**Module–2(Manual Testing)**

**1. What is Exploratory Testing?**

**»**• Exploratory testing is a type of manual testing that relies on testers’ experience and intuition, rather than scripts, to guide the testing process. It is especially good for catching issues that slip through the cracks with automated tests, for testing UI elements and for ensuring software actually meets its requirements. Exploratory testers are often guided by “charters,” which set out the ultimate goals for testing but do not tell the tester what to do or focus on.

**»**• Exploratory Testing is a type of software testing where Test cases are not created in advance but testers check system on the fly. They may note down ideas about what to test before test execution. The focus of exploratory testing is more on testing as a “thinking” activity.

**»**• Exploratory Testing is widely used in Agile models and is all about discovery, investigation, and learning. It emphasizes personal freedom and responsibility of the individual tester.

**2. What is traceability matrix?**

**»**• A Traceability Matrix is a document that co-relates any two-baseline documents that require a many-to-many relationship to check the completeness of the relationship.

**»**• It is used to track the requirements and to check the current project requirements are met. • RTM Requirement Traceability Matrix (RTM) is a document that maps and traces user requirement with test cases. It captures all requirements proposed by the client and requirement traceability in a single document, delivered at the conclusion of the Software development life cycle. The main purpose of Requirement Traceability Matrix is to validate that all requirements are checked via test cases such that no functionality is unchecked during Software testing.

**3. What is Boundary value testing?**

**»**• Boundary Value Analysis is based on testing the boundary values of valid and invalid partitions. The behavior at the edge of the equivalence partition is more likely to be incorrect than the behavior within the partition, so boundaries are an area where testing is likely to yield defects.

**»**• It checks for the input values near the boundary that have a higher chance of error. Every partition has its maximum and minimum values and these maximum and minimum values are the boundary values of a partition.

**4. What is Equivalence partitioning testing?**

**»**• Equivalence Partitioning Method is also known as Equivalence class partitioning (ECP). It is a software testing technique or black-box testing that divides input domain into classes of data, and with the help of these classes of data, test cases can be derived. An ideal test case identifies class of error that might require many arbitrary test cases to be executed before general error is observed.

**»**• In equivalence partitioning, equivalence classes are evaluated for given input conditions. Whenever any input is given, then type of input condition is checked, then for this input conditions, Equivalence class represents or describes set of valid or invalid states.

**5. What is Integration testing?**

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

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

**6. What determines the level of risk?**

**»**• Determining the level of risk usually involves trying to assess not only the likelihood of an identified risk from actually occurring, but also the potential magnitude the consequences this risk could have on an organisation and its stakeholder, should it occur.

• High

• The impact of this risk would be very damaging and potentially nontolerable. Ultimately, a risk of this scale could see the organisation make a loss should it occur. If a high risk is identified and cannot be solved, the software project may be too big of a risk to complete.

• Medium

• Problems, challenges or glitches labeled as ‘medium risk’ may be tolerable but are certainly not desirable. These risks may see financial loss in the short-term, however, if a solution can be found, the positives of finishing the software project will outweigh the negative risks.

• Low

• ‘Low’ risks can almost be classed as inconveniences or minor snags rather than actual threats to the project. Little to no financial loss would be seen in the event of one of these risks playing out

**7. What is Alpha testing?**

**»**• Alpha testing is the first end-to-end testing of a product to ensure it meets the business requirements and functions correctly. It is typically performed by internal employees and conducted in a lab/stage environment. An alpha test ensures the product really works and does everything it's supposed to do.

**8. What is beta testing?**

**»**• Beta testing software evaluates product performance in the real world, prior to an official product launch, by obtaining feedback from a targeted group of users.

• Beta testing is a form of user acceptance testing and takes place after alpha testing. Beta tests help to isolate any bugs and issues that may have been missed during earlier testing phases, which if left, could severely impact the quality and stability of the software.

**9. What is component testing?**

**»**• Component testing, also known as program or module testing, is done after unit testing. In this type of testing those test objects can be tested independently as a component without integrating with other components e.g. modules, classes, objects, and programs. This testing is done by the development team.

**»**• Component testing is like unit testing with the difference that the developer uses real data instead of dummy data for testing of the written code. Suppose there is a software application which consists of five components modules. The testing of each module is done independently by the developer as part of the development cycle before it is ready for integration testing. By doing component testing,

bugs can be found at a very early stage in the cycle and helps save time. Debugging tools or unit test structure tools are used for this type of testing since this is done by programmers on the code written by them and with the support of integrated development environment. Defects are fixed as soon as possible when they are found without formally recording incidents. Component testing plays an important role in finding the issue. Before we proceed with the integration testing it’s always advised to do the component testing in order to ensure that each module of an application is working correctly and as per requirement.

**10.What is functional system testing?**

**»**• Functional testing is a type of testing that seeks to establish whether each application feature works as per the software requirements. Each function is compared to the corresponding requirement to ascertain whether its output is consistent with the end user’s expectations. The testing is done by providing sample inputs, capturing resulting outputs, and verifying that actual outputs are the same as expected outputs.

**»**• At the end of functional testing, you should have software that has a coherent user interface, a consistent API, and seamlessly integrates with business processes functional testing focuses on the results of processing and not the mechanics of the processing, and determines whether the application satisfies the basic minimum user expectations.

**11.What is Non-Functional Testing?**

**»**• Non-Functional Testing is defined as a type of Software testing to check non-functional aspects (performance, usability, reliability, etc) of a software application. It is designed to test the readiness of a system as per nonfunctional parameters which are never addressed by functional testing.

**• Types of Non Functional Testing :**

1. Performance Testing

2. Load Testing

3. Failover Testing

4. Compatibility Testing

5. Usability Testing

6. Stress Testing

7. Maintainability Testing

8. Scalability Testing

9. Volume Testing

10.Security Testing

11.Disaster Recovery Testing

12.Compliance Testing

13.Portability Testing

14.Efficiency Testing

15.Reliability Testing

16.Baseline Testing

17.Endurance Testing

18.Documentation Testing

**12.What is GUI Testing ?**

**»**• GUI Testing is a software testing type that checks the Graphical User Interface of the Software. The purpose of Graphical User Interface (GUI) Testing is to ensure the functionalities of software application work as per specifications by checking screens and controls like menus, buttons, icons, etc.

• A user doesn’t have any knowledge about XYZ software/Application. It is the UI of the Application which decides that a user is going to use the Application further or not.

• A normal User first observes the design and looks of the Application/Software and how easy it is for him to understand the UI. If a user is not comfortable with the Interface or find Application complex to understand he would never going to use that Application Again. That’s why, GUI is a matter for concern, and proper testing should be carried out in order to make sure that GUI is free of Bugs.

**13.What is Adhoc testing ?**

**»**• Ad hoc Testing is an informal or unstructured software testing type that aims to break the testing process in order to find possible defects or errors at an early possible stage. Ad hoc testing is done randomly and it is usually an unplanned activity which does not follow any documentation and test design techniques to create test cases.

• Ad hoc Testing does not follow any structured way of testing and it is randomly done on any part of application. Main aim of this testing is to find defects by random checking. Adhoc testing can be achieved with the Software testing technique called Error Guessing. Error guessing can be done by the people having enough experience on the system to “guess” the most likely source of errors.

• This testing requires no documentation/ planning /process to be followed. Since this testing aims at finding defects through random approach, without any documentation, defects will not be mapped to test cases. This means that, sometimes, it is very difficult to reproduce the defects as there are no test steps or requirements mapped to it.

**14.What is load testing?**

**»**• Load Testing is a type of Performance Testing that determines the performance of a system, software product, or software application under real-life based load conditions. Basically, load testing determines the behavior of the application when multiple users use it at the same time. It is the response of the system measured under varying load conditions. The load testing is carried out for normal and extreme load conditions.

• Load testing is a type of performance testing that simulates a real-world load on a system or application to see how it performs under stress. The goal of load testing is to identify bottlenecks and determine the maximum number of users or transactions the system can handle. It is an important aspect of software testing as it helps ensure that the system can handle the expected usage levels and identify any potential issues before the system is deployed to production.

• During load testing, various scenarios are simulated to test the system’s behavior under different load conditions. This can include simulating a high number of concurrent users, simulating a large number of requests, and simulating heavy network traffic. The system’s performance is then measured and analyzed to identify any bottlenecks or issues that may occur.

• Some common load testing techniques include:

• Stress testing: Testing the system’s ability to handle a high load above normal usage levels

• Spike testing: Testing the system’s ability to handle sudden spikes in traffic

• Soak testing: Testing the system’s ability to handle a sustained load over a prolonged period of time

• Tools such as Apache JMeter, LoadRunner, Gatling, and Grinder can be used to simulate load and measure system performance. It’s important to ensure that the load testing is done in an environment that closely mirrors the production environment to get accurate results.

**15.What is stress Testing ?**

**»**• Stress Testing is a software testing technique that determines the robustness of software by testing beyond the limits of normal operation. Stress testing is particularly important for critical software but is used for all types of software. Stress testing emphasizes robustness, availability, and error handling under a heavy load rather than what is correct behavior under normal situations. Stress testing is defined as a type of software testing that verifies the stability and reliability of the system.

• This test particularly determines the system on its robustness and error handling under extremely heavy load conditions. It even tests beyond the normal operating point and analyses how the system works under extreme conditions. Stress testing is performed to ensure that the system would not crash under crunch situations. Stress testing is also known as Endurance Testing or Torture Testing.

**16.What is white box testing and list the types of white box testing?**

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

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

**17.What is black box testing? What are the different black box testing techniques?**

**»**• Black box testing involves testing a system with no prior knowledge of its internal workings. A tester provides an input, and observes the output generated by the system under test. This makes it possible to identify how the system responds to expected and unexpected user actions, its response time, usability issues and reliability issues.

• Black box testing is a powerful testing technique because it exercises a system end-to-end. Just like end-users “don’t care” how a system is coded or architected, and expect to receive an appropriate response to their requests, a tester can simulate user activity and see if the system delivers on its promises. Along the way, a black box test evaluates all relevant subsystems, including UI/UX, web server or application server, database, dependencies, and integrated systems.

**18.Mention what are the categories of defects.**

**»**• Software Defect is some kind of error, flaw or some kind of mistake from the development team which prevent the software from the smooth working. It directly affect software quality, software quality is some thing how smooth and reliable your software is. Smoothness and reliability is how less defects your software have.

• Categories of defects: Categories of defects are: Errors of commissions, Errors of omissions, Errors of clarity, and Error of speed and capacity.

**19.Mention what bigbang testing is?**

**»**• Big Bang Integration Testing is an integration testing strategy wherein all units are linked at once, resulting in a complete system. When this type of testing strategy is adopted, it is difficult to isolate any errors found, because attention is not paid to verifying the interfaces across individual units.

**20.What is the purpose of exit criteria?**

**»**• Exit criterion is used to determine whether a given test activity has been completed or NOT. Exit criteria can be defined for all of the test activities right from planning, specification and execution.

• Exit criterion should be part of test plan and decided in the planning stage.

• Verify if All tests planned have been run.

• Verify if the level of requirement coverage has been met.

• Verify if there are NO Critical or high severity defects that are left outstanding.

• Verify if all high risk areas are completely tested.

• Verify if software development activities are completed within the projected cost.

• Verify if software development activities are completed within the projected timelines.

**21.When should "Regression Testing" be performed ?**

**»**• Regression testing should be performed after introducing new code changes to ensure they have not impacted the existing functionality of your system. This kind of testing can be done using manual and automated techniques

• whenever changes are made to the system, including new features, bug fixes, and performance improvements. It is an important part of the software development process and can help to ensure that changes made to a system do not introduce new bugs.

**22.What is 7 key principles? Explain in detail?**

**»**• According to the ISTQB (International Software Testing Qualifications Board), the seven principles of software testing are:

1. Testing shows the presence of defects.

2. Exhaustive testing is impossible.

3. Early testing.

4. Defect clustering.

5. Pesticide paradox.

6. Testing is context dependent.

7. Absence-of-errors fallacy.

Testing shows the presence of defects You test software to identify problems so you can fix them before you deploy the software to production environments. However, this process doesn't mean that there aren't any bugs in the product. It just means that there may be bugs, but you didn't find them. There could be any number of reasons that you didn't uncover every bug, including the fact that the test cases didn't cover every scenario. This principle, which helps to set stakeholder expectations, means that you shouldn't guarantee that the software is error-free. Exhaustive testing is impossible The truth is that you can't test everything, i.e., every combination of preconditions and inputs. And if you try to do so you'll waste time and What you need to do is assess risk and plan your tests around these risks so you can be sure you're testing the key functions. money, but it won't affect the overall quality of the software.

Careful planning and assessment ensures your test coverage is good so you can have confidence in your final product — and you don't even have to test every individual line of code. Early testing When it comes to the software development lifecycle, testing early is the key to identifying any defects in the requirements or design phase as soon as possible. It's much easier and less expensive to fix bugs in the early stages of testing than at the end of the software lifecycle as then you might have to rewrite entire areas of functionality. And that likely means missed deadlines and cost overruns.

Defect clustering Defect clustering is the idea that a small number of software modules or components contain the most defects — sort of applying the Pareto Principle to software testing, i.e., approximately 80% of the issues are found in 20% of the components.

Understanding this can help in your testing because if you find one defect in a particular area, you'll likely find more in that same module. If you identify the complex areas that are changing the most or the ones that have more dependencies, you can focus your testing on these key areas of risk. Pesticide paradox This principle centers around the theory that if you repeatedly use a particular pesticide on your crops, the insects you're trying to kill or repel will eventually become immune to the pesticide and it will no longer be effective. Likewise, if you continuously run the same tests, eventually they'll fail to find new defects, even though they'll probably confirm the software is working. Consequently, you must continue to review your tests as well as add to your scenarios or modify them to help prevent this pesticide paradox. For example, maybe you could use a variety of testing techniques, methods, and approaches simultaneously.

Testing is context dependent Software testing is all about the context, which means that no one strategy will fit every scenario. The types of testing and the methods you use totally depend on the context of the systems or the software, e.g., the testing of an iOS application is different from the testing of an e-commerce website. Put simply, what you're testing will always affect the approach you use. Absence-of-errors fallacy If your software is 99% error-free but it doesn't follow your user's requirements, it's still not usable. That's why it's critical to run tests that pertain to the requirements of the system. Software testing isn't just about finding bugs, it's about ensuring that the software meets the user's needs and requirements. As such, you should also test your software with the users. You can test against early prototypes at the usability testing phase so you can get feedback from the users that you can use to ensure the software is usable. Even though your software might have relatively few issues, doesn't mean it is ready to ship; it also has to meet your customer's requirements and expectations.