1. **What is traceability matrix?**

It is graph of requirement vs component which represent that we should able to track back form every system component to original requirement.

Test conditions should be able to be linked back to their sources

in the test basis, this is known as traceability.

Traceability can be horizontal through all the test documentation for a

given test level (e.g. system testing, from test conditions through test

cases to test scripts) or it can be vertical through the layers of

development documentation (e.g. from requirements to components).

To protect against changes you should be able to trace back from every

system component to the original requirement that caused its presence.

A software process should help you keeping the virtual table up-to-date.

Simple technique may be quite valuable (naming convention)Forward Traceability – Mapping of Requirements to Test cases

Backward Traceability – Mapping of Test Cases to Requirements

Bi-Directional Traceability - A Good Traceability matrix is the

References from test cases to basis documentation and vice

versa.

**Pros of Traceability Matrix**

* Make obvious to the client that the software is being developed as per the requirements.
* To make sure that all requirements included in the test cases
* To make sure that developers are not creating features that no one has requested
* Easy to identify the missing functionalities.
* If there is a change request for a requirement, then we can easily find out which test cases need to update.
* The completed system may have “Extra” functionality that may have not been specified in the design specification, resulting in wastage of manpower, time and effort.

**Cons of Traceability Matrix**

* Poor or unknown test coverage, more defects found in production
* It will lead to miss some bugs in earlier test cycles which may arise in later test cycles. Then a lot of discussions arguments with other teams and managers before release.
* Difficult project planning and tracking, misunderstandings between different teams over project dependencies, delays, etc.

1. **What is Integration testing?**

* Integration Testing - Testing performed to expose defects in the interfaces and in the interactions between integrated components or systems.
* Integration Testing is a level of the software testing process where individual units are combined and tested as a group.
* The purpose of this level of testing is to expose faults in the interaction between integrated units. Test drivers and test stubs are used to assist in Integration Testing.
* Integration testing tests integration or interfaces between components, interactions to different parts of the system such as an operating system, file system and hardware or interfaces between systems.
* Integration testing is done by a specific integration tester or test team.
* Components may be code modules, operating systems, hardware and even complete systems
* There are 2 levels of Integration Testing
  + Component Integration Testing
  + System Integration Testing

**Need of Integration Testing**

* A Module in general is designed by an individual software developer who understanding and programming logic may differ from other programmers. Integration testing becomes necessary to verify the software modules work in unity
* At the time of module development, there wide chances of change in requirements by the clients. These new requirements may not be unit tested and hence integration testing becomes necessary.
* Interfaces of the software modules with the database could be erroneous.
* External Hardware interfaces, if any, could be erroneous
* Inadequate exception handling could cause issues.

**Component Integration Testing**

* Component Integration Testing: Testing performed to expose defects in the interfaces and interaction between integrated components
* Usually formal (records of test design and execution are kept)
* All individual components should be integration tested prior to system testing
* It tests the interactions between software components and is done after component testing.
* The software components themselves may be specified at different times by different specification groups, yet the integration of all of the pieces must work together.
* It is important to cover negative cases as well because components might make assumption with respect to the data.
* The following testing techniques are appropriate for Integration Testing:
  + Functional Testing using Black Box Testing techniques against the interfacing requirements for the component under test
  + Non-functional Testing (where appropriate, for performance or reliability testing of the component interfaces, for example)

**System Integration Testing**

* It tests the interactions between different systems and may be done after system testing.
* It verifies the proper execution of software components and proper interfacing between components within the solution.
* The objective of SIT Testing is to validate that all software module dependencies are functionally correct and that data integrity is maintained between separate modules for the entire solution.
* As testing for dependencies between different components is a primary function of SIT Testing, this area is often most subject to Regression Testing.

**Integration Testing Methods**

* During the process of manufacturing a ballpoint pen, the cap, the body, the tail and clip, the ink cartridge and the ballpoint are produced separately and unit tested separately. When two or more units are ready, they are assembled and Integration Testing is performed. For example, whether the cap fits into the body or not.
* Any of Black Box Testing, White Box Testing, and Gray Box Testing methods can be used. Normally, the method depends on your definition of ‘unit’.
* There is two types methods of Integration Testing:
  + Bing Bang Integration Testing
  + Incremental Integration Testing
    - Top Down Approach
    - Bottom Up Approach

**Big Bang Integration Testing**

* In Big Bang integration testing all components or modules is integrated simultaneously, after which everything is tested as a whole.
* Big Bang testing has the advantage that everything is finished before integration testing starts.
* The major disadvantage is that in general it is time consuming and difficult to trace the cause of failures because of this late integration.
* Here all component are integrated together at once, and then tested.

**Advantages:**

* Convenient for small systems.

**Disadvantages:**

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

**Top Down Approach**

* Testing takes place from top to bottom, following the control flow or architectural structure (e.g. starting from the GUI or main menu). Components or systems are substituted by stubs.
* In Top to down approach, testing takes place from top to down following the control flow of the software system.

**Advantages:**

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

**Disadvantages:**

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

**Bottom Up Approach**

* Testing takes place from the bottom of the control flow upwards. Components or systems are substituted by drivers.
* In the bottom up strategy, each module at lower levels is tested with higher modules until all modules are tested. It takes help of Drivers for testing

**Advantages:**

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

**Disadvantages:**

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

**3. What is Alpha testing?**

* It is always performed by the developers at the software development site.
* Sometimes it is also performed by Independent Testing Team.
* Alpha Testing is not open to the market and public
* It is conducted for the software application and project.
* It is always performed in Virtual Environment.
* It is always performed within the organization.
* It is the form of Acceptance Testing.
* Alpha Testing is definitely performed and carried out at the developing organizations location with the involvement of developers.
* It comes under the category of both White Box Testing and Black Box Testing.

**4. What is beta testing?**

* Beta testing also known as Field testing.
* It is always performed by the customers at their own site.
* It is not performed by Independent Testing Team.
* Beta Testing is always open to the market and public.
* It is usually conducted for software product.
* It is performed in Real Time Environment.
* It is always performed outside the organization.
* It is also the form of Acceptance Testing.
* Beta Testing (field testing) is performed and carried out by users or you
* can say people at their own locations and site using customer data.
* It is only a kind of Black Box Testing.
* Beta Testing is always performed at the time when software product and project are marketed.
* It is always performed at the user’s premises in the absence of the development team.
* It is also considered as the User Acceptance Testing (UAT) which is done at customers or users area.
* Beta testing can be considered “pre-release” testing.
* Pilot Testing is testing to product on real world as well as collect data on the use of product in the classroom.

1. **What is component testing?**

* Component(Unit) – A minimal software item that can be tested in isolation. It means “A unit is the smallest testable part of software.”
* Component Testing – The testing of individual software components.
* Unit Testing is a level of the software testing process where individual units/components of a software/system are tested. The purpose is to validate that each unit of the software performs as designed.
* Unit testing is the first level of testing and is performed prior to Integration Testing.
* Sometimes known as Unit Testing, Module Testing or Program Testing
* Component can be tested in isolation – stubs/drivers may be employed
* Unit testing frameworks, drivers, stubs and mock or fake objects are used to assist in unit testing.
* Functional and Non-Functional testing
* Unit tests are typically written and run by software developers to ensure that code meets its design and behaves as intended with debugging tool.
* A unit is the smallest testable part of an application like functions/procedures, classes, interfaces.
* The goal of unit testing is to isolate each part of the program and show that the individual parts are correct.
* A unit test provides a strict, written contract that the piece of code must satisfy. As a result, it affords several benefits.
* Unit tests find problems early in the development cycle.
* Unit testing is performed by using the White Box Testing method.

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

* Non-Functional Testing: Testing the attributes of a component or system that do not relate to functionality, e.g. reliability, efficiency, usability, interoperability, maintainability and portability.
* Measuring the characteristics of the system/software that can be quantified on a varying scale- e.g. performance test scaling.
* Non-functional testing includes, but is not limited to, performance testing, load testing, stress testing, usability testing, maintainability testing, reliability testing and portability testing.
* It is the testing of “how” the system works. Non-functional testing may be performed at all test levels.
* The term non-functional testing describes the tests required to measure characteristics of systems and software that can be quantified on a varying scale, such as response times for performance testing.
* To address this issue, performance testing is carried out to check & fine tune system response times. The goal of performance testing is to reduce response time to an acceptable level.
* Hence load testing is carried out to check systems performance at different loads i.e. number of users accessing the system.

**Web Based Testing:**

* + Identify the software processes that directly influence the overall performance of the system.
  + In website number of user/customer will increase, how the website will handled to every customer/user.

**Desktop Based Testing:**

* Numerous other such GUI test cases, the desktop application tester must view
* Guarantee that error messages are instructive and helpful for the client
* Memory, and different other issues

**Mobile Based Testing:**

* In mobile, automatically will switch off without any reason.
* To stop the application which is not in our hand.

**Game Based Testing:**

* Confirms workability and stability of the software.
* Validate whether the user interface of the app is as per the screen size of the device and ensure high quality.

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

* White Box Testing: Testing based on an analysis of the internal structure of the component or system.
* Structure-based testing technique is also known as ‘white-box’ or ‘glass-box’ testing technique because here the testers require knowledge of how the software is implemented, how it works.
* In white-box testing the tester is concentrating on how the software does it.
  + For example, a structural technique may be concerned with exercising loops in the software.
* Different test cases may be derived to exercise the loop once, twice, and many times. This may be done regardless of the functionality of the software.
* Structure-based techniques are also used in system and acceptance testing, but the structures are different.
  + For example, the coverage of menu options or major business transactions could be the structural element in system or acceptance testing.
* Testing based upon the structure of the code
* Typically undertaken at Component and Component Integration Test phases by development teams.
* White box testing is the detailed investigation of internal logic and structure of the code.
* White box testing is also called glass testing or open box testing. In order to perform white box testing on an application, the tester needs to possess knowledge of the internal working of the code.
* The tester needs to have a look inside the source code and find out which unit/chunk of the code is behaving inappropriately.

Input output

* **Types of white box testing**
  + Statement coverage
  + Decision coverage
  + Condition coverage

1. **What is black box testing? What are the different black box testing techniques?**
2. Black-box testing: Testing, either functional or non-functional, without reference to the internal structure of the component or system.
3. Specification-based testing technique is also known as ‘black-box’ or input/output driven testing techniques because they view the software as a black-box with inputs and outputs.
4. The testers have no knowledge of how the system or component is structured inside the box. In black-box testing the tester is concentrating on what the software does, not how it does it.
5. Specification-based techniques are appropriate at all levels of testing (component testing through to acceptance testing) where a specification exists.
6. For example, when performing system or acceptance testing, the requirements specification or functional specification may form the basis of the tests.
7. The technique of testing without having any knowledge of the interior workings of the application is Black Box testing.
8. What a system does, rather than HOW it does it
9. Typically used at System Test phase, although can be useful throughout the test lifecycle
10. The tester is oblivious to the system architecture and does not have access to the source code.
11. Typically, when performing a black box test, a tester will interact with the system's user interface by providing inputs and examining outputs without knowing how and where the inputs are worked upon.

Input output

**Advantages**

* Well suited and efficient for large code segments.
* Code Access not required.
* Clearly separates user's perspective from the developer's perspective through visibly defined roles.
* Large numbers of moderately skilled testers can test the application with no knowledge of implementation, programming language or operating systems.

**Disadvantage**

* Limited Coverage since only a selected number of test scenarios are actually performed.
* Inefficient testing, due to the fact that the tester only has limited knowledge about an application.
* Blind Coverage, since the tester cannot target specific code segments or error prone areas.
* The test cases are difficult to design.

**Black Box Testing Examples**

**Web Based Testing:**

* Takes more time to execute as testers look for game play issues, graphicsc issues, audio-visual issues, etc.
* Validates whether installation goes smoothly, the app works in minimized mode, the app allows social networking options, supports payment gateways, and many more.
* Login by the user is must for accessing the sensitive information.

**Desktop Based Testing:**

* Resolution change effect on the application
* Installation Testing (Upgrade/Downgrade)

**Mobile Based Testing:**

* In mobile , automatically will switch off without any reason.
* To stop the application which is not in our hand.

**Game Based Testing:**

* The game tester must know how to play the game, utilization of the gamepad, know the game flow and the rules.

**Techniques of Black Box Testing**

* There are four specification-based or black-box technique:
* Equivalence partitioning
* Boundary value analysis
* Decision tables
* State transition testing
* Use-case Testing
* Other Black Box Testing
  + Syntax or Pattern Testing

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

Regressiontesting perform in following condition.

* Change in requirements and code is modified according to the requirement
* New feature is added to the software
* Defect fixing
* Performance issue fix

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

1. Testing shows presence of Defects
2. Exhaustive Testing is Impossible!
3. Early Testing
4. Defect Clustering
5. The Pesticide Paradox
6. Testing is Context Dependent
7. Absence of Errors Fallacy

**Testing shows presence of Defects**

* Testing can show that defects are present, but cannot prove that there are no defects.
* Testing reduces the probability of undiscovered defects remaining in the software but, even if no defects are found, it is not a proof of correctness.
* We test to find Faults
* As we find more defects, the probability of undiscovered defects remaining in a system reduces.
* However Testing cannot prove that there are no defects present

**Exhaustive Testing is Impossible!**

* Testing everything including all combinations of inputs and preconditions is not possible.
* So, instead of doing the exhaustive testing we can use risks and priorities to focus testing efforts.
* For example: In an application in one screen there are 15 input fields, each having 5 possible values, then to test all the valid combinations you would need 30 517 578 125 (515) tests.
* This is very unlikely that the project timescales would allow for this number of tests.
* So, accessing and managing risk is one of the most important activities and reason for testing in any project.
* We have learned that we cannot test everything (i.e. all combinations of inputs and pre-conditions).
* That is we must Prioritise our testing effort using a Risk Based Approach.

**Early Testing**

* Testing activities should start as early as possible in the software or system development life cycle, and should be focused on defined objectives.
* Testing activities should start as early as possible in the development life cycle
* These activities should be focused on defined objectives – outlined in the Test Strategy
* Remember from our Definition of Testing, that Testing doesn’t start once the code has been written!

**Defect Clustering**

* A small number of modules contain most of the defects discovered during pre-release testing, or are responsible for the most operational failures.
* Defects are not evenly spread in a system
* They are ‘clustered’
* In other words, most defects found during testing are usually confined to a small number of modules
* Similarly, most operational failures of a system are usually confined to a small number of modules
* An important consideration in test prioritisation!

**Pesticide Paradox**

* If the same tests are repeated over and over again, eventually the same set of test cases will no longer find any new defects.
* To overcome this “pesticide paradox”, the test cases need to be regularly reviewed and revised, and new and different tests need to be written to exercise different parts of the software or system to potentially find more defects.
* Testing identifies bugs, and programmers respond to fix them
* As bugs are eliminated by the programmers, the software improves
* As software improves the effectiveness of previous tests erodes
* Therefore we must learn, create and use new tests based on new techniques to catch new bugs
* N.B It's called the "pesticide paradox" after the agricultural phenomenon, where bugs such as the boll weevil build up tolerance to pesticides, leaving you with the choice of ever-more powerful pesticides followed by ever-more powerful bugs or an altogether different approach.’ – Beizer 1995

**Testing is Context Dependent**

* Testing is basically context dependent.
* Testing is done differently in different contexts
* Different kinds of sites are tested differently.
* For example
  + Safety – critical software is tested differently from an e-commerce site.
* Whilst, Testing can be 50% of development costs, in NASA's Apollo program it was 80% testing
* 3 to 10 failures per thousand lines of code (KLOC) typical for commercial software
* 1 to 3 failures per KLOC typical for industrial software
* 0.01 failures per KLOC for NASA Shuttle code!
* Also different industries impose different testing standard

**Absence of Errors Fallacy**

* If the system built is unusable and does not fulfill the user’s needs and expectations then finding and fixing defects does not help.
* If we build a system and, in doing so, find and fix defects ....
* It doesn’t make it a good system
* Even after defects have been resolved it may still be unusable and/or does not fulfil the users’ needs and expectations

**11. Difference between QA v/s QC v/s Tester**

|  |  |  |  |
| --- | --- | --- | --- |
| **S.N.** | **Quality Assurance** | **Quality Control** | **Testing** |
| **1.** | Activities which ensure the implementation of processes, procedures and standards in context to verification of developed software and intended requirements. | Activities which ensure the verification of developed software with respect to documented (or not in some cases) requirements. | Activities which ensure the identification of bugs/error/defects in the Software. |
| **2.** | Focuses on processes and procedures rather than conducting actual testing on the system. | Focuses on actual testing by executing Software with intend to identify bug/defect through implementation of procedures and process. | Focuses on actual testing. |
| **3.** | Process oriented activities. | Product oriented activities. | Product oriented activities. |
| **4.** | Preventive activities. | It is a corrective process. | It is a preventive process. |
| **5.** | It is a subset of Software Test Life Cycle (STLC). | QC can be considered as the subset of Quality Assurance. | Testing is the subset of Quality Control. |

**12. Difference between Smoke and Sanity?**

|  |  |  |
| --- | --- | --- |
| **S.N.** | **Smoke Testing** | **Sanity Testing** |
| **1.** | Smoke Testing is performed to ascertain that the critical functionalities of the program is working fine | Sanity Testing is done to check the new functionality/bugs have been fixed |
| **2.** | The objective of this testing is to verify the “stability” of the system in order to proceed with more rigorous testing | The objective of the testing is to verify the “rationality” of the system in order to proceed with more rigorous testing |
| **3.** | This testing is performed by the developers or testers | Sanity testing in software testing is usually performed by testers |
| **4.** | Smoke testing is usually documented or scripted | Sanity testing is usually not documented and is unscripted |
| **5.** | Smoke testing is a subset of Acceptance testing | Sanity testing is a subset of [Regression Testing](https://www.guru99.com/regression-testing.html) |

**13. Difference between verification and Validation**

|  |  |  |
| --- | --- | --- |
| **Criteria** | **Verification** | **Validation** |
| Definition | The process of evaluating work-products (not the actual final product) of a development phase to determine whether they meet the specified requirements for that phase. | The process of evaluating software during or at the end of the development process to determine whether it satisfies specified business requirements. |
| Objective | To ensure that the product is being built according to the requirements and design specifications. In other words, to ensure that work products meet their specified requirements. | To ensure that the product actually meets the user’s needs, and that the specifications were correct in the first place. In other words, to demonstrate that the product fulfills its intended use when placed in its intended environment. |
| Question | Are we building the product right? | Are we building the right product? |
| Evaluation Items | Plans, Requirement Specs, Design Specs, Code, Test Cases | The actual product/software. |
| Activities | * Reviews * Walkthroughs * Inspections | * Testing |

**14. What is Error, Defect, Bug and failure?**

“A mistake in coding is called error, error found by tester is

called defect, defect accepted by development team then it is

called bug, build does not meet the requirements then it is

failure”

**Error**: A discrepancy between a computed, observed, or measured value or condition and the true, specified, or theoretically correct value or condition. This can be a misunderstanding of the internal state of the software, an oversight in terms of memory management, confusion about the proper way to calculate a value, etc.

**Defect**: Commonly refers to several troubles with the software

products, with its external behavior or with its internal features.

**Bug**: A fault in a program which causes the program to perform in an unintended or unanticipated manner. See: anomaly, defect, error, exception, and fault. Bug is terminology of Tester.

**Failure**: The inability of a system or component to perform its

required functions within specified performance requirements. See: bug, crash, exception, and fault.

**15. Explain the difference between Functional testing and Non-Functional testing.**

|  |  |  |
| --- | --- | --- |
| **S.N.** | **Functional Testing** | **Non-Functional Testing** |
| **1.** | Functional testing is performed using the functional specification provided by the client and verifies the system against the functional requirements. | Non-Functional testingchecksthe Performance, reliability, scalability and other non-functional aspects of the software system. |
| **2.** | Functional testing is executed first | Non-functional testing should be performed after functional testing. |
| **3.** | Manual testing or automation tools can be used for functional testing | Using tools will be effective for this testing. |
| **4.** | Business requirements are the inputs to functional testing | Performance parameters like speed , scalability are inputs to non-functional testing. |
| **5.** | Functional testing describes what the product does | Nonfunctional testing describes how good the product works |
| **6.** | Types of Functional testing are   * Unit Testing * Smoke Testing * Sanity Testing * Integration Testing * White box testing * Black Box testing * User Acceptance testing * Regression Testing | Types of Nonfunctional testing are   * Performance Testing * Load Testing * Volume Testing * Stress Testing * Security Testing * Installation Testing * Penetration Testing * Compatibility Testing * Migration Testing |

**16. What is the difference between the STLC (Software Testing Life Cycle) and SDLC (Software Development Life Cycle)?**

|  |  |  |
| --- | --- | --- |
| **S.N.** | **SDLC (Software Development Life Cycle)** | **STLC (Software Testing Life Cycle)** |
| **1.** | Development Life Cycle | Testing Life Cycle |
| **2.** | The main object of SDLC life cycle is to complete successful development of the software including testing and other phases. | The only objective of the STLC phase is testing. |
| **3.** | In SDLC the business analyst gathers the requirements and create Development Plan | In STLC, the QA team analyze requirement documents like functional and non-functional documents and create System Test Plan |
| **4.** | In SDLC, the development team creates the high and low-level design plans | In STLC, the test analyst creates the Integration Test Plan |
| **5.** | The real code is developed, and actual work takes place as per the design documents. | The testing team prepares the test environment and executes them |
| **6.** | SDLC phase also includes post-deployment supports and updates. | Testers, execute regression suits, usually automation scripts to check maintenance code deployed. |

**17. What is the difference between test scenarios, test cases, and test script?**

**Test Scenarios:** A Test Scenario is any functionality that can be tested. It is also called Test Condition or Test Possibility.

**Test Cases:** It is a document that contains the steps that has to be executed, it has been planned earlier.

**Test Script:** It is written in a programming language and it's a short program used to test part of functionality of the software system. In other words a written set of steps that should be performed manually.

1. **Explain what Test Plan is? What is the information that should be covered What is priority?**

* A document describing the scope, approach, resources and schedule of intended test activities
* Determining the scope and risks, and identifying the objectives of testing.
* Defining the overall approach of testing (the test strategy), including the definition of the test levels and entry and exit criteria.
* Integrating and coordinating the testing activities into the software life cycle activities:
* acquisition, supply, development, operation and maintenance.
* Making decisions about what to test, what roles will perform the test activities, how the test activities should be done, and how the test results will be evaluated?
* Scheduling test analysis and design activities.
* Scheduling test implementation, execution and evaluation.
* Assigning resources for the different activities defined
* Defining the amount, level of detail, structure and templates for the test documentation.

**Test Planning Factors**

* Factors which affect test planning
* The organisation’s test policy
* Scope of the testing being performed
* Testing objectives
* Project Risks – e.g. business, technical, people
* Constraints – e.g. business imposed, financial, contractual etc
* Criticality (e.g. system/component level)
* Testability
* Availability of resources
* Test plans are continuously refine
* As more information becomes available
* As new risks arise or others are mitigated
* Not set in concrete, but changes must be carefully managed

**Test Planning Activities**

* Approach: Defining the overall approach of testing (the test strategy), including the definition of the test levels and entry and exit criteria.
* Integrating and coordinating the testing activities into the software life cycle activities: acquisition, supply, development, operation and maintenance.
* Making decisions about:
  + what to test
  + who do testing? i.e. what roles will perform the test activities
  + when and how the test activities should be done and when they should be stopped (exit criteria – see next slides)
  + how the test results will be evaluated.
  + Assigning resources for the different tasks defined.
  + Test ware: Defining the amount, level of detail, structure and templates for the test documentation.
  + Selecting metrics for monitoring and controlling test preparation and execution, defect resolution and risk issues.
  + Process: Setting the level of detail for test procedures in order to provide enough information to support reproducible test preparation and execution.

1. **What is Exploratory Testing?**

Exploratory testing is an approach to software testing that emphasizes the tester's freedom and creativity in designing and executing test cases. Unlike traditional scripted testing, where test cases are predefined and executed in a planned manner, exploratory testing involves simultaneous learning, test design, and test execution.

In exploratory testing, the tester explores the software system in an unstructured and ad-hoc manner, actively learning about its features, behavior, and potential issues while conducting tests. The tester relies on their domain knowledge, experience, and intuition to identify test scenarios, execute test cases, and evaluate the software's behavior. The goal is to uncover defects, risks, and usability issues that might not be easily identified through scripted testing.

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

Boundary Value Analysis is a software testing technique used to identify errors at the boundaries of input values. It focuses on testing the boundary conditions, which are the minimum and maximum valid values as well as values just above and below those boundaries. By testing these critical points, it aims to expose any defects or issues that may arise due to boundary-related problems.

Let's consider an example scenario to illustrate Boundary Value Analysis:

Suppose you are developing a registration form for a website, and one of the input fields is "Age." The requirements specify that the age must be between 18 and 60 years.

Test Case 1: Minimum Valid Value (Boundary Test)

Input: Age = 18

Expected Result: The registration should be successful.

Test Case 2: Value Just Below the Minimum (Boundary Test)

Input: Age = 17

Expected Result: The registration should fail, displaying an error message indicating that the age is not valid.

Test Case 3: Maximum Valid Value (Boundary Test)

Input: Age = 60

Expected Result: The registration should be successful.

Test Case 4: Value Just Above the Maximum (Boundary Test)

Input: Age = 61

Expected Result: The registration should fail, displaying an error message indicating that the age is not valid.

Test Case 5: Typical Value within the Range

Input: Age = 30

Expected Result: The registration should be successful.

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

Equivalence Partitioning is a software testing technique used to divide the input data of a system into groups or partitions based on their behavior and characteristics. The idea behind equivalence partitioning is to select representative test cases from each partition that have similar behavior, with the assumption that if a test case from a partition uncovers a defect, it is likely that other test cases in the same partition will also uncover defects.

The main objective of equivalence partitioning is to reduce the total number of test cases while still maintaining reasonable test coverage. By selecting representative test cases from each partition, it is possible to achieve good test coverage without exhaustively testing every possible input combination.

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

Risk – ‘A factor that could result in future negative

consequences; usually expressed as impact and likelihood’

Risks are of two types

* Project Risks
* Product Risk

1. **What is functional system testing?**
   * Functional System Testing: A requirement that specifies a function that a system or system component must perform
   * A Requirement may exist as a text document and/or a model
   * There is two types of Test Approach

* Requirement Based Functional Testing
* Process Based Testing
* Functional System Testing Functionality As below:

|  |  |
| --- | --- |
| Accuracy | Provision of right or agreed results or effects |
| Interoperability | Ability to interact with specified systems |
| Compliance | Adhere to applicable standards, conventions, regulations or laws |
| Auditability | Ability to provide adequate and accurate audit data |
| Suitability | Presence and appropriateness of functions for specified tasks |

1. **What is GUI Testing?**

Graphical User Interface (GUI) testing is the process of testing the system’s GUI of the System under Test. GUI testing involves checking the screens with the controls like menus, buttons, icons, and all types of bars – tool bar, menu bar, dialog boxes and windows etc.

* We can check following things in GUI testing
* Check all the GUI elements for size, position, width, length and acceptance of characters or numbers. For instance, you must be able to provide inputs to the input fields.
* Check you can execute the intended functionality of the application using the GUI
* Check Error Messages are displayed correctly
* Check for Clear demarcation of different sections on screen
* Check Font used in application is readable
* Check the alignment of the text is proper
* Check the Color of the font and warning messages is aesthetically pleasing
* Check that the images have good clarity
* Check that the images are properly aligned
* Check the positioning of GUI elements for different screen resolution.

1. **What is Adhoc testing?**

* Adhoc testing is an informal testing type with an aim to break the system.
* It does not follow any test design techniques to create test cases.
* In fact is does not create test cases altogether!
* This testing is primarily performed if the knowledge of testers in the system under test is very high.
* Testers randomly test the application without any test cases or any business requirement document.
* Adhoc 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 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.
* The Error guessing is a technique where the experienced and good testers are encouraged to think of situations in which the software may not be able to cope.
* Some people seem to be naturally good at testing and others are good testers because they have a lot of experience either as a tester or working with a particular system and so are able to find out its weaknesses.
* This is why an error guessing approach, used after more formal techniques have been applied to some extent, can be very effective.
* It also saves a lot of time because of the assumptions and guessing made by the experienced testers to find out the defects which otherwise won’t be able to find.
* Using experience to postulate errors.
* Use Error Guessing to Complement Test Design Techniques.
* There are different types of Adhoc testing and they are listed as below:
  + Buddy Testing
  + Pair testing
  + Monkey Testing

1. **What is load testing?**

* Load testing - Its a performance testing to check system behavior under load. Testing an application under heavy loads, such as testing of a web site under a range of loads to determine at what point the system’s response time degrades or fails.
* Load testing is a kind of performance testing which determines a system’s performance under real-life load conditions. This testing helps determine how the application behaves when multiple users access it simultaneously.
* This testing usually identifies –
* The maximum operating capacity of an application
* Determine whether current infrastructure is sufficient to run the application
* Sustainability of application with respect to peak user load
* Number of concurrent users that an application can support, and scalability to allow more users to access it.
* It is a type of non-functional testing. Load testing is commonly used for the Client/Server, Web based applications – both Intranet and Internet.

1. **What is stress Testing?**

* Stress testing - System is stressed beyond its specifications to check how and when it fails. Performed under heavy load like putting large number beyond storage capacity, complex database queries, continuous input to system or database load.
* Stress testing is used to test the stability & reliability of the system. This test mainly determines the system on its robustness and error handling under extremely heavy load conditions.
* It even tests beyond the normal operating point and evaluates how the system works under those extreme conditions.
* Stress Testing is done to make sure that the system would not crash under crunch situations.
* Stress testing is also known as endurance testing.
* Under Stress Testing, AUT is be stressed for a short period of time to know its withstanding capacity.
* Most prominent use of stress testing is to determine the limit, at which the system or software or hardware breaks.
* It also checks whether system demonstrates effective error management under extreme conditions.
* The application under testing will be stressed when 5GB data is copied from the website and pasted in notepad.
* Notepad is under stress and gives ‘Not Responded’ error message.
* Examples of stress conditions include:
  + Excessive volume in terms of either users or data; examples might include a denial of service (DoS) attack or a situation where a widely viewed news item prompts a large number of users to visit a Web site during a three-minute period.
  + Resource reduction such as a disk drive failure.
  + Application components fail to respond.

1. **Mention what are the categories of defects?**

Types of Defect

* Data Quality/Database Defects: Deals with improper handling of data in the database.
* Examples:
  + Values not deleted/inserted into the database properly
  + Improper/wrong/null values inserted in place of the actual values
  + Critical Functionality Defects: The occurrence of these bugs hampers the crucial functionality of the application. Examples: - Exceptions
* **Functionality Defects**: These defects affect the functionality of the application.
* **Examples:**
  + All JavaScript errors
  + Buttons like Save, Delete, Cancel not performing their intended functions
  + A missing functionality (or) a feature not functioning the way it is intended to
  + Continuous execution of loops
  + Security Defects: Application security defects generally involve improper handling of data sent from the user to the application. These defects are the most severe and given highest priority for a fix.
  + **Examples:**
    - Authentication: Accepting an invalid username/password
    - Authorization: Accessibility to pages though permission not given
    - User Interface Defects: As the name suggests, the bugs deal with problems related to UI are usually considered less severe.
    - **Examples**:
      * Improper error/warning/UI messages
      * Spelling mistakes
      * Alignment problems

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

* In Big Bang integration testing all components or modules is integrated simultaneously, after which everything is tested as a whole.
* Big Bang testing has the advantage that everything is finished before integration testing starts.
* The major disadvantage is that in general it is time consuming and difficult to trace the cause of failures because of this late integration.
* Here all component are integrated together at once, and then tested.

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

* Purpose of exit criteria is to define when we STOP testing either at the:
  + End of all testing – i.e. product Go Live
  + End of phase of testing (e.g. hand over from System Test to UAT)

1. **Explain types of Performance testing.**
2. Load testing
3. Stress testing
4. Endurance testing
5. Spike testing
6. Volume testing
7. Scalability testing
8. **Difference between Priority and Severity**

|  |  |  |
| --- | --- | --- |
|  | **Priority** | **Severity** |
| Definition | The relative importance or urgency of addressing an issue | The impact or seriousness of an issue or its consequences |
| Focus | Determines the order in which issues should be addressed | Evaluates the impact of an issue on the system or users |
| Purpose | Helps prioritize tasks and allocate resources | Guides decision-making and risk assessment |
| Factors | Business requirements, customer impact, deadlines | Technical impact, system stability, potential harm |
| Examples | Low, medium, high, critical | Minor, major, critical, catastrophic |

1. **What is Bug Life Cycle?**

“A computer bug is an error, flaw, mistake, failure, or fault in a computer program that prevents it from working correctly or produces an incorrect result. Bugs arise from mistakes and errors, made by people, in either a program’s source code or its design.”

1. **What is priority?**

Priority is Relative and Business-Focused. Priority defines the order in which we should resolve a defect. Should we fix it now, or can it wait? This priority status is set by the tester to the developer mentioning the time frame to fix the defect. If high priority is mentioned then the developer has to fix it at the earliest. The priority status is set based on the customer requirements.

**For example**: If the company name is misspelled in the home page of the website, then the priority is high and severity is low to fix it.

1. **What is severity?**

Severity is absolute and Customer-Focused. It is the extent to

which the defect can affect the software. In other words it defines the impact that a given defect has on the system.

**For example:** If an application or web page crashes when a remote link is clicked, in this case clicking the remote link by an user is rare but the impact of application crashing is severe. So the severity is high but priority is low.

1. **Advantage of Bugzila.**

Bugzilla is a widely used open-source bug tracking system that offers several advantages for software development teams and organizations. Here are some of the key advantages of Bugzilla:

* Bug Tracking and Management: Bugzilla provides a comprehensive platform for tracking and managing software issues, bugs, and enhancement requests. It allows users to submit bug reports, track their progress, assign them to developers, and monitor their resolution status.
* Centralized System: Bugzilla serves as a centralized repository for all bug-related information, making it easier for team members to access and collaborate on bug reports. It provides a unified platform for communication, ensuring that everyone involved in the development process has access to the latest information.
* Customizable Workflow: Bugzilla offers a highly customizable workflow that can be tailored to fit the specific needs of different development teams or organizations. It allows you to define your own bug states, priorities, severities, and other parameters, enabling you to adapt the bug management process to match your development methodology.
* Collaboration and Communication: Bugzilla facilitates collaboration and communication among team members. It provides features such as commenting, attachments, and email notifications, which allow developers, testers, and stakeholders to discuss and share information about specific bugs. This improves transparency and ensures that everyone is on the same page.
* Extensive Search and Reporting: Bugzilla includes powerful search capabilities that enable users to quickly find specific bugs or generate custom reports based on various criteria. This can help in identifying trends, analyzing bug patterns, and making data-driven decisions to improve the software development process.

1. **What are the different Methodologies in Agile Development Model?**

**Scrum**

* SCRUM is an agile development method which concentrates particularly on how to manage tasks within a team based development environment.
* Basically, Scrum is derived from activity that occurs during rugby match. Scrum believes in empowering the development team and advocates working in small teams (say- 7 to 9 members).
* It consists of three roles and their responsibilities are explained as follows:
  + **Scrum Master**: Master is responsible for setting up the team, sprint meeting and removes obstacles to progress
  + **Product owner**: The Product Owner creates product backlog, prioritizes the backlog and is responsible for the delivery of the functionality at each iteration
  + **Scrum Team**: Team manages its own work and organizes the work to complete the sprint or cycle

**Kanban**

* Kanban is a very popular framework for development in the agile software development methodology.
* It provides a transparent way of visualizing the tasks and work capacity of a team.
* It mainly uses physical and digital boards to allow the team members to visualize the current state of the project they are working on.
* Kanban originated in Toyota in the 1940s.
* Kanban’s meaning in Japanese is “billboards.”
* The Kanban board has columns and story cards.
* The columns are nothing, but workflow states and cards are nothing but a demonstration of the actual task a team member is performing.

1. **When to used Usablity Testing?**

In following scenario usability testing use.

* Product development: Usability testing is typically conducted during the development process to ensure that a product or system meets the needs of its intended users. It helps uncover design flaws, navigation issues, and other usability problems that may hinder user satisfaction and productivity.
* Prototype evaluation: Usability testing is valuable for evaluating prototypes at different stages of the design process. It allows designers to gather user feedback and validate their design decisions before investing further resources in development.
* Website and application design: Usability testing is often used to evaluate websites, mobile applications, and software interfaces. It helps determine whether users can easily navigate, understand the layout, complete tasks efficiently, and find the information they need.
* Redesign or improvement projects: When redesigning an existing product or system, usability testing can be employed to assess the effectiveness of the changes. It helps validate whether the redesign has addressed the previous usability issues or introduced new problems.
* Comparative analysis: Usability testing can be used to compare different versions or variations of a product or system. By testing multiple designs or concepts, it helps identify the most user-friendly option and informs design decisions.
* Accessibility evaluation: Usability testing can also be used to evaluate the accessibility of a product or system for users with disabilities. It helps uncover barriers that may prevent users with visual, auditory, motor, or cognitive impairments from effectively using the product.

1. **What is the procedure for GUI Testing?**

The procedure for GUI testing typically involves the following steps

* Understand the Requirements: Thoroughly analyze the software requirements and gain a clear understanding of the expected behavior and features of the GUI.
* Identify Test Scenarios: Based on the requirements, create a list of test scenarios that cover the various functionalities and interactions of the GUI. This includes testing different GUI components, navigation, user inputs, and error handling.
* Develop Test Cases: For each test scenario, design specific test cases that outline the steps to be performed, expected results, and any preconditions or test data required.
* Set Up the Test Environment: Prepare the test environment by installing the necessary hardware, software, and tools. This includes configuring the operating system, installing the application, and ensuring that any required test data or sample databases are available.
* Execute Test Cases: Run the test cases by interacting with the application's GUI. Perform actions such as clicking buttons, entering data into input fields, selecting menu options, and navigating through different screens or windows.
* Verify Expected Behavior: During test execution, compare the actual behavior of the GUI with the expected results specified in the test cases. Validate that the GUI elements function correctly, respond appropriately to user actions, and display accurate information.
* Validate Visual Appearance: Assess the visual aspects of the GUI, including layout, alignment, font styles, colors, and graphics. Verify that the GUI adheres to design standards, is visually appealing, and provides a consistent user experience.