Quality Assurance.
- Participating members in SDLC includes more of Developers and Participating members in STLC includes more of Testers.
- In SDLC, Business Analysts gather the requirements from the Client and Development Plan is generally designed by the Development Architect while in STLC, test analysts create the Test Integration Plan and Test Architect designs the end-to-end testing framework.

- SDLC phase also includes post-deployment supports and updates while STLC phase follows any escalation which is encountered and does mitigation.
- In SDLC, Low Level Design[actual coding] is developed as per the BRD (business requirement document) or FRS(functional requirement specification) while in STLC, the testing team prepares the test environment and executes them.

What did you Grasp?

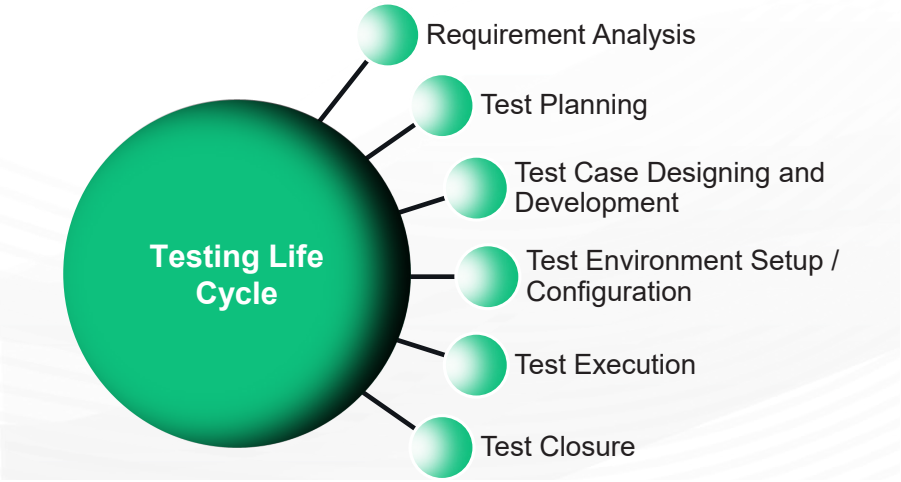


Ans: (e)

1.3 Testing Life Cycle

Testing Life Cycle is an overall operation comprising of testing activities from planning, execution, deployment and maintenance

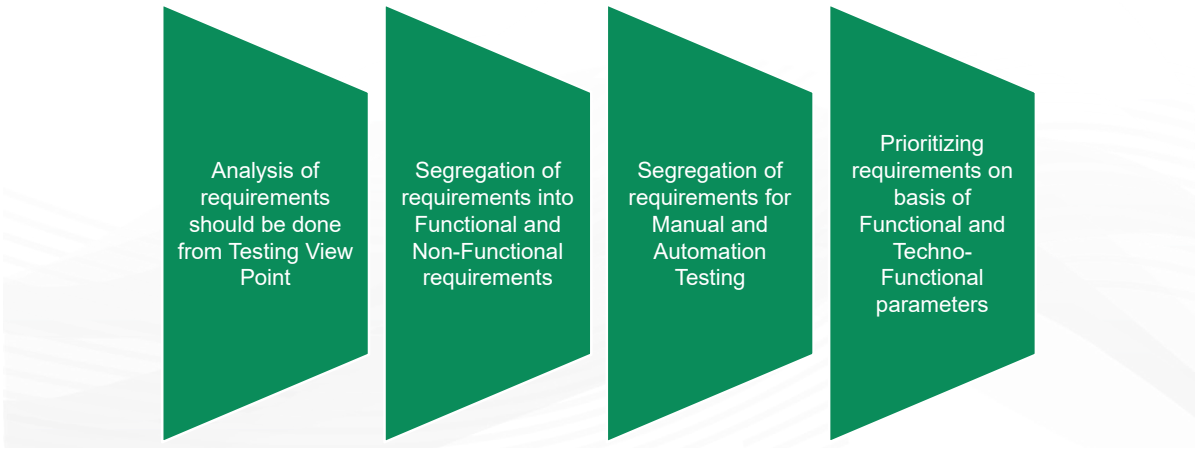
The various phases in Software Testing Life Cycle are as follows:-



Let us understand in the form of an example:- "***There is a team of developers and testers working on a Project. from the very first phase of SDLC, simultaneously the testing activities have started. The Development team is designing a certain module and testing team is ready with the testing layer ready to test once the development team hands over the developed software module to them. The developers are bound to make mistakes. So the main role of STLC is to identify those mistakes and report them adequately and get them fixed. Hence, the ultimate intent of STLC is to maintain quality and maintain Agreed Contract Level Standards"***.

1.3.1 Testing Life Cycle – Requirement Analysis

Understanding of Requirements thoroughly is very important hence,



Requirements can be received in various stages – from the very beginning or in the middle of a project in the form of a CR (Change Request). Understanding of Requirements thoroughly is very important. In case, there is a certain gap, then there is a risk of compromising on the quality and reverting the same will be a complicated effort which might lead to escalation. So understanding requirements from the very beginning is extremely important. Doubts, questions, misinterpretations should all be clarified before hand. Any time in between or in the middle, any doubt comes even while understanding the requirement should be immediately reported and addressed accordingly.

Testing team segregates the requirements into Functional and Non-Functional requirements as well. Functional requirements in some cases even get sub divided into Functional and Techno-Functional requirements as well. Manual Testing approach gets different requirement document and Automation Testing approach gets different.

Prioritization of requirements is also a part of Requirement Analysis.

1.3.1 Testing Life Cycle – Test Planning

Test Plan is a Document that describes test strategy, objectives, schedule, estimation and deliverables and resources required for testing. Test Plan is Probably the heart of the testing operation.

Test Plan contains:

- Reference documents, test items, software risk issues, features to be tested and not to be tested.
- Test strategy.
- Test items pass/fail criteria.
- Suspension criteria and resumption requirements .
- Test deliverables, environmental needs, staffing, training needs, risks, contingency plans, approvals

Test Plan is a Document rather call it a Version Control document which describes test strategy, objectives, schedule, estimation and deliverables and resources required for testing. There have been number of discussions of who designs the Test Plan, who maintains it, do authors change continuously and when can we say this is the final Test Plan Document for this project.

To answer these questions we need to understand why is Test Plan such an important document :-

It becomes a guiding reference for people outside the testing team such as developers, business managers, customers understand the details of testing. It also gives them the time interval to approach a certain functionality. And it also gives section to implement their ideas for better practices which can be introduced within the Test Plan for a better testing objective. So Test Plan also becomes quite an acceptable document which renders all ideas and comments. It is editable in nature.

Test Plan gives the testing team a structured guideline, set of rules of how, what, when and where to test – it also gives us who to approach in case we face certain difficulty. Other Testing artifacts like test estimation document, test scope, Test Strategy are documented in Test Plan and also it gives a lot of information about the various people involved in this. Who is testing what. Who has what responsibility. Who is the POC of what. Historical outline of Test Plan also plays a very important role when a new Version of a certain software is about to be tested. The old test plan acts as a reference document and also guides what best things can be taken out from the previous plan and what odds could be avoided

1.3.1 Testing Life Cycle - Test Case Designing and Development

Test Case Designing and Development should:

- Be based on the test plan, testers design and develop test cases.
- Verification and validation of specified requirements in the documentation stage
- Also, the reviewing, updating, and approval of automation scripts and test cases are essential processes of this stage.
- This phase also includes defining different test conditions with input data and expected outcomes
- Test Case Design Techniques are :-
- Specification Based Technique
- Structure Based Technique
- Experience Based Technique

Test Case Designing involves a lot of Prerequisites which needs to be followed before taking an approach. If not then it might lead to frequent changes and modifications which might affect the Estimation Process. A good TCER [Test Conditions and Expected Results] document is very important. A good test design is symbolic of how good your Test Plan is and how good all your test artifacts have been to give a good platform for designing of Test Cases.A good test case should cover all the functional and technical base of each and every scenario and

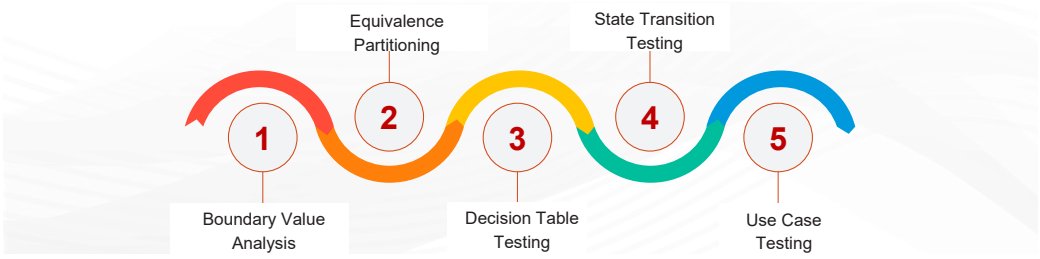
more. It should be reusable and flexible enough and the language should be simple and precise. It should not include un-necessary clumsiness and complexities.

1.3.1 Testing Life Cycle - Test Case Designing and Development

Specification Based Technique – Black Box 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.

The Specification-based or black box test case design techniques are divided further into 5 categories. These categories are as follows:



Answers:

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

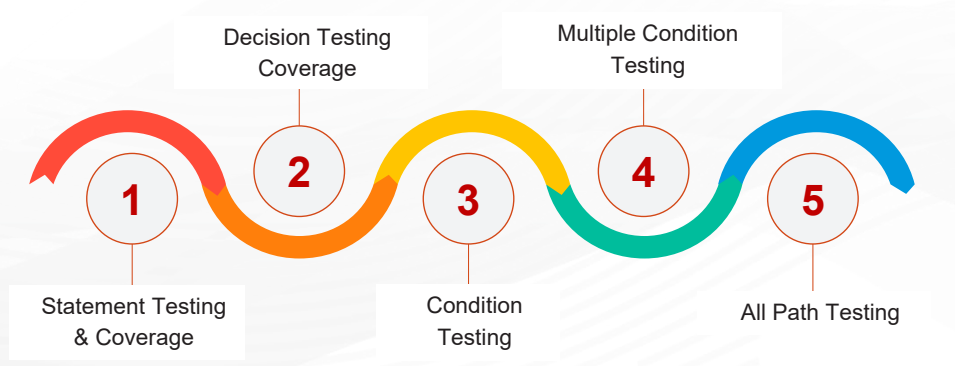
Enter invalid credentials and attempting login and validating system response.

1.3.1 Testing Life Cycle - Test Case Designing and Development

Structure Based Technique – White Box 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.

This technique is further divided into five categories:



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.

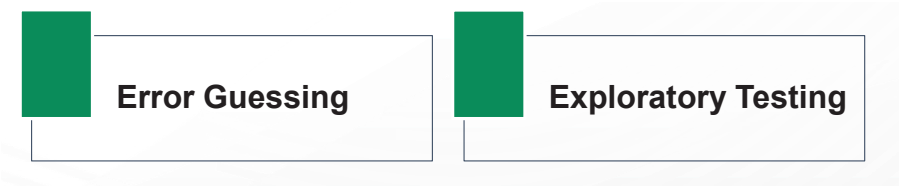
- **All Path Testing**

  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.

1.3.1 Testing Life Cycle - Test Case Designing and Development

Experience Based Technique

These techniques are highly dependent on tester's experience to understand the most important areas of the software. The outcomes of these techniques are based on the skills, knowledge, and expertise of the people involved. The types of experience-based techniques are as follows:



Experienced Based testing technique is purely based on historical behaviour, knowledge of vulnerable areas, knowledge of susceptible boundaries and trying to break the system with intuitive inputs and determining the nature of the application under test under certain payload of the data.

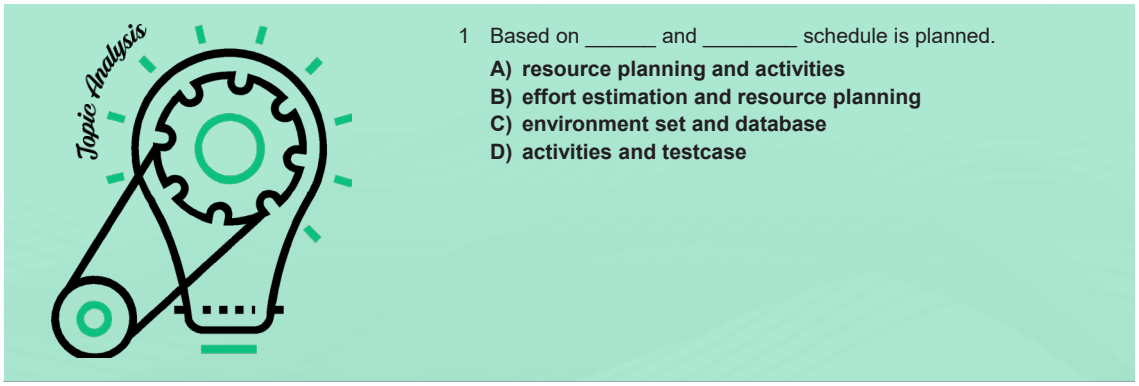
- **Error Guessing**

  In this technique, the testers anticipate errors based on their experience, availability of data and their knowledge of product failure. Error guessing is dependent on the skills, intuition, and experience of the testers.

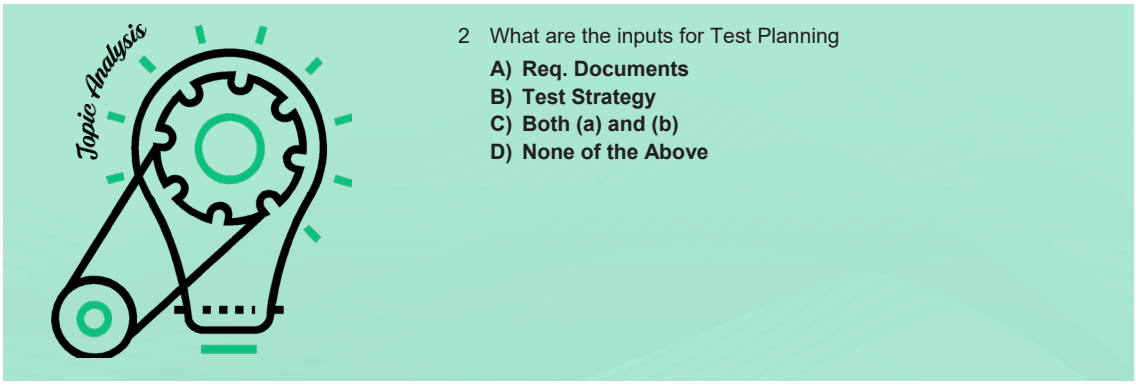
- **Exploratory Testing**

  This technique is used to test the application without any formal documentation. There is minimum time available for testing and maximum for test execution. In exploratory testing, the test design and test execution are performed concurrently.

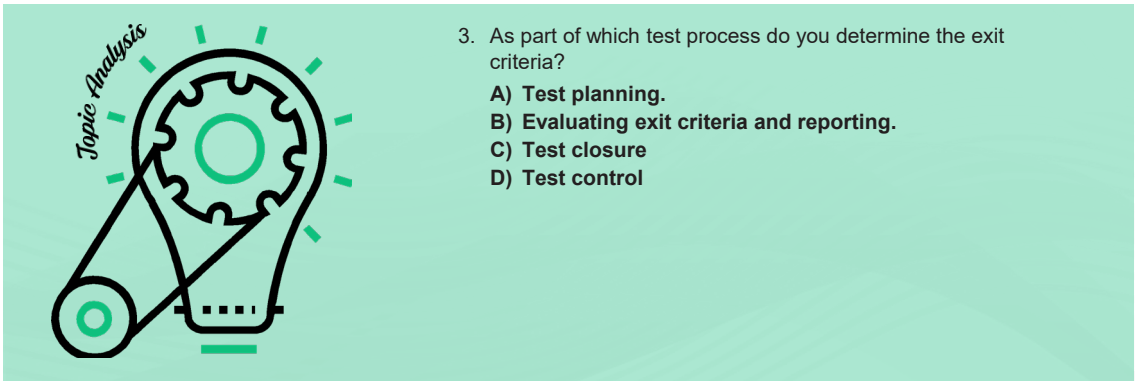
What did you grasp?



Answer: 1 : (B)

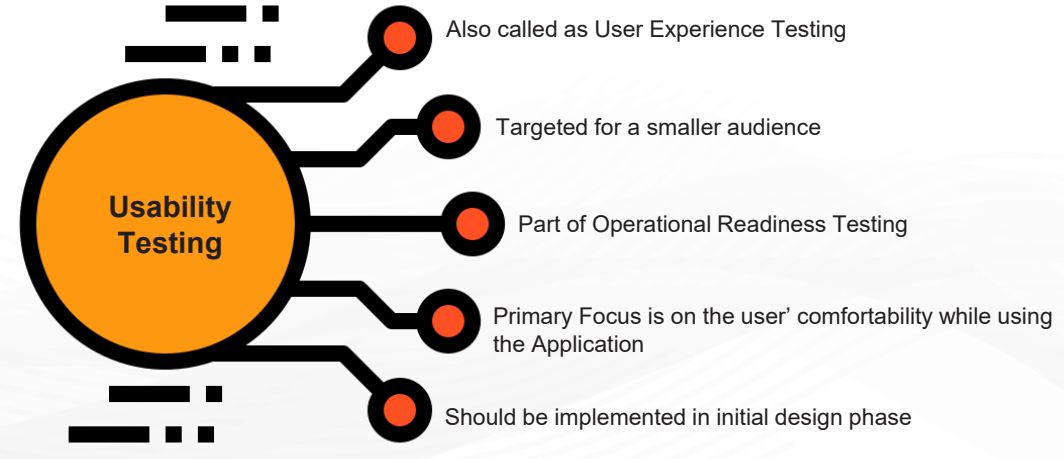


Answer: 2 : (C)



Answer: 3 : Students this one is interesting. Please discuss among yourselves and come with the right answer

1.3.2 Usability Testing



Usability testing is carried out to determine the user friendliness of the application. Let's say for example there is an old website which is undergoing a complete changeover and all the feedback, however critical of the previous versions from end users have been considered and worked upon and all the changes have been implemented in this release. So before releasing it world wide, the beta version or the alpha version is released to certain customers to get their initial rounds of feedback and as per their feedback corrections are implemented. This could do multiple rounds of Development and Testing but this enhances user friendliness with real time user feedback.

To understand more about Usability Testing, we need to understand few things first:-

Why do we need usability testing ?

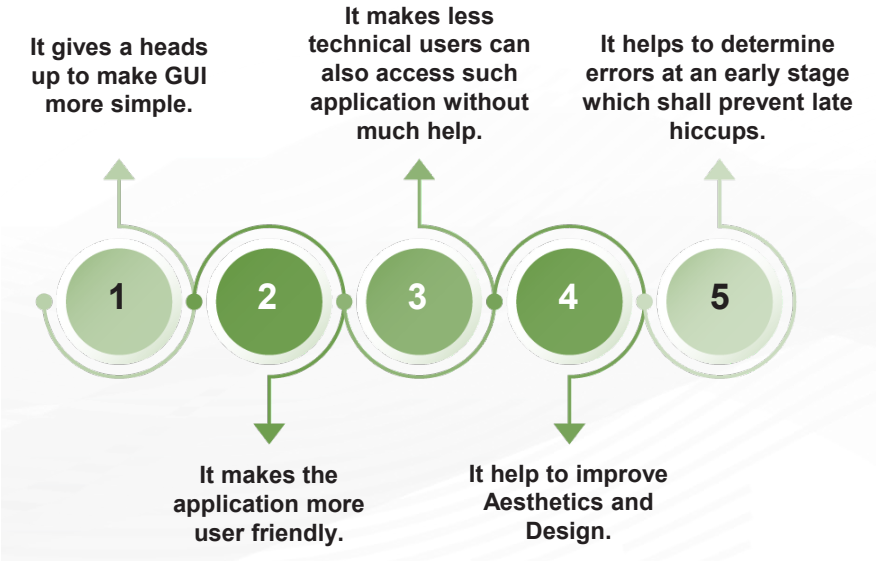
How to do Usability testing ?

Advantages of Usability testing

Disadvantages of Usability testing.

1.3.2 Usability Testing – Why do we need Usability Testing ?

Usability testing is helpful as:



Aesthetics, design, look & feel, easy accessibility are important. How user-friendly a product looks usually determines how well it works.

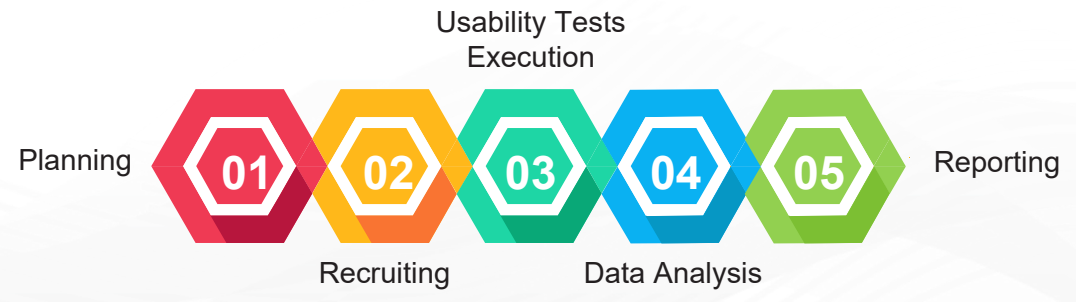
There are many software applications/websites, which miserably fail, once launched, due to following reasons -

- Users are clueless about what to do next. So calling customer care or emailing to the technical team becomes a lethargic option.
- Navigation to next page or navigation to anywhere else becomes a challenge.
- Which Icon or Link or hyperlink or flash or pop-up represents what functionality .
- Notifications, Error messages are not too convincing or are confusing.
- Session timeout and Session Cookies effectiveness.

So to counter the above issues, Usability Testing comes very handy. What matters most to end users is the GUI interface and for that Usability Testing is very important

1.3.2 Usability Testing – How to do Usability testing ?

Usability Testing process consists of these phases:-



Usability Testing process consists of these phases:

Planning:- During this phase the goals of usability test are determined. Need to determine critical functionalities and objectives of the system. Need to assign tasks to testers, which exercise these critical functionalities. During this phase, the usability testing approach, number & demographics of usability testers, test report formats are also determined

Recruiting: During this phase, recruitment of testers as per usability test plan is given a green.

Usability Tests Execution: 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 architects, developers, client's technical team, and Senior Management

1.3.2 Usability Testing – Advantages of Usability Testing

Advantages

Some advantages of Usability Testing are:

- Usability testing is helpful in uncovering the usability issues before the marketing of the product starts.
- End-user satisfaction is higher as the product is tested from their perspective.
- This not only improves the system effectiveness but also makes it more efficient.
- Usability testing provides more practical and valuable feedback from the target audience, who are using the product during this phase of testing.

Usability Testing definitely plays a major role in improving the user friendliness of the application, especially the GUI.

Let's learn via an example:-

Let's say you belong to one of the Tier 1 Cities in India and a Company named "Robot Design Tech" is launching a domestic Robot in the market. To test its first prototype and market response, the company announces releasing prototype version 1 in your Tier 1 city and it also keeps a reward "anyone who can determine a certain level of Defect will be awarded so and

so much amount cash prize or gift voucher or anything" - So a consumer who is a user, will buy the prototype and use it as well as try to find any defect in the product and report if found.

Just imagine the same kind of a situation, but instead the users are software testers and not consumers. So these testers test the product to be released keeping in perspective "How a normal user would perceive the product"

1.3.2 Usability Testing – Disadvantages of Usability Testing

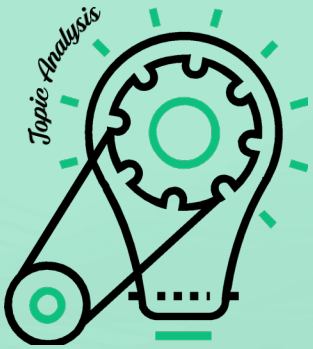
Disadvantages

Some disadvantages of Usability Testing are:

- Usability testing requires some financial burns hence costing can ne constraint in usability testing.
- Usability Test Lab setting requires lots of planning and time.
- Recruiting and management of usability testers can also be expensive
- Training testers to reach that level of expertise needs infrastructure investment
- Sometimes feedback can be ambiguous as users cannot be technically aligned with the design pattern which is a complex logic
- Re-work and multiple iterations can be time consuming and also not cost effective

Usability Testing is a good practice but good practices come at a price, so every software product does not undergo Usability Test.

What did you grasp?

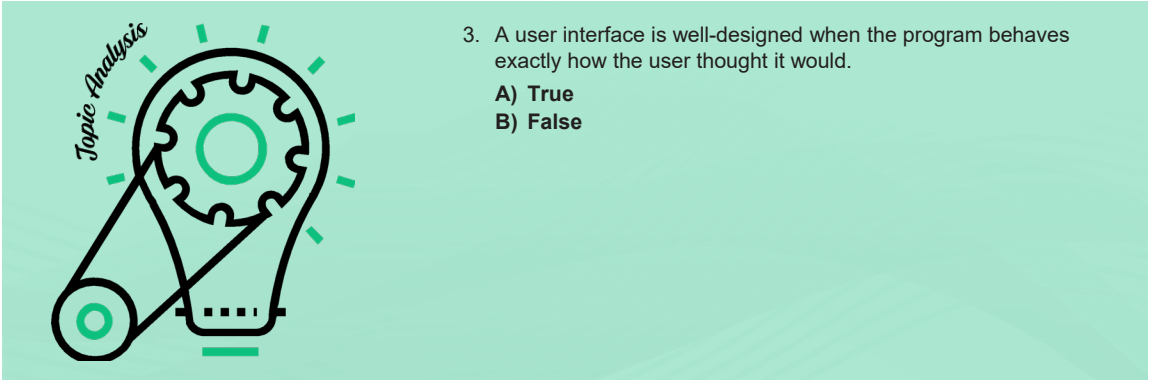


1. Goals of usability testing:
   - **A) Performance**
   - **B) Accuracy**
   - **C) Recall -- How much does the person remember afterwards or after periods of non-use?**
   - **D) All of above**

Answer: 1 : (D)

2. Think about why you will be doing a usability test:
   - **A) How do people interact with the system you are testing?**
   - **B) What is difficult or easy for people to do?**
   - **C) What makes sense about it? What is exciting about it?**
   - **D) What changes would users like to see?**
   - **E) All of above**

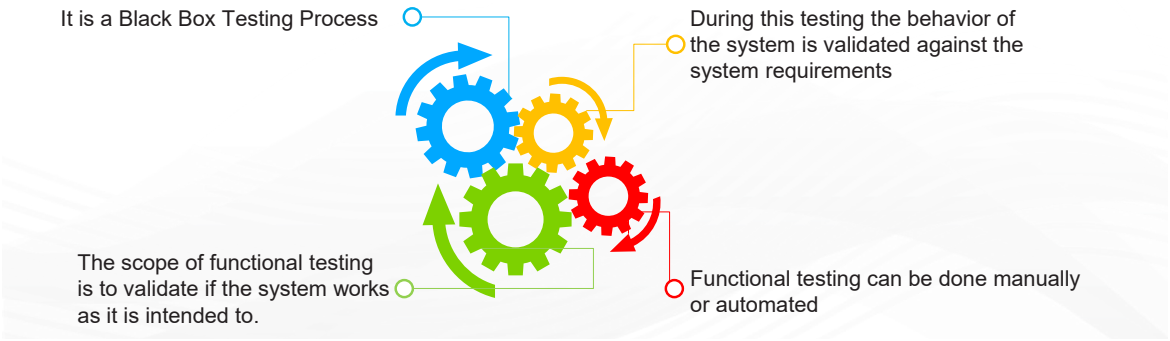
Answer: 2 : (E)



Answer: 3 : (A)

1.3.3 Functional Testing

Key features of functional testing:



Functional testing is the process through which testers determine if a piece of software is acting in accordance with predetermined requirements. It uses black-box testing techniques, where the tester need not have a deep knowledge of the internal system logic. Functional testing is only concerned with validating if a system works as intended. The tester has a set of data which acts as an inputs, tester is knowledgeable about the system' logic and also is aware of the output. Tester is also aware of the expected outputs depending upon the variations in the inputs.

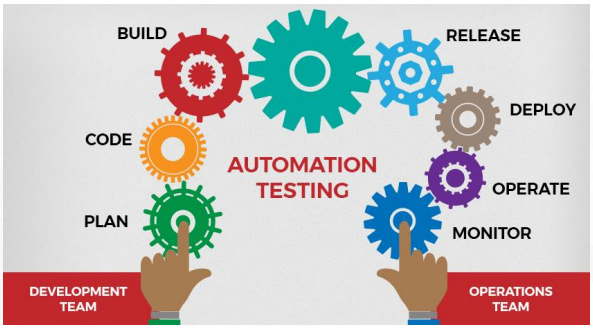
The most important observation for the testers performing the functional testing is to evaluate the compliance of a system or component with specified functional requirements. Any deviation shall be reported accordingly.

There is a distinct advantage for the functional testers. They need not know the in-depth logic of the functionality, so they will think purely on inputs and outputs and they shall match the Actual Output v/s Expected Output and pass the judgement.

However, the functional test cases should be very well documented and should undergo multiple rounds of reviews. All the Functional Testing artifacts should be up to date as functional testing mostly involves the heart of the application's response to the requirements.

1.3.4 End-to-End Testing

Key features of End-to-End testing:



- Through this testing methodology an application flow is tested from start to end
- Here, the real user scenario simulation is very important.
- The system is tested to validate how each component behaves independently and in sync with other systems as a cohesive unit

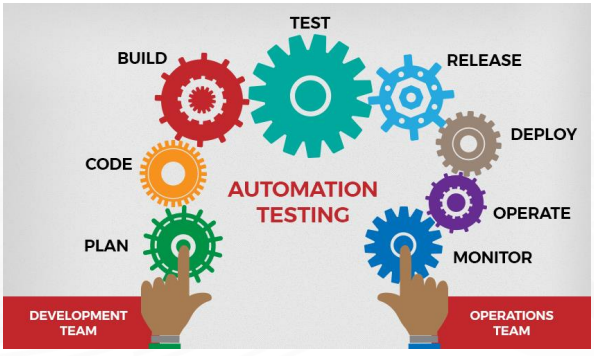
There are various End-to-End testing methods:-

- Horizontal Test
- Vertical Test

End to End testing is vast.. Let's discuss about an application which does not undergo End-to-End testing and it just follows the typical behaviour instances approach. Even if this might work well for a system as a single entity but when integrating with multiple units might fail or behave abruptly. Sometimes defects or deviations creep in or are encountered at stages of the testing process where it is least expected but it could be highly critical. So just testing the functional and technical aspect of things of a product and marking it defect free and ready for release is an amateur approach and it could be highly risky.

1.3.4 End-to-End Testing – Methods

End-to-End testing methods are:-

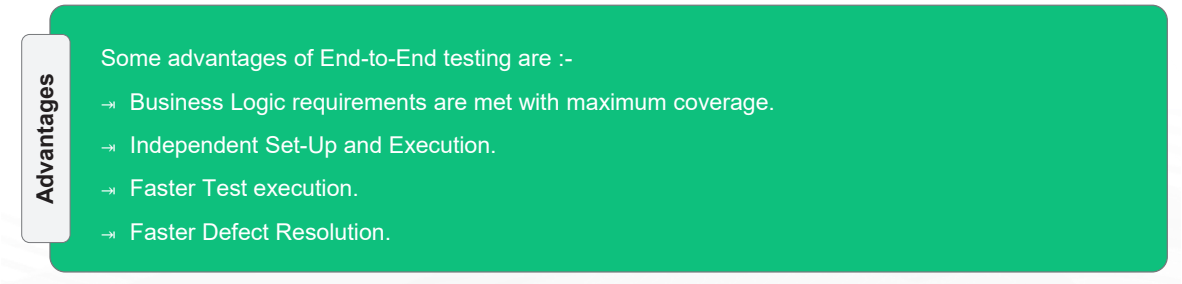


- Horizontal Test
- Vertical Test
- **Horizontal Test:** Let's say you are a common user and neither a developer nor a tester. So a normal user's perspective is kept in mind here while testing. If a user is exposed to testing a particular application' business logic from the very beginning to end and also to make sure all the interrelated logics are working fine, then the projection of a user might be different from a tester or a developer.

  Keeping the above practical logic in mind - Horizontal testing is formulated. This occurs at the end of a release cycle when all the changes have been implemented. The test environment is set up by the testing team prior to execution. Then the testers typically test each module, sub-module, inter dependent or inter-related interfaces from a user's perspective

- **Vertical Test:** This is a time taking process as it involves extra dedication and efforts to test a module as a separate entity. Success in one layer, gives a green to begin testing of the other layer. Once the other layer is completed, and depending on its success or failure, testing of the subsequent layers are determined. This cannot be independently done by a sole testers. As there has to be a lot of intervention in the form of business logic changes and other enhancement implications.Major stakeholders here are - developers, testers, product owners - all need to pitch in and assist the testing process.

1.3.4 End to End Testing – Advantages



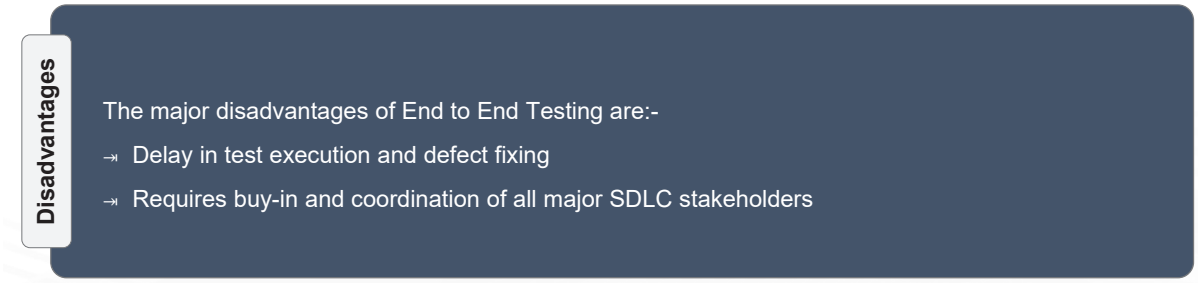
Applications are getting complicated these days. There are major integrations, dependent systems, build tools, compile tools, third party tools and Repositories. So it can happen that systems might pass certain tests individually but when integrated together with multiple systems to work as one unit – the same tests might fail. Conducting end-to-end testing helps to ensure the software is production ready, layer by layer.

So end-to-end testing ensures that health of the application as a single entity and as an integrated unit remains good and it works as intended. It also expands test coverage beyond what's normally considered in more isolated testing practices. It helps in mitigating risks

In today's modern methodologies, end-to-end testing gives a distinct advantage of identification of errors at an early stage resulting in faster resolutions and increased productivity. It also reduces the complexity which might get involved of finding a bug at a deeper phase of SDLC and thereby, become more cost effective in nature

The one most wonderful advantage of end-to end testing is that it can help in testing the system as a single unit, as well as testing the system as a cohesive unit. So with the same approach, single and multiple integrated platforms can be tested which gives it quite a diversity.

1.3.4 End to End Testing – Disadvantages

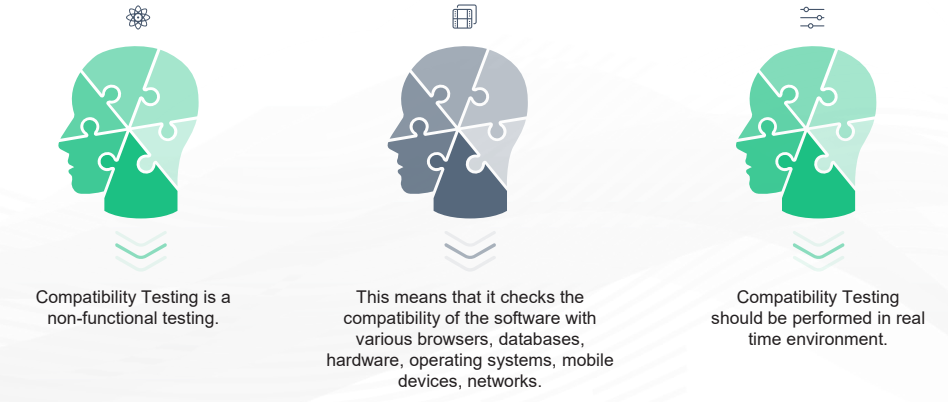


End to End Testing has lots of advantages but in some cases it tends to invoke multiple rounds of testing with every defect fixing and it tends to delay execution of test cases as regression becomes important at every stage.

Such practices, also need approval and support of all the senior personnel within the Organization. They tend to keep a strict vigil over things which may delay the release.

1.3.5 Compatibility Testing

Key features of compatibility testing are:

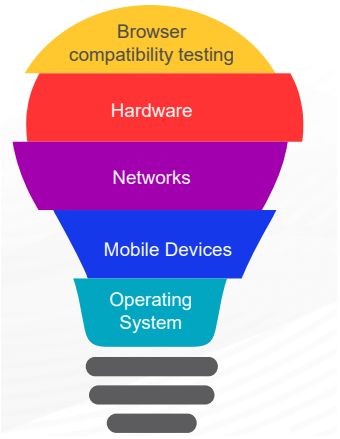


It is all about meeting customer needs these days. So compatibility testing is a major stepping stone in that direction. Today's software has to be vivacious enough to run in multiple browsers, databases, hardware, operating systems, mobile devices, networks. And there are also various versions of these. So to determine the minimum version on which the software should run and verifying the same is the basic agenda in compatibility testing.

It is much advisable to be tested in a real time environment rather than a virtual environment as it will give honest feedback of how system is behaving.

1.3.5 Compatibility Testing - Types

The various types of compatibility testing are listed below:



Browser compatibility testing:-

With emergence of various browsers in the market, and end users being flexible to use quite a bunch of them, any software application should run alongside these browsers with respective versions. Their match, their response, their acknowledgement with respect to the application needs quite thorough testing. Even comparisons are drawn as in which browser is the most compatible for this application as a form of a suggestion. So browser compatibility testing is very important these days

Hardware:-

Complexity has not helped outdated systems to thrive in the market these days. With software of any form becoming more and more complex, it requires high performing hardware platforms to be operational at full capacity. So hardware compatibility is always measured at an initial stage and hardware compatibility becomes an inevitable exercise in today's age.

Networks:-

Gone are the days of 2G and 3G. LTE has no limits and soon 4G will become outdated. So to keep proper tab of today' growing network bandwidth, typical applications these days have a design of that sync to make sure how it works well with the minimum bandwidth available to it.

Mobile Devices:-

These also fall in the category of hardware but mobile devices have a different sort of hardware and they adapt to network compatibility in a different way. So with various versions of iOS or Android and behavior with respect to change and upgrade in software versions, compatibility tends to become a serious cross check. Some might fail in UI, some in functionality. So Mobile device compatibility is of paramount importance

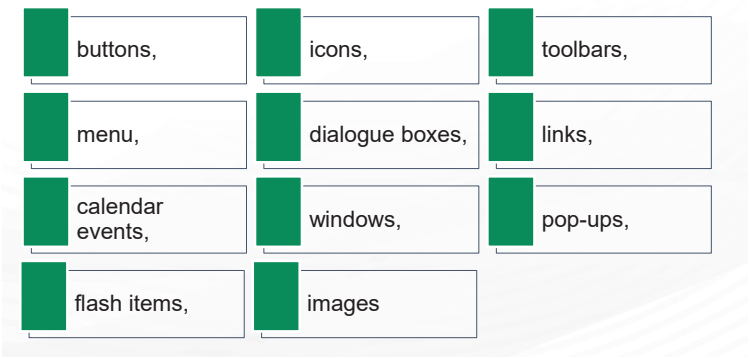
Operating System:-

Checks whether software is compatible with Windows, Mac OS or Linux or other Operating Systems

1.3.6 GUI Testing

GUI (Graphical User Interface) testing is defined as the process of testing the system's Graphical User Interface of the Application Under Test.

It involves testing of all the GUI elements –



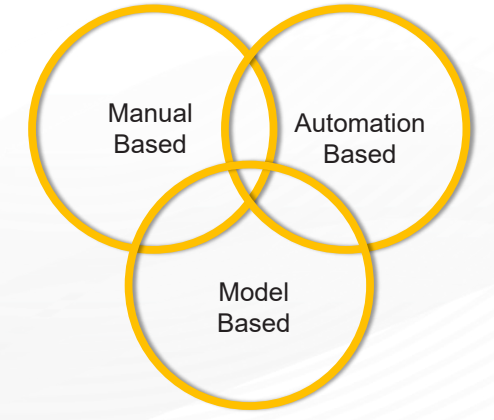
Without even knowing end users end up doing GUI testing almost on a daily basis though their goal is not to try to break the system, rather it is to complete their task as per their need. GUI testing is mostly checking the overall visibility of the application page. As we all know, there are various elements present in a webpage which are mentioned above and testing the behaviour of these elements with respect to a certain pre-condition followed by an action is what is called GUI Testing.

Lot of automation suites also help in doing GUI testing some of which are : Selenium, Sikuli, Water, Dojo Toolkit, etc.

GUI Testing is unavoidable. They fall under category of BlackBox testing but it requires quite a bit of thorough checks.

1.3.6 GUI Testing – Techniques

Various GUI testing techniques are:



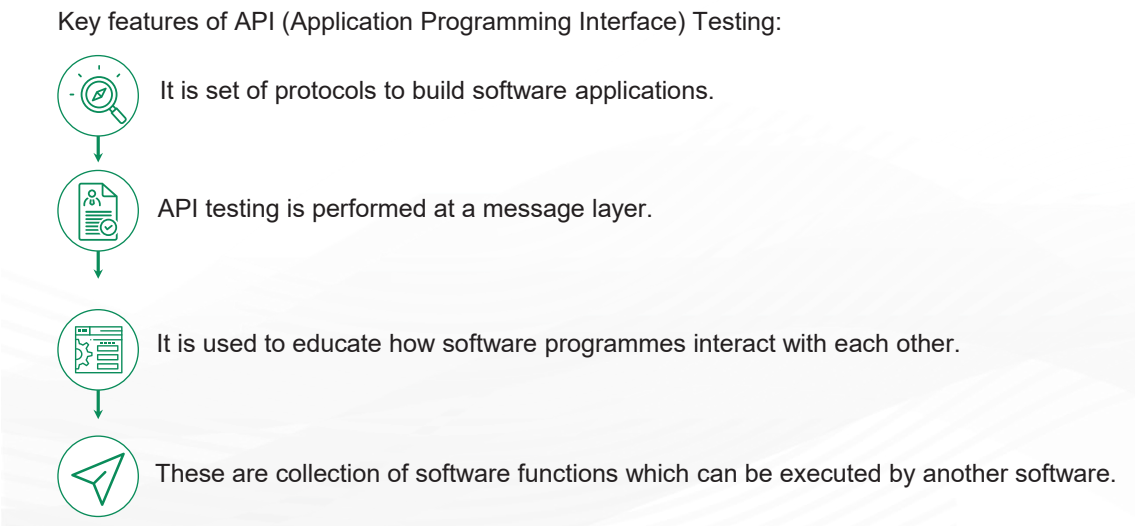
Manual Based: Testers follow the BRD or FRS documents and especially for GUI, a lot of images are mentioned along with lot of markings to give precise workability of these elements. Testers follow these documents and do the testing and report the defects accordingly.

Automation Based: GUI testing can be done using automation practices. Again manual testing has to be present as the automation tests need to mention assertions of various degree to validate these elements.

Model Based: A model is a graphical description of a system's behavior. It helps to understand and predict the system behavior. Models help in a generation of efficient test cases using the system requirements. Need to build the model, validate inputs for the model, assert the outputs for the model, run the tests for the model, expected output v/s actual output.

1.3.7 API Testing

Key features of API (Application Programming Interface) Testing:

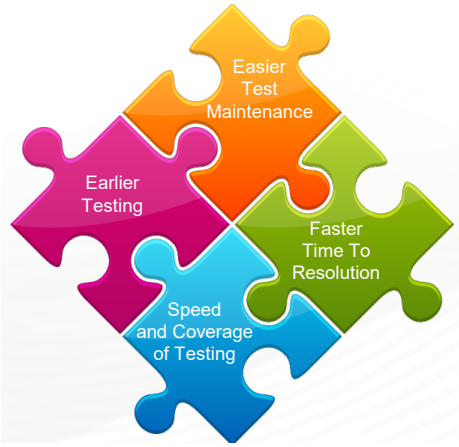


APIs, or Application Programming Interfaces, they act as connectors between different systems or system layers of an application. Due to high profile security and blockades and copyrights – accessing a certain system would not be possible unless APIs are involved. APIs customize what is the permissible limit of information. GET, PUT, POST, DELETE – these API calls determine what an application can return and what it can absorb depending on the access rights.

Let's take an example of Google. Google has various services. One of the most popular is Google Maps. We use Google Maps in Ola, Uber and what not and where not. Does this mean Google is exposing its Google Map code to every software who need it ? Answer is NO. What Google does is it exposes its Google Map APIs to these these software but not the code.

1.3.7 API Testing - Advantages

Listed below are some advantages of API testing



Imagine life without Google Maps. Imagine life without makemytrip, goibibo, easemytrip, trivago. Imagine remaining hungry without Swiggy and Zomato

Now further imagine life without Amazon.in and Flipkart.com All these above mentioned Websites – expose their APIs and these APIs are the ones which we receive and live our life on. They are easy to test using a tool known as 'Postman'. Also,these can be automated by learning REST Assured API Testing. They are not difficult to understand either as they are mostly in JSON format which is a key value pair aggregation.

The deviations are identified immediately and it does not take much time to fix these. API Testing is in a world in itself. And API Testers are also paid very handsomely

In a nutshell, we learnt:

1. Seven principles of Software Testing
2. SDLC vs STLC
3. Testing Life Cycle
4. Usability Testing
5. Functional Testing
6. End to End Testing
7. Compatibility Testing
8. GUI testing
9. API testing

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

- Seven principles of Software Testing
- SDLC vs STLC
- Testing Life Cycle
- Usability Testing
- Functional Testing
- End to End Testing
- Compatibility Testing
- GUI testing
- API 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