1. What is Exploratory Testing?

-> Exploratory testing is a concurrent process where

Test design, execution and logging happen simultaneously Testing is often not recorded

Makes use of experience, heuristics and test patterns Testing is based on a test charter that may include

Scope of the testing (in and out)

The focus of exploratory testing is more on testing as a “thinking” activity.

A brief description of how tests will be performed Expected problems

Is carried out in time boxed intervals

More structured than Error guessing

1. What is traceability matrix?

-> 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.

Type of traceability matrix:-

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

1. What is Boundary value testing?

-> Boundary value analysis is a methodology for designing test cases that concentrates software testing effort on cases near the limits of valid ranges

Boundary value analysis is a method which refines equivalence partitioning.

Boundary value analysis generates test cases that highlight errors better than equivalence partitioning.

The trick is to concentrate software testing efforts at the extreme ends of the equivalence classes.

At those points when input values change from valid to invalid errors are most likely to occur.

Boundary Value Analysis (BVA) uses the same analysis of partitions as EP and is usually used in conjunction with EP in test case design

1. What is Equivalence partitioning testing?

-> Equivalence Partitioning (EP) is a Black Box Testing technique that divides input data into partitions of valid and invalid values, called equivalence classes, to reduce the number of test cases while still covering a wide range of inputs.

Main Idea:

If one value in a group (partition) works correctly, all other values in that group are assumed to behave the same.

Example:

Suppose a form accepts age from 18 to 60.

You can divide input into 3 partitions:

1. Valid partition: 18–60 (e.g., test with 30)
2. Invalid partition (below range): Less than 18 (e.g., test with 15)
3. Invalid partition (above range): Greater than 60 (e.g., test with 65)

Instead of testing every value, test one value from each partition.

Benefits:

* Reduces total test cases.
* Improves test coverage efficiently.
* Identifies invalid input handling.

Limitations:

* Assumes system behaves the same for all values in a partition (may not always be true).
* Doesn’t test boundary values precisely (handled better with Boundary Value Analysis).

Often used with:

* Boundary Value Analysis (BVA) for better coverage of edge cases.

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.

1. What determines the level of risk?

->

1. Probability (Likelihood)

* This considers the chance that something might go wrong.
* Influenced by:
  + Code complexity
  + New or untested features
  + Past defect rates
  + Changes in critical areas
  + Inexperienced developers or testers

1. Impact (Severity / Consequence)

What is the potential damage if the defect does occur?

* This considers how serious the problem would be.
* Influenced by:
  + Business importance of the feature
  + Financial loss
  + Safety or legal implications
  + Effect on customers or system performance

Risk Level = Probability × Impact

| Probability | Impact | Risk Level |
| --- | --- | --- |
| Low | Low | Low |
| Low | High | Medium |
| High | Low | Medium |
| High | High | High |

1. 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.

1. What is beta 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.

1. What is component testing?

-> 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 tools

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 Auditability Suitability 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 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

May be performed at all Test levels (not just Non Functional Systems Testing)

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

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.

WHAT DO YOU CHECK 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.

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

-> White Box Testing (also known as Clear Box, Glass Box, or Structural Testing) is a software testing technique where the internal structure, logic, and code of the program are tested. In this type of testing, the tester has full knowledge of the source code.

There are type of White box testing:

* Unit Testing
* Statement coverage Testing
* Branch/Decision Coverage
* Condition Coverage
* Path Coverage
* Loop Testing
* Data Flow Testing
* Mutation Testing

1. What is black box testing? What are the different black box testing techniques?

-> Black Box Testing is a software testing method where the tester does not need to know the internal code, structure, or logic of the application. The focus is purely on inputs and outputs testing whether the software behaves as expected.

Different Black Box Testing Techniques:

Here are the most common techniques used:

1. Equivalence Partitioning

* Divides input into valid and invalid groups (partitions).
* Test only one value from each group.

1. Boundary Value Analysis (BVA)

* Focuses on testing the edges or boundaries of input ranges.
* Errors often occur at boundaries.

1. Decision Table Testing

* Used when the system has complex business rules or logic.
* Represent conditions and outcomes in a table format.

1. State Transition Testing

* Tests the system's behavior based on different states and events.
* Useful for applications like login, traffic lights, etc.

1. Error Guessing

* Based on the tester’s experience and intuition to guess where bugs may exist.
* No formal technique, but effective for finding hidden bugs.

1. Use Case Testing

* Based on real-world user scenarios or workflows.
* Ensures the system works for practical usage paths.

1. Mention what are the categories of defects?

-> Defects (also called bugs or issues) in software can be categorized in several ways depending on their nature, impact, cause, or area of the system they affect.

Here are the most commonly used categories of defects:

1. Functional Defects

* Defects where the system does not behave as expected based on the requirements.

2. Performance Defects

* Issues related to speed, scalability, or responsiveness.

3. Usability Defects

* Issues that affect user experience, ease of use, or accessibility.

4. Compatibility Defects

* Problems that occur when the software doesn’t work correctly on different devices, OS, browsers, etc.

5. Security Defects

* Vulnerabilities that may lead to unauthorized access, data leaks, or breaches.

6. Logical Defects

* Errors in the business logic or algorithms implemented.

7. Boundary Related Defects

* Issues occurring at the edges of input ranges.

8. Data Defects

* Involve incorrect, missing, or corrupted data.

9. Integration Defects

* Issues found when modules or systems interact.

10. Installation and Configuration Defects

* Problems during software setup, deployment, or environment configuration.

11. Regression Defects

* Bugs that reappear after a change or fix in the code.

1. Mention what big bang 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 components are integrated 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

1. What is the purpose of exit criteria?

-> Exit Criteria are a set of predefined conditions that must be met to officially end a particular testing phase (like Unit Testing, System Testing, or UAT).

They help ensure that testing is complete, the software is stable, and it's ready to move to the next phase

1. When should "Regression Testing" be performed?

-> Regression Testing should be performed whenever the software undergoes a change, to ensure that existing functionality still works correctly and no new bugs have been introduced into previously tested code.

when the system is stable and the system or the environment changes

when testing bug-fix releases as part of the maintenance phase It should be applied at all Test Levels

It should be considered complete when agreed completion criteria for regression testing have been met

Regression test suites evolve over time and given that they are run frequently are ideal candidates for automation

1. What is 7 key principles? Explain in detail?

-> Here are 7 key principles:

1. 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

2. 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.

3. 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

4. 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 prioritization!

5. The 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

6. Testing is Context Dependent:

* Testing is 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% of 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 standards

7. Absence of Errors Fallacy

* If the system built is unusable and does not full fill 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

1. 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 software with developed 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. |

1. Difference between Smoke and Sanity?

->

|  |  |
| --- | --- |
| Smoke Testing | Sanity Testing |
| Smoke Testing is performed to ascertain that the critical functionalities of the program is working fine. | Sanity Testing is done to check the new functionality / bugs have been fixed. |
| The objective of this testing is to verify "stability" of the system in order to the rigorous testing. | The objective of the testing is to verify the "rationality" of the system in order proceed with more to proceed with more rigorous testing. |
| This testing is performed by the developers or testers | Sanity testing is usually performed by testers |
| Smoke testing is usually documented or scripted. | Sanity testing is usually not documented and is unscripted. |
| Smoke testing is subset of Acceptance testing. | Sanity testing is subset of Regression testing. |
| Smoke testing exercises the entire system from end to end. | Sanity testing exercises only the particular component of the entire system. |
| Smoke testing is like General Health Check Up. | Sanity Testing is like specialized health checkup. |

1. 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 full fills 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 |

1. Explain types of Performance testing.

->

* Load testing: checks the application’s ability to perform under anticipated user loads. The objective is to identify performance bottlenecks before the software application goes live.
* Stress testing: involves testing an application under extreme workloads to see how it handles high traffic or data processing. The objective is to identify the breaking point of an application.
* Endurance testing: is done to make sure the software can handle the expected load over a long period of time.
* Spiketesting: tests the software’s reaction to sudden large spikes in the load generated by users.
* Volume testing: Under Volume Testing large no. of. Data is populated in a database, and the overall software system’s behavior is monitored. The objective is to check software application’s performance under varying database volumes.
* Scalability testing: The objective of scalability testing is to determine the software application’s effectiveness in “scaling up” to support an increase in user load. It helps plan capacity addition to your software system.

1. What is Error, Defect, Bug and 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

1. Difference between Priority and Severity

->

| Aspect | Severity | Priority |
| --- | --- | --- |
| Meaning | How serious the bug is from a technical or business perspective | How urgently the bug should be fixed or addressed |
| Focus On | Impact of the bug on the system | Order in which the bug should be fixed |
| Set By | Usually set by Testers or QA | Usually set by Project Manager/Product Owner/Client |

1. What is Bug Life Cycle?

->

1. Explain the difference between Functional testing and Non Functional testing

->

|  |  |
| --- | --- |
| Functional testing | Non-Functional testing |
| Functional testing is performed using the Non-Functional functional specification provided by the client and verifies the system against the functional requirements. | Testing checks the Performance, reliability, scalability and other non-functional aspects of the software system. |
| Functional testing is executed first | Non functional testing should be performed after functional testing. |
| Manual testing or automation tools can be used for functional testing | Using tools will be effective for this testing |
| Business requirements are the inputs to functional testing | Performance parameters like speed , scalability are inputs to non-functional testing. |
| Functional testing describes what the product does | Nonfunctional testing describes how good the product works |
| Easy to do manual testing | Tough to do manual testing |
| 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 209  ∙ Migration Testing |

1. To create HLR & Test Case of (Instagram , Facebook) first page and chat functionality

-> Instagram: <https://docs.google.com/spreadsheets/d/1g5GUgXMtsHriSss5oYkgKOY5vArdD0rY7_sZ_LHQRt4/edit?usp=sharing>

Facebook:

https://docs.google.com/spreadsheets/d/1bBxbuuK2DL3ctNfzJ-\_CIoMBqLF\_mFziVkkjHk2iaiI/edit?usp=sharing

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

->

| Feature | SDLC | STLC |
| --- | --- | --- |
| Full Form | Software Development Life Cycle | Software Testing Life Cycle |
| Purpose | To develop a complete software product | To ensure the product is tested and meets quality standards |
| Focus Area | Requirement analysis to maintenance | Test planning to test closure |
| Participants | Business Analysts, Developers, Project Managers, Testers | Testers, QA Leads, Test Managers |
| Involves | Design, coding, integration, deployment, support | Test planning, design, execution, bug tracking |
| Starts When | When the project starts | After requirements are defined and development begins |
| Output | Fully developed, working software | Verified and validated software (bug report, test report) |
| Documents Used | SRS, Design Docs, Architecture Docs | Test Plan, Test Case Docs, Bug Reports |

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

->

| Feature | Test Scenario | Test Case | Test Script |
| --- | --- | --- | --- |
| Focus | What to test | How to test a specific function | Automate/execute the test case |
| Level | High-level | Medium-level | Low-level (step-by-step or c ode) |
| Type | Conceptual | Documented (manual/automated) | Mostly automated (or manual instructions) |
| Used by | Test designers, stakeholders | Testers | QA engineers, automation testers |
| Reusability | Yes (for creating multiple test cases) | Yes (linked to scenarios) | Yes (saves time in regression testing) |

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

-> A Test Plan is a formal document that outlines the strategy, objectives, resources, schedule, scope, approach, and activities for testing a software application or system. It serves as a blueprint for the entire testing process and ensures all stakeholders are aligned on how testing will be conducted.

Information should be covered:

| **Section** | **Description** |
| --- | --- |
| **1. Test Plan ID** | A unique identifier for the test plan (e.g., TP-01) |
| **2. Introduction** | Overview of the project, product, and objectives of the test plan |
| **3. Scope of Testing** | What will be tested (in-scope) and what will not be tested (out-of-scope) |
| **4. Test Objectives** | The goals of testing, e.g., verify login, validate data accuracy |
| **5. Test Strategy** | Overall approach: manual vs. automation, types of testing (functional, regression, etc.) |
| **6. Test Criteria** | - **Entry Criteria**: Conditions to begin testing- **Exit Criteria**: Conditions to stop testing |
| **7. Test Deliverables** | List of documents and outcomes like test cases, test data, defect reports, test summary reports |
| **8. Schedule & Milestones** | Testing timelines, phases, and key deadlines |
| **9. Resource Planning** | Team members involved, their roles, and responsibilities |
| **10. Environment Requirements** | Hardware, software, test tools, and setup needed for testing |
| **11. Risk & Mitigation Plan** | Possible risks (e.g., resource shortage, unstable builds) and mitigation strategies |
| **12. Test Tools** | Any tools used (e.g., Selenium, JIRA, TestRail) |
| **13. Approvals** | Sign-off by test manager, QA lead, or project stakeholders |

1. What is priority?

-> In software testing, Priority refers to the urgency or importance of fixing a defect or implementing a test task. It tells the developer or project team how quickly a bug should be resolved based on business or customer needs, rather than technical severity.

1. What is severity?

-> In software testing, Severity refers to the impact a defect has on the functionality or performance of the application. It indicates how serious the issue is from a technical or system perspective, regardless of how quickly it needs to be fixed.

1. Bug categories are…

->

* Functional Bugs
* UI/UX Bugs
* Performance Bugs
* Security Bugs
* Compatibility Bugs
* Usability Bugs
* Data Integrity Bugs
* Boundary/Validation Bugs
* Crash/Blocking Bugs
* Configuration Bugs
* Logical Bugs

1. Advantage of Bugzila .

->

| Feature | Description |
| --- | --- |
| 1. Open Source & Free | Bugzilla is completely free to use, making it ideal for startups and open-source projects. |
| 2. Web-Based Interface | Accessible through any web browser — no need for heavy client installations. |
| 3. Customizable Workflow | Supports custom fields, bug states, and workflows tailored to your project’s needs. |
| 4. Advanced Search & Filters | Powerful bug search with filters, keywords, and saved queries. |
| 5. Email Notifications | Sends automatic notifications for any changes in bug status or comments. |
| 6. Role-Based Access Control | Provides permissions based on user roles (admin, developer, tester, etc.). |
| 7. Time Tracking | Allows estimation and tracking of time spent on issues and bugs. |
| 8. Supports Multiple Projects | Can manage multiple projects and products within the same installation. |
| 9. Integration Friendly | Integrates with version control systems (like Git, CVS), CI/CD tools, and test management tools. |
| 10. Strong Reporting & Charts | Generates reports, pie charts, and bar graphs for bug trends and team performance. |
| 11. Audit Trail | Maintains detailed history of every change in bug status or comments. |
| 12. Security Features | Robust access control, private bugs, and user authentication mechanisms. |

1. Difference between priority and severity

->

| Aspect | Priority | Severity |
| --- | --- | --- |
| Definition | Urgency to fix the defect | Impact of the defect on the system |
| Focus | Business/User need | Technical functionality |
| Set By | Project Manager / Product Owner | Tester / QA Engineer |
| Represents | How quickly the bug should be fixed | How serious the bug is |
| Affects | Development schedule | System performance/functionality |

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

->

| Agile Methodology | Description |
| --- | --- |
| 1. Scrum | Most popular Agile framework with fixed-length sprints (usually 2–4 weeks), daily stand-ups, and specific roles like Product Owner, Scrum Master, and Development Team. |
| 2. Kanban | Focuses on visualizing the workflow using a Kanban board (To Do → In Progress → Done) and limiting work-in-progress (WIP) to improve flow. |
| 3. Extreme Programming (XP) | Emphasizes technical excellence and frequent releases with practices like pair programming, TDD, and continuous integration. |
| 4. Lean | Derived from Lean manufacturing, it focuses on eliminating waste, optimizing processes, and delivering only what adds value. |
| 5. Crystal | A family of Agile methodologies tailored by team size and project criticality. Emphasizes people, interaction, and simplicity over rigid processes. |
| 6. Feature-Driven Development (FDD) | A model-driven, short-iteration process focused on designing & building features in a structured way. |
| 7. Dynamic Systems Development Method (DSDM) | A full Agile project delivery framework that focuses on fixed cost and schedule, while flexing scope. |
| 8. Agile Unified Process (AUP) | A simplified version of the Rational Unified Process (RUP), adapted to Agile principles. |
| 9. SAFe (Scaled Agile Framework) | A structured framework that enables Agile at scale across large enterprises with multiple teams and layers. |

1. Explain the difference between Authorization and Authentication in Web testing. What are the common problems faced in Web testing?

->

| Aspect | Authentication | Authorization |
| --- | --- | --- |
| Definition | Verifies who the user is | Determines what the user can access |
| Purpose | Confirms user identity | Grants or denies permission |
| Process | Usually involves login (username & password) | Happens after successful authentication |
| Example | Logging into a website with valid credentials | Accessing admin dashboard only if you're an admin |
| Occurs When | Before authorization | After authentication |
| Testing Focus | - Valid/Invalid login- Session timeout- Multi-factor login | - Role-based access- Page/resource restrictions- Access control testing |

common problems faced in Web testing:

Functionality Issues:

- Broken links  
- Form submission failures  
- Incorrect business logic

Security Issues:

- Broken authentication  
- Unauthorized access  
- SQL injection or XSS vulnerabilities

Performance Issues:

- Slow page load times  
- Crashes under heavy load

Compatibility Issues:

- Website not rendering correctly on different browsers or devices

UI/UX Issues:

- Misaligned buttons  
- Inconsistent fonts/colors  
- Non-responsive design

Localization Issues:

- Incorrect translations  
- Date/time format errors

1. Create TestCases on Whatsapp Group Chat.

-> https://docs.google.com/spreadsheets/d/1W3v374UcrrYoNKJXOzPfPM02NzR7S9UHBG3nMMZd9kk/edit?usp=sharing

1. Write a scenario of only Whatsapp chat messages

-> https://docs.google.com/spreadsheets/d/1b6ZWFwp4008LbbNT9Qn2DoEYEFEKAiAGXA-0qhTnYAE/edit?usp=sharing

1. Write a Scenario of Pen

->

UI Scenario:-

Verify that the length and the diameter of the pen are as per the specifications.

Verify the outer body material of the pen. Check if it is metallic, plastic, or any other material specified in the requirement specifications.

Check the color of the outer body of the pen. It should be as per the specifications.

Verify that the brand name and/or logo of the company creating the pen should be clearly visible.

Verify that any information displayed on the pen should be legible and clearly visible.

Functional Scenario:-

Verify the type of pen, whether it is a ballpoint pen, ink pen, or gel pen.

Verify that the user is able to write clearly over different types of papers.

Check the weight of the pen. It should be as per the specifications. In case not mentioned in the specifications, the weight should not be too heavy to impact its smooth operation.

Verify if the pen is with a cap or without a cap.

Verify the color of the ink on the pen.

Check the odor of the pen’s ink on writing over a surface.

Verify the pen’s performance on different paper textures (smooth, rough, glossy, etc.) to ensure consistent ink dispersion and grip.

Verify the surfaces over which the pen is able to write smoothly apart from paper e.g. cardboard, rubber surface, etc.

Verify that the text written by the pen should have consistent ink flow without leaving any blob.

Check that the pen’s ink should not leak in case it is tilted upside down.

Verify if the pen’s ink should not leak at higher altitudes.

Verify if the text written by the pen is erasable or not.

Check the functioning of the pen by applying normal pressure during writing.

Verify the strength of the pen’s outer body. It should not be easily breakable.

Verify that text written by pen should not get faded before a certain time as mentioned in the specification.

1. Write a Scenario of Pen Stand

->

Verify the pen stand can hold one pen securely.

Verify the pen stand can hold multiple pens/pencils (up to full capacity).

Verify the pen stand supports pens of different thicknesses.

Verify whether tall or long stationery items stay upright without falling.

Verify pens do not slide out or tilt at extreme angles.

Verify the ease of inserting and removing pens.

Verify the pen stand is stable when fully loaded.

Verify that the pen stand does not topple when a single item is removed.

Verify if the stand can accommodate non-pen items

Verify it supports both round and hexagonal pen shapes.

1. Write a Scenario of Door

->

Verify if the door is single door or bi-folded door.

Check if the door opens inwards or outwards.

Verify that the dimension of the doors are as per the specifications.

Verify that the material used in the door body and its parts is as per the specifications.

Verify that color of the door is as specified.

Verify if the door is sliding door or rotating door.

Check the position, quality and strength of hinges.

Check the type of locks in the door.

Check the number of locks in the door interior side or exterior side.

Verify if the door is having peek-hole or not.

Verify if the door is having stopper or not.

Verify if the door closes automatically or not – spring mechanism.

Verify if the door makes noise when opened or closed.

Check the door condition when used extensively with water.

Check the door condition in different climatic conditions- temperature, humidity etc.

Check the amount of force- pull or push required to open or close the door.

1. Write a Scenario of ATM

->

UI Scenario:-

Verify that all the labels and controls including text boxes, buttons, images, and links are present on the screen.

Check the informative text written displayed on the screen is clearly visible and legible.

Verify that the size, color, and UI of the different objects are as per the specifications.

Verify that the application’s UI is responsive i.e. it should adjust to different screen resolutions of ATM machines.

Functional Scenario:-

Verify the type of ATM machine, if it has a touch screen, both keypad buttons only, or both.

Verify that on properly inserting a valid card different banking options appear on the screen.

Check that no option to continue and enter credentials is displayed to the user when the card is inserted incorrectly.

Verify that the touch of the ATM screen is smooth and operational.

Verify that the user is presented with the option to choose a language for further operations.

Check that the user is asked to enter a pin number before displaying any card/bank account detail.

Verify that there is a limited number of attempts up to which the user is allowed to enter the pin code.

Verify that if the total number of incorrect pin attempts gets surpassed then the user is not allowed to continue further. And operations like temporary blocking of the card, etc get initiated.  
Check that the pin is displayed in masked form when entered.

Verify that the user is presented with different account type options like- saving, current, etc.

Verify that the user is allowed to get account details like available balance.

Check that the correct amount of money gets withdrawn as entered by the user for cash withdrawal.

Verify that the user is only allowed to enter the amount in multiple denominations as per the specifications.

Verify that the user is prompted to enter the amount again in case the amount entered is less than the minimum amount configured.

Check that the user cannot withdraw more amount than the total available balance and a proper message should be displayed.

1. When to used Usablity Testing?

->

Usability testing is used to evaluate how easy and user-friendly a product is by observing real users as they interact with it. You should use usability testing in the following situations

* During Design phase
* Before launching the product
* After major update or redesign
* To compare alternative

1. What is the procedure for GUI Testing?

->

GUI Testing is used to ensure that the user interface of a software application meets its specifications in terms of functionality, design, and usability.

Step by step procedure for GUI testing:

* Understand GUI requirement
* Prepare GUI test plan
* Design test cases
* Prepare test data
* Execute GUI test cases
* Report and track bug
* Re- test and regression testing

1. Write a scenario of Microwave Owen

->

Verify that the dimensions of the oven are as per the specification provided.

Verify that the oven’s material is optimal for its use as an oven and as per the specification.

Verify that the oven heats the food at the desired temperature properly.

Verify that the oven heats food at the desired temperature within a specified time duration.

Verify the ovens functioning with the maximum attainable temperature.

Verify the ovens functioning with minimum attainable temperature.

Verify that the oven’s plate rotation speed is optimal and not too high to spill the food kept over it.

Verify that the oven’s door gets closed properly.

Verify that the oven’s door opens smoothly.

Verify the battery requirement of the microwave oven and check that it function’s smoothly at that power.

Verify that the text written over the oven’s body is clearly readable.

Verify that the digital display is clearly visible and functions correctly.

Verify that the temperature regulator is smooth to operate.

Verify that the temperature regulator works correctly.

Check the maximum capacity of the oven and test its functioning with that volume of food.

1. Write a scenario of Coffee vending Machine

->

UI scenario – Verify that the dimension of the coffee machine is as per the specification.

Verify that outer body, as well as inner part’s material, is as per the specification.

Verify that the machine’s body color as well brand is correctly visible and as per specification.

Verify the input mechanism for coffee ingredients-milk, water, coffee beans/powder, etc.

Verify that the quantity of hot water, milk, coffee powder per serving is correct.

Verify the power/voltage requirements of the machine.

Verify the effect of suddenly switching off the machine or cutting the power. The machine should stop in that situation and in power resumption, the remaining coffee should not get come out of the nozzle.

Verify that coffee should not leak when not in operation.

Verify the amount of coffee served in single-serving is as per specification.

Verify that the digital display displays correct information.

Check if the machine can be switched on and off using the power buttons.

Check for the indicator lights when the machine is switched on-off.

Verify that the functioning of all the buttons work properly when pressed.

Verify that each button has an image/text with it, indicating the task it performs.

Verify that complete quantity of coffee should get poured in a single operation, no residual coffee should be present in the nozzle.

1. Write a scenario of chair

->

Verify that the chair is stable enough to take an average human load.

Check the material used in making the chair-wood, plastic etc.

Check if the chair’s leg are level to the floor.

Check the usability of the chair as an office chair, normal household chair.

Check if there is back support in the chair.

Check if there is support for hands in the chair.

Verify the paint’s type and color.

Verify if the chair’s material is brittle or not.

Check if cushion is provided with chair or not.

Check the condition when washed with water or effect of water on chair.

Verify that the dimension of chair is as per the specifications.

Verify that the weight of the chair is as per the specifications.

Check the height of the chair’s seat from floor.

1. Create Test Cases on Compose Mail Functionality.

->

<https://docs.google.com/spreadsheets/d/1TyR3HM4CnBL8ytug6FDdjz0gc83aEUGn5OFja6HnrPY/edit?usp=sharing>

1. Online shopping to buy product (flipkart)

->

https://docs.google.com/spreadsheets/d/17bc2P8Q8qImRjnuhE0t6bWEqgiNzGSi9EKFVHQ7dTRk/edit?usp=sharing

1. Write a Scenario of Wrist Watch

->

Verify the type of watch – analog or digital.

In the case of an analog watch, check the correctness time displayed by the second, minute, and hour hand of the watch.

In the case of a digital watch, check the digital display for hours, minutes, and seconds is correctly displayed.

Verify the material of the watch and its strap.

Check if the shape of the dial is as per specification.

Verify the dimension of the watch is as per the specification.

Verify the weight of the watch.

Check if the watch is waterproof or not.

Verify that the numbers in the dial are clearly visible or not.

Check if the watch is having a date and day display or not.

Verify the color of the text displayed in the watch – time, day, date, and other information.

Verify that clock’s time can be corrected using the key in case of an analog clock and buttons in case of a digital clock.

56. Write a Scenario of Lift(Elevator)

->

Verify the dimensions of the lift.

Verify the type of door of the lift is as per the specification.

Verify the type of metal used in the lift interior and exterior.

Verify the capacity of the lift in terms of the total weight.

Verify the buttons in the lift to close and open the door and numbers as per the number of floors.

Verify that the lift moves to the particular floor as the button of the floor is clicked.

Verify that the lift stops when the up/down buttons on a particular floor are pressed.

Verify if there is an emergency button to contact officials in case of any mishap.

Verify the performance of the floor – the time taken to go to a floor.

Verify that in case of power failure, the lift doesn’t free-fall and gets halted on the particular floor.

Verify lifts working in case the button to open the door is pressed before reaching the destination floor.

Verify that in case the door is about to close and an object is placed between the doors if the doors sense the object and again open or not.

1. Write a Scenario of Whatsapp payment

->

Verify that the user can access the WhatsApp Payments setup.

Verify that user can add a bank account using UPI

Verify that the UPI PIN setup works correctly.

Verify that user receives a confirmation after successfully linking a bank account

Verify behavior when linking a bank not supported by WhatsApp.

Verify error handling when entering an incorrect bank account detail.

Verify user can remove and re-link a bank account.

Send money to a contact with WhatsApp Payments enabled

Send money to a contact without WhatsApp Payments set up.

Send money with valid UPI PIN.

Send money with invalid UPI PIN.

Send money and then immediately check the transaction history.

Send money while connected to a slow or unstable internet.

Attempt to send an amount exceeding bank or app limits.

Send money with no balance in bank account.

Send money with invalid amount

Send money using dual SIM where only one SIM is UPI-registered.

Receive money from a WhatsApp contact.

Receive money when bank account is unlinked or inactive.

Receive money with notification and in-chat confirmation.

Receive money and verify it in the transaction history.

Request money from a contact.