Assignment -2

Model-2

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

* White box testing based on an analysis of the internal structure of the component or system.
* The testers have no knowledge of how the system or component is structured inside the box.
* For example, when performing system or acceptance testing, the requirements specification or functional specification many from the basis of the tests.
* Typically used at System Test phase, although can be useful throughout the test lifecycle.
* The tester is oblivious to the system architecture and dose not have access to the source code.

**Types of white box testing**

Top Down testing

Hybrid Approach

Bottom up approach

Unit testing

Integration testing

Execution testing

Operations testing

Mutation testing

Statement Coverage

Brach Coverage

Path Coverage

White Box testing

1. **Unit testing:** - **Unit Testing** is a software testing technique in which individual units or components of a software application are tested in isolation.

* **Unit tests** for these code units and run them automatically every time you make changes.
* Unit testing promotes modular code, ensures better test coverage, and saves time by allowing developers to focus more on coding than manual testing.
* Understanding of the software development process.
* Knowledge of programming languages and development tools.

1. **Integration testing: -** Integration testing is the process of testing the interface between two software units or modules.

* The purpose of integration testing is to expose faults in the interaction between integrated units.
* Integration testing is typically performed after unit testing and before system testing.
* Integration testing is typically performed after unit testing and before system testing.

1. **What is error, defect, bug and failure?**

* **Error: -** A human action that produces an incorrect result.
* An error is a mistake, an incorrect action, or a deviation from what is correct.
* Error can occur in many situations, including in speech, writing, calculations.
* **Defect: -** A flaw in a component or system that can cause the component or system to fail to perform its required function.
* A bug is a type of error in software code, while a defect is a broader term that includes bugs and other issues.
* Defect testing is the process of finding and fixing coding errors in software.
* **Failure: -** Deviation of the component or system from its expected delivery, service or result.
* Failure is the social concept of not meeting a desirable or intended objective, and is usually viewed as the opposite of success.
* Failures occur when defects remain undetected or unaddressed and reach the end users.
* **Bug: -** A fault in a program which causes the program which causes the program to perform in an unintended or unanticipated manner.
* In computer technology, a bug is a coding error in a computer program.

1. **Difference between QA v/s QC tester.**

|  |  |  |  |
| --- | --- | --- | --- |
| S.N. | Quality Assurance | Quality Control | Testing |
| 1 | Activities which ensure the implementation of procedures, 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 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. **What is 7 key principles? Explain in detail?**

* Software Testing is a crucial part of the Software Development Life Cycle (SDLC), ensuring that the final product meets quality standards. The **7 key principles of software testing** help testers perform efficient and effective testing.

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**

1. **Testing shows presence of defects**

* Testing helps in identifying defects in the software but **cannot guarantee** that the software is completely defect-free.
* Even if no defects are found, it does not mean the system is perfect; there may still be hidden issues.
* The goal of testing is to **reduce the number of defects**, not to prove that the software is completely flawless.
* However, Testing cannot prove that there are no defects present.

1. **Exhaustive testing in impossible!**

* It is impossible to test all possible inputs, conditions, and scenarios in a real-world application.
* Instead of testing everything, **risk-based and prioritized testing** is used.
* Testers focus on the most **important functionalities, high-risk areas, and critical features**.
* 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.

1. **Early testing**

* The earlier a defect is found in the SDLC, the cheaper it is to fix.
* Testing should start at the **requirement analysis and design phase** rather than waiting until development is complete.
* Early testing, such as **unit testing and integration testing**, helps in identifying issues before they become costly.
* Testing activities should start as early as possible in the development life cycle.

1. **Defect clustering**

* A small number of modules in a software system often contain the most defects.
* This follows the **Pareto Principle (80-20 rule)**: 80% of the defects come from 20% of the modules.
* Testers focus more on these **high-risk areas** to increase testing efficiency.
* Similarly, most operational failures of a system are usually confined to a small number of modules
* An important consideration in test prioritisation!

1. **The pesticide paradox**

* Running the **same test cases repeatedly** will eventually stop finding new defects.
* To overcome this, test cases should be **regularly reviewed, updated, and modified**.
* New test scenarios should be created to improve defect detection.
* As bugs are eliminated by the programmers, the software improves.

1. **Testing is Context Dependent**

* The approach to testing depends on the **type of software** being tested.
* Testing is done differently in different contexts.
* Different kinds of sites are tested differently.
* For example:
* A banking application requires **high security and precision** testing.
* A gaming application requires **performance and usability** testing.
* The testing strategy must align with the business goals and project requirements.

1. **Absence-of-Errors is a fallacy**

* Even if the software is **100% bug-free,** it may still fail to meet user expectations.
* The software must be **tested against requirements and user needs**, not just for technical correctness.
* A product that is error-free but does not solve the user’s problem is considered a failure.
* Even after defects have been resolved it may still be unusable and/or does not fulfil the users’ needs and expectations.

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

* Boundary Value testing is a software testing technique where test cases are created based on the boundary values of input conditions.
* Boundary value analysis is a methodology for designing test cases that concentrates software testing effort on cases near **the limits of valid ranges.**
* The trick is to concentrate software testing efforts at the extreme ends of the equivalence classes.
* Boundary value Analysis (BVA) uses the same analysis of partitions as EP and is usually used in conjunction with EP in the test case design.
* For example, if a system accepts input between 1 and 100, boundary values would be 0,1,100 and 101. These values are tested to check if the system behaves correctly at the limits.
* This technique is widely used because it helps detect issues with minimal test cases while ensuring good coverage.

1. **What is Integration testing?**

* Integration testing is a type of software testing where individual modules or components of a software application are combined and tested as a group.
* The main goal is to verify how different modules interact with each other**.**

**Why is Integration Testing Important?**

* Ensures that different modules work together correctly.
* Defects issues related to data flow and communication between components.
* Helps uncover interface defects, such as mismatched data formats or incorrect API calls.

**Types of integration testing**

1. **Big Bang Integration Testing** – All components are integrated at once and tested together**.**
2. **Incremental Integration testing –** Components are integrated and tested step by step. It includes:

* **Top-Down Approach –** Testing starts from the top module and moves downward.
* **Bottom-Up Approach** – Testing starts from the lower modules and moves upward.
* **Hybrid (sandwich) Approach** – A combination of top-down and bottom-up testing.

**Example of integration testing**

**Imagine an e-commerce application where:**

* The **login module** sends user credentials to the **Authentication Module.**
* The **Authentication Module** verifies the credentials and sends a response back.
* Integration testing ensures that these two modules communicate properly and work as expected.

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

* Black box testing is a software testing technique where the tester evaluates the functionality of an application without knowing its internal code structure, implementation, or logic.
* The focus is on input and output—ensuring that the software behaves as expected for given inputs.

1. **Equivalence partitioning (EP)**

* Divides input data into different partitions or groups that should produce the same behaviour.
* Example: If an age input field accepts values from 18 to 60, you can test three partitions: below 18 (invalid), between 18-60 (valid), and above 60 (invalid).

1. **Boundary Value Analysis (BVA)**

* Focuses on testing the boundaries between partitions because errors are more likely to occur at edges.
* Example: If a password field accepts 8-16 characters, test cases include 7 (invalid), 8 (valid), 16 (valid), and 17 (invalid).

1. **Decision Table Testing**

* Uses a decision table to test combinations of inputs and corresponding outputs.
* Example: If an e-commerce site offers a discount based on customer type (new/existing) and order amount, a decision table can help cover all possibilities.

1. **State Transition Testing**

* Used when a system behaves differently based on its current state.
* Example: A login system may transition from "Logged Out" → "Logged In" → "Locked Out" based on user actions.

1. **Use case testing**

* Tests real-world scenarios or user interactions with the system.
* Example: Testing an ATM withdrawal process from card insertion to cash withdrawal.

1. **Error Guessing**

* Relies on the tester's experience and intuition to identify common mistakes.
* Example: Testing a registration form by entering special characters or leaving fields empty to check error handling.

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

* **Equivalence Partitioning (EP)** is a **black-box testing technique** used to divide input test data into different groups (partitions) that are expected to behave similarly.
* Instead of testing every possible input, testers select one representative value from each partition, reducing the number of test cases while maintaining test effectiveness.

**Key Concept**

* Inputs are **divided into classes (partitions)** based on similar behaviour.
* One test case from each partition is **enough to represent the entire group**.
* Helps in **reducing redundant test cases** while ensuring good coverage.

Example: Validating Age Input (18-60 years allowed)

|  |  |  |
| --- | --- | --- |
| **Input Range** | **Equivalence Class** | **Test Case Example** |
| Below 18 | Invalid | 16 |
| 18 to 60 | Valid | 30 |
| Above 60 | Invalid | 65 |

**Benefits of Equivalence Partitioning**

* Reduces the number of test cases while maintaining efficiency.
* Ensures better **test coverage** with minimal effort.
* Identifies defects effectively without exhaustive testing.

1. **What is exploratory testing?**

* **Exploratory Testing** is a **hands-on, unscripted testing approach** where testers actively explore the software without predefined test cases.
* Instead of following a structured test plan, testers use their creativity, experience, and intuition to find defects and improve the software's quality.

**Key Features of Exploratory Testing**

* **Unscripted & Dynamic** – No pre-written test cases; testers design tests on the fly.
* **Simultaneous Learning** – Testers learn about the application while testing it.
* **Experience-Based** – Relies on the tester's domain knowledge and skills.
* **Flexible & Adaptive** – Can quickly adjust to new findings or changes in the system.
* **Good for Finding Edge Cases** – Helps discover unexpected bugs that structured tests may miss.

**When to Use Exploratory Testing?**

* When **requirements are unclear** or incomplete.
* When there is **limited time** for testing.
* To **complement scripted testing** for better defect detection.
* In **agile environments**, where software changes frequently.

**Benefits of Exploratory Testing**

* Helps uncover **critical bugs quickly**.
* Encourages **tester creativity and critical thinking**.
* Saves time by avoiding unnecessary documentation.
* Useful in **agile and fast-paced projects**.

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

* In software testing, the **level of risk** is determined by two main factors:

1. **Likelihood (Probability) of Failure** – How likely is it that a defect will occur in the software?

* **Complexity of the code**: More complex code has a higher chance of defects.
* **Changes in code**: Frequent modifications increase the chances of introducing defects.
* **Developer experience**: Less experienced developers may introduce more errors.
* **Technology stack**: New or less familiar technologies can increase the likelihood of issues.

1. **Impact of Failure** – How severe would the consequences be if the defect occurs?

* **Business impact**: A bug in a banking system is more critical than in a gaming app.
* **User impact**: If a large number of users are affected, the risk is higher.
* **Legal and compliance issues**: Some defects may lead to legal consequences (e.g., GDPR violations).

1. **What is alpha testing?**

* **Alpha testing** is a type of **acceptance testing** performed before releasing the software to end users
* It is conducted **internally by developers, testers, and sometimes a selected group of users** in a controlled environment.

**Key Features of Alpha Testing**

* **Conducted by:** Internal testers and developers (sometimes with potential users).
* **Testing Environment:** Controlled, similar to real-world conditions but within the organization.
* **Objective:** Detect and fix **major bugs, crashes, and usability issues** before moving to Beta Testing.
* **Phase:** Done **before Beta Testing**, usually at the end of development.
* **Type of Testing:** Includes both **functional and non-functional testing**, such as performance, usability, and security testing.

**Example Scenario**

* Suppose a company is developing a **new e-commerce app**. Before launching it for public testing (Beta Testing), the **in-house testing team** tests it for **major bugs, navigation issues, and security loopholes**. This phase is called **Alpha Testing**.

1. **What is a beta testing?**

* **Beta Testing** is the final phase of software testing before a product is officially released.
* It is performed by **real users in a real-world environment** to identify any remaining issues that were not found during **Alpha Testing**.

**Key Features of Beta Testing**

* **Conducted by:** Real users or customers outside the development team.
* **Testing Environment:** Real-world conditions (outside the company).
* **Objective:** Gather feedback, identify **bugs, usability issues, and performance problems**.
* **Phase:** Done **after Alpha Testing** and before the final release.
* **Type of Testing:** Focuses on **user experience, reliability, and compatibility** on different devices and environments.

**Example Scenario**

* A company develops a new **music streaming app**. After internal Alpha Testing, they release a **Beta version** to a limited number of users. These users test the app in real-world conditions and report issues like **app crashes, slow loading, or missing features**. The company then fixes these issues before the final launch.

1. **What is component testing?**

* **Component Testing** (also called **Module Testing** **or Unit Testing**) is a type of **software testing** where individual components or modules of an application are tested in **isolation** to ensure they function correctly.

**Key Features of Component Testing**

* **Conducted by:** Developers or testers (often using test scripts).
* **Testing Level:** **Lowest level of testing** (before integration testing).
* **Objective:** Verify that **each component or module works correctly** before integrating them.
* **Scope:** Focuses on a **single module, function, or feature**.
* **Type of Testing:** Can be **manual or automated** and includes both **functional and non-functional testing**.

### ****Example Scenario****

### A developer is working on an **online banking application**. The login module is tested separately to check if: The user can enter a valid username and password. Incorrect credentials show an error message. The "Forgot Password" link works properly.

### What is functional testing?

### **Functional Testing** is a type of software testing that verifies whether the application **functions correctly** according to the **specified requirements**. It focuses on testing the software’s **features, inputs, outputs, and behaviour** to ensure that it meets the expected functionality.

### Key Features of Functional Testing

### **Conducted by:** Testers (Manual & Automation Testing).

### **Testing Focus:** Ensures the software's **functionalities work as expected**.

### **Testing Basis:** Based on **requirements, user stories, and specifications**.

### **Type of Testing:** **Black Box Testing** (no need to check the internal code).

### **Objective:** Validate **correctness, usability, and compliance** with business needs

### 

1. **Unit Testing** – Tests individual components (conducted by developers).
2. **Integration Testing** – Ensures modules work together properly.
3. **System Testing** – Checks the entire system’s functionality.
4. **User Acceptance Testing (UAT)** – Ensures the system meets business needs.
5. **Smoke & Sanity Testing** – Quick checks for major functionality.

**Example Scenario**

A company develops an **online shopping website**. Functional Testing ensures:  
 Users can **log in** with valid credentials.  
 Users can **add products to the cart** and proceed to checkout.  
 Payment processing works with different methods (Credit Card)

Confirmation emails are sent after purchase.

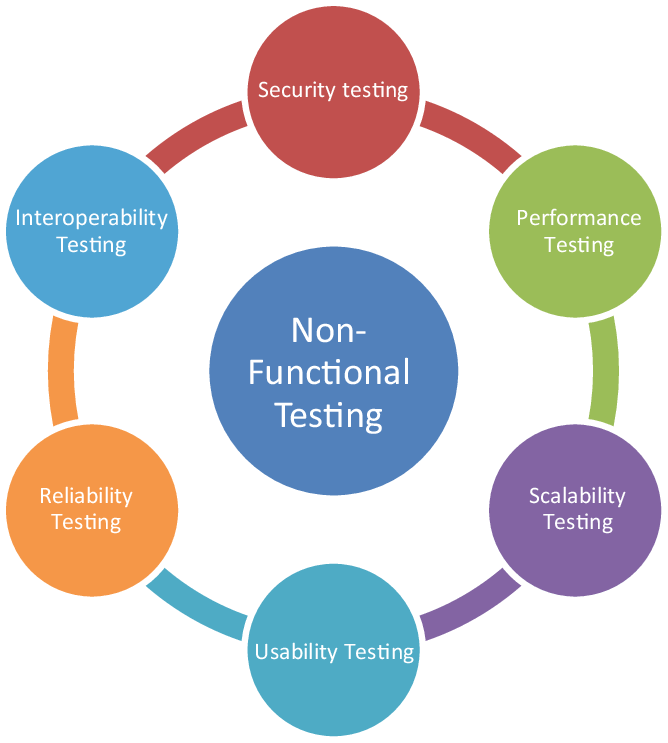
1. **What is Non-functional testing?**

* **Non-Functional Testing** is a type of software testing that evaluates the **performance, reliability, usability, and scalability** of an application.
* It ensures that the system meets **quality standards** beyond just functional correctness.

**Key Features of Non-Functional Testing**

* **Conducted by:** Testers (Performance Testers, Security Testers, etc.).
* **Testing Focus:** Checks **how well** the system performs under various conditions.
* **Testing Basis:** Based on **performance requirements & industry standards**.
* **Type of Testing:** **Black Box Testing** (focuses on external behaviours, not internal code).
* **Objective:** Improve **user experience, system stability, and security**.

**Types of Non-Functional Testing**

****

1. **Performance Testing** – Tests speed, responsiveness, and stability.
2. **Load Testing** – Checks system behaviours under expected user load.
3. **Stress Testing** – Evaluates performance under extreme conditions.
4. **Usability Testing** – Ensures the system is user-friendly.
5. **Security Testing** – Identifies vulnerabilities and security risks.
6. **Compatibility Testing** – Tests system behaviours on different devices, browsers, and OS.
7. **Reliability Testing** – Ensures the system runs without failure for a specific time.

### ****Example Scenario****

A company develops a **movie streaming app**. Non-Functional Testing ensures:  
 The app can handle **1 million users streaming at the same time** (**Load Testing**).  
 The app does not crash when users **watch videos for hours** (**Stress Testing**).  
 The app loads within **3 seconds on mobile and web** (**Performance Testing**).  
 The app's payment system is **protected against hacking** (**Security Testing**).

1. **What is GUL testing?**

* **GUI (Graphical User Interface) Testing** is a type of software testing that checks the **visual elements, layout, and functionality** of a software application’s user interface. It ensures that the **buttons, menus, text fields, colours, icons, and other UI elements** work correctly and provide a smooth user experience.

**Key Features of GUI Testing**

* **Conducted by:** Testers (manual or automated GUI testing tools).
* **Testing Focus:** Ensures the **interface looks and works as expected**.
* **Objective:** Validate **design consistency, usability, and responsiveness**.
* **Type of Testing:** **Black Box Testing** (focuses on the external interface).

**Types of GUI Testing**

1. **Visual Testing** – Ensures UI elements are correctly aligned and displayed.
2. **Usability Testing** – Checks if the application is easy to use.
3. **Functionality Testing** – Tests if buttons, forms, and menus work correctly.
4. **Compatibility Testing** – Ensures UI works across different devices and browsers.
5. **Responsive Testing** – Checks adaptability of UI on different screen sizes.
6. **What is Adhoc testing?**

* **Adhoc Testing** is an informal and unstructured type of testing where testers explore the application randomly without following any predefined test cases or plans.
* The goal is to find defects **quickly and unpredictably** by simulating real-world user behaviour.

Key Features of Adhoc Testing

* **Conducted by:** Testers or developers (without test cases).
* **Testing Focus:** **Randomly explores** the application to find hidden bugs.
* **Testing Basis:** **No documentation or predefined test plan**.
* **Type of Testing:** **Exploratory Testing** (relies on tester’s knowledge & intuition).
* **Objective:** Discover **unexpected defects and edge-case issues**.

Types of Adhoc Testing

1. **Monkey Testing** – Random inputs are given to check system stability.
2. **Exploratory Testing** – Testers use their experience to explore functionalities.
3. **Error Guessing** – Based on intuition and past defect knowledge.

**Example Scenario**

A tester is checking an **e-commerce website** without predefined test cases:  
 Tries adding **1000 items** to the cart to see if it crashes.  
 Enters **special characters** in the login field to check for errors.  
 Clicks buttons **repeatedly in quick succession** to test responsiveness.

1. **What is load testing?**

* **Load Testing** is a type of **performance testing** that evaluates how a system performs under an expected user load.
* It helps determine whether the software can handle a specific number of users, transactions, or data volume without performance degradation.

**Key Features of Load Testing**

* **Conducted by:** Performance Testers using tools like **JMeter, LoadRunner, and Gatling**.
* **Testing Focus:** Measures system performance under **normal and peak load conditions**.
* **Objective:** Identify **bottlenecks, slow responses, and potential failures**.
* **Type of Testing:** **Non-Functional Testing** (focuses on system behavior).

**Example Scenario**

A company launches a **new flight booking website**. Load Testing ensures:  
 The website can handle **5,000 concurrent users** searching for flights.  
 Response time remains **under 2 seconds** during high traffic.  
 No system crashes occur when processing **1,000 bookings per minute**.

1. **What is stress testing?**

* **Stress Testing** is a type of **non-functional testing** that evaluates how a software application behaves **under extreme conditions** (beyond normal usage).
* It helps determine the **breaking point** of a system and assesses its ability to recover from failure.

**Key Features of Stress Testing**

* **Conducted by:** Performance Testers using tools like **JMeter, LoadRunner, and Gatling**.
* **Testing Focus:** Checks system **stability, error handling, and recovery** under extreme conditions.
* **Objective:** Identify **bottlenecks, crashes, memory leaks, and performance degradation**.
* **Type of Testing:** **Non-Functional Testing** (focuses on reliability & robustness).

**Example Scenario**

A **banking app** is stress-tested by:  
 **Simulating 1 million users** logging in at once to test server overload.  
 **Forcing high CPU & memory usage** to see if the app crashes.  
 **Sudden network disconnections** to check recovery time.

**Stress Testing vs. Load Testing**

|  |  |  |
| --- | --- | --- |
| **Aspect** | **Load Testing** | **Stress Testing** |
| Objective | Measures system behavior under expected load | Tests system under extreme conditions |
| Focus | Performance under normal & peak load | System failure and recovery testing |
| Example | Checking if a website can handle **10,000 users** | Seeing if the website **crashes at 100,000 users** |

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

* In software testing, defects (or bugs) are classified into different **categories** based on their nature, severity, and impact on the application.

Business Logic Defects

Configuration Defects

Internationalization Defects

Data Defects

Functional Defects

Compatibility Defects

Usability Defects

Performance Defects

Security Defects

Integration Defects

Integration Defects

Categories of Defects in Software Testing

1. **Functional Defects**

* Issues related to the core functionality of the application.
* **Example:** A login button does not work or a user cannot add items to the cart.

1. **Performance Defects**

* Problems related to the speed, stability, and responsiveness of the system.
* **Example:** A webpage takes **too long to load** or crashes under high traffic.

1. **Usability Defects**

* Issues that make the application difficult to use or navigate.
* **Example:** Poor UI design, **unreadable font**, or **confusing navigation.**

1. **Compatibility Defects**

* Issues that occur when the software does not work properly across different **browsers, devices, or operating systems**.
* **Example:** A website works fine in Chrome but **breaks in Safari**.

1. **Security Defects**

* Vulnerabilities that could lead to **data breaches, unauthorized access, or hacking**.
* **Example:** A user can **view someone else's personal data** without permission.

1. **Integration Defects**

* Problems that occur when different system modules or third-party services do not interact properly.
* **Example: A payment gateway fails** to process transactions due to an API error.

1. **Data Defects**

* Issues related to incorrect or inconsistent **data storage, retrieval, or display**.
* **Example:** A **user's order history** shows **wrong** or **duplicate** records.

1. **Localization & Internationalization Defects**

* Issues that arise when software is adapted for different languages, regions, or cultural preferences.
* **Example:** The **date format** is incorrect for different countries **(MM/DD/YYYY vs. DD/MM/YYYY).**

1. **Hardware & Configuration Defects**

* Errors due to incorrect or unsupported hardware and system configurations.
* **Example:** An app **fails to run on lower-end devices** or crashes on **specific screen resolutions.**

1. **Business Logic Defects**

* Issues where the software does not follow the expected **business rules** or requirements.
* **Example:** A **discount coupon** applies incorrectly, giving users **more discount than allowed**.

1. **Mention what big bang testing?**

* **Big Bang Testing** is an **integration testing approach** where all **modules or components** of the software are integrated **at once** and then tested as a whole.
* This method is used when the entire system is ready for testing instead of integrating and testing modules step by step.

**Key Features of Big Bang Testing**

* **Conducted by:** Testers after all modules are developed.
* **Testing Approach:** **All components** are combined and tested together.
* **Objective:** Check if the integrated system works as expected.
* **Type of Testing:** **Integration Testing** (verifies module interactions).

**Example Scenario**

A company develops a **hotel booking system** with modules like **Login, Room Selection, Payment, and Confirmation**. Instead of testing each module separately, all modules are integrated at once and tested together.

**Advantages of Big Bang Testing**

* Saves time on individual module testing.
* Useful when all components are developed simultaneously.
* Helps find major integration issues.

**Disadvantages of Big Bang Testing**

* **Difficult to identify root causes** of defects.
* Fixing issues is costly and time-consuming.
* **High risk** since testing is delayed until all modules are ready.

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

* **Exit Criteria** in software testing define the conditions that must be met **before testing can be stopped**.
* It ensures that the testing process is **complete, meets quality standards, and is ready for release or the next phase.**

**Key Purposes of Exit Criteria**

* **Ensures Testing Completeness** – Confirms that all test cases have been executed.
* **Defines Quality Standards** – Verifies that the software meets the **required quality level**.
* **Prevents Premature Release** – Ensures that testing is not stopped **before major defects are fixed**.
* **Facilitates Decision-Making** – Helps stakeholders decide **whether to move to the next phase**.
* **Improves Software Reliability** – Ensures that critical bugs are resolved and the software is stable.

**Common Exit Criteria Examples**

* **Test Case Execution:** **95%–100%** of planned test cases are executed.
* **Defect Fixing:** No **critical or high-severity defects** remain unresolved.
* **Performance Goals:** The system meets **performance and security benchmarks**.
* **User Acceptance:** UAT (User Acceptance Testing) is successfully completed.
* **Documentation:** All test reports, defect logs, and test summaries are prepared.

Example Scenario

A **banking application** is being tested. Before exiting testing, the team checks:  
 **All functional tests** (e.g., login, transactions) are completed.  
 **No major security vulnerabilities** remain.  
 **Performance tests confirm** that transactions are processed within **2 seconds**.  
 **All test reports** are documented for review.

1. **When should “regression testing” be performed?**

* **Regression Testing** should be performed whenever there are **changes or updates** in the software to ensure that existing functionality still works correctly. It helps detect any **new defects (regressions)** introduced by modifications.

**Key Situations for Regression Testing**

* **After Bug Fixes** – Ensures fixing a defect **did not break other functionalities**.
* **After New Features Are Added** – Verifies that new changes **don’t affect existing features**.
* **After Code Enhancements or Optimizations** – Confirms that **performance improvements** don’t introduce unexpected issues.
* **After Integration with Other Modules** – Ensures that newly integrated modules work **without affecting previous functionality**.
* **During Regular Maintenance or Updates** – Checks for regressions when the software **undergoes updates or migrations**.

**Example Scenario**

A company updates its **e-commerce website** by adding a **new payment method**. Regression Testing ensures:  
**Existing payment options (Credit Card, PayPal)** still work correctly.

**Cart, checkout, and order confirmation processes** are not broken.  
**No new performance issues** appear due to the changes.

1. **Different between smoke and sanity?**

* Smoke testing and sanity testing are both types of software testing performed to verify the stability of an application, but they have different purposes and scopes.

1. **Smoke Testing**

* **Purpose:** Ensures that the basic and critical functionalities of the application work properly before proceeding with further testing.
* **When Conducted:** Performed after a new build is deployed to check its stability.
* **Scope:** Broad, covering all major functionalities but in a high-level manner.
* **Execution:** Usually automated but can be manual.
* **Failure Impact:** If it fails, the build is rejected, and further testing is halted.
* **Example:** Checking if the application launches, the login functionality works, and essential navigation is functional.

1. **Sanity Testing**

* **Purpose:** Ensures that specific functionalities or bug fixes are working correctly without affecting other parts of the application.
* **When Conducted:** Performed after a minor change, bug fix, or patch update.
* **Scope:** Narrow, focusing only on specific changes or fixes.
* **Execution:** Usually manual, as it involves logical verification.
* **Failure Impact:** If it fails, the specific feature is reworked before further testing continues.
* **Example:** Verifying that a bug fix in the checkout process is correctly implemented and doesn’t impact payment processing.

**Key Differences:**

|  |  |  |
| --- | --- | --- |
| **Feature** | **Smoke Testing** | **Sanity Testing** |
| Purpose | Checks overall system stability | Checks specific functionalities or bug fixes |
| When Performed | After a new build | After bug fixes or minor changes |
| Scope | Broad (entire system) | Narrow (specific features) |
| Execution | Automated or manual | Mostly manual |
| Failure Impact | Build is rejected | Specific features are reworked |

1. **Difference between verification and validation.**

|  |  |
| --- | --- |
| **Verification** | **validation** |
| We check whether we are developing the right product or not. | We check whether the developed product is right. |
| Verification is also known as static testing. | Validation is also known as dynamic testing. |
| Verification includes different methods like Inspections, Reviews, and Walkthroughs. | Validation includes testing like funicle testing, system testing, integration, and User acceptance testing. |
| It is a process of checking the work-products (not the final product) of a development cycle to decide whether the product meets the specified requirements. | It is a process of checking the software during or at the end of the development cycle to decide whether the software follow the specified business requirements. |
| **Quality assurance** comes under verification testing. | **Quality control** comes under validation testing. |
| The execution of code does not happen in the verification testing. | In validation testing, the execution of code happens. |
| In verification testing, we can find the bugs early in the development phase of the product. | In the validation testing, we can find those bugs, which are not caught in the verification process. |
| Verification testing is executed by the Quality assurance team to make sure that the product is developed according to customers' requirements. | Validation testing is executed by the testing team to test the application. |
| Verification is done before the validation testing. | After verification testing, validation testing takes place. |
| In this type of testing, we can verify that the inputs follow the outputs or not. | In this type of testing, we can validate that the user accepts the product or not. |

1. **Explain types of performance testing?**

* Performance testing evaluates how a system or application performs under various conditions, focusing on speed, stability, scalability, and responsiveness to identify potential bottlenecks and ensure the application meets performance requirements.

Types of performance testing

Performance testing

1. **Load testing**

* **Purpose:** To check the system’s performance under expected user load.
* **Goal:** Identify bottlenecks before the software goes live.
* **Example:** Testing a website with 1000 users simultaneously accessing it.

1. **Stress Testing**

* **Purpose:** To test the system beyond its normal load limit.
* **Goal:** Find the system’s breaking point and how it recovers from failure.
* **Example:** Gradually increasing the number of users until the server crashes.

1. **Spike Testing**

* **Purpose:** To test the system’s reaction to sudden large spikes in load.
* **Goal:** See if the system can handle sudden and extreme increases in traffic.
* **Example:** Testing how a ticket booking system performs when a big event goes live.

1. **Endurance Testing (Soak Testing)**

* **Purpose:** To check the system’s performance under a significant load for a long period.
* **Goal:** Identify memory leaks or performance degradation over time.
* **Example:** Running the system with 500 users for 12 hours straight.

1. **Scalability Testing**

* **Purpose:** To determine the system’s ability to scale up or down (users, data, etc.).
* **Goal:** Understand how the system handles increased workload.
* **Example:** Adding more users to a cloud app and monitoring its response time.

1. **Volume Testing (Flood Testing)**

* **Purpose:** To check the system’s performance with a large volume of data.
* **Goal:** Ensure the system can handle high data input/output.
* **Example:** Uploading a million records into the database and checking performance.

1. **Difference between priority and severity.**

|  |  |  |
| --- | --- | --- |
| **Aspect** | **Severity** | **Priority** |
| Definition | Indicates how **serious** the defect is. | Indicates how **urgently** the defect should be fixed. |
| Focus | Related to the **impact** on the system. | Related to the **business needs or urgency.** |
| Set By | Usually set by **Testers or QA team**. | Usually set by **Developers or Product Managers**. |
| Example Levels | Critical, Major, Moderate, Minor | High, Medium, Low |
| Example | App crashes when clicking "Login" = **High severity** | Fix needed before release = **High priority** |

1. **What is bug life cycle?**

* The **Bug Life Cycle** (also called the **Defect Life Cycle**) is the **process a bug goes through from the moment it is found to the moment it is closed or fixed.**

Bug Life Cycle Stages:

1. **New**

* Bug is found and reported by a tester.
* Status: New

1. **Assigned**

* Bug is assigned to a developer to fix.
* Status: Assigned

1. **Open**

* Developer starts analysing and working on the bug.
* Status: Open

1. **Fixed**

* Developer fixes the bug.
* Status: Fixed

1. **Retest**

* Tester retests the bug to confirm if it’s really fixed.
* Status: Retest

1. **Verified**

* If the bug no longer exists, it’s verified by the tester.
* Status: Verified

1. **Closed**

* The bug is officially closed if everything works fine.
* Status: Closed

1. **Explain the different between functional testing and non-functional testing**

|  |  |  |
| --- | --- | --- |
| **Feature** | **Functional Testing** | **Non-Functional Testing** |
| Definition | Testing that verifies **what** the system does | Testing that verifies **how well** the system performs |
| Purpose | To ensure the software behaves according to requirements | To ensure the software meets performance, usability, and other quality standards |
| Focus Area | Features, functions, user interactions | Performance, speed, security, scalability, reliability |
| Based On | Functional Requirements | Non-Functional Requirements |
| Testing Type | Mainly **Black-box Testing** | Can be Black-box or use specific tools |
| Examples | Login, registration, search, payment processing | Load testing, stress testing, usability testing, security testing |
| Tools Used | Selenium, QTP, Test Complete | JMeter, LoadRunner, Postman (API testing), Apache Benchmark |
| Performed By | QA/Testers | QA/Testers or Performance Engineers |

1. **What is the different between the STLC(software testing life cycle)and SDLC(software development life cycle)?**

|  |  |  |
| --- | --- | --- |
| **Aspect** | **SDLC (Software Development Life Cycle)** | **STLC (Software Testing Life Cycle)** |
| Definition | It is the process of **developing** software from start to finish. | It is the process of **testing** the software to ensure quality. |
| Main Focus | Software **design, development, and deployment** | Software **testing activities and bug finding** |
| Involves | Business analysts, developers, project managers, testers, etc. | Testers, QA analysts, and sometimes developers |
| Stages Include | Requirement → Design → Development → Testing → Deployment → Maintenance | Requirement Analysis → Test Planning → Test Design → Test Execution → Test Closure |
| Goal | To deliver a **functional software product** | To ensure the product is **defect-free and meets quality standards** |
| Starts When | As soon as requirements are gathered | After the SDLC requirements phase is complete |
| Ends When | The software is deployed and maintained | Testing is completed and the product is ready for release |

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

|  |  |  |  |
| --- | --- | --- | --- |
| **Aspect** | **Test Scenario** | **Test Case** | **Test Script** |
| Definition | A high-level idea of what needs to be tested | A detailed document with test steps, data, and expected results | A set of instructions or code to automate or execute tests |
| Level | Conceptual (High-Level) | Descriptive (Mid-Level) | Executable (Low-Level) |
| Purpose | To cover end-to-end functionality | To guide manual testing step-by-step | To perform automated testing |
| Detail Level | Brief and broad | Detailed and structured | Very detailed (often code-based) |
| Example | Test the login feature | Steps to login with valid credentials | Selenium/Python script that logs into the system |
| Used in | Test planning | Manual testing | Automated testing |
| Format | One-liner or simple statement | Structured template (title, steps, expected results, etc.) | Programming language or automation tool |

1. **Explain what test plan is? What is the information that should be covered?**

* A **Test Plan** is a detailed document that outlines the **strategy, approach, resources, schedule, and scope** of testing activities for a project.
* It acts like a **roadmap** for your testing process — telling what, how, when, and who will be involved in testing.

**Why is it Important?**

* Helps everyone understand the testing process.
* Ensures test coverage is complete
* Sets expectations for timelines and responsibilities
* Helps manage risks

**What Information Should a Test Plan Cover?**

|  |  |
| --- | --- |
| **Section** | **Description** |
| Test Plan ID | Unique ID for the test plan |
| **Objectives** | What is the goal of testing? (e.g., verify login functionality works as expected) |
| Scope | What will be tested (in scope) and what won’t (out of scope) |
| Test Strategy | Manual or automated? Which levels of testing? (unit, integration, system, etc.) |
| Test Environment | Hardware, software, network setup required |
| Test Deliverables | What documents will be produced? (test cases, reports, defect logs, etc.) |
| Schedule & Milestones | Start/end dates, deadlines for each testing phase |
| Roles and Responsibilities | Who will do what? (testers, developers, QA lead, etc.) |
| Test Items | What components/modules will be tested |
| Entry and Exit Criteria | When can testing start and when is it considered complete |
| Risk Management | Possible risks and mitigation plans |
| Test Tools | Tools used (e.g., Selenium, JIRA, TestRail, etc.) |

1. **What is priority?**

* **Priority** refers to **how soon a defect or test should be fixed or executed**, based on its **business importance** or **customer impact**.
* Priority = "How urgently it needs to be fixed"

**Types of Priority Levels:**

|  |  |  |
| --- | --- | --- |
| **Priority Level** | **Meaning** | **Example** |
| High (P1) | Must be fixed immediately; business impact is huge | Login page not working |
| Medium (P2) | Needs to be fixed soon, but not blocking | Minor UI issue on dashboard |
| Low (P3) | Can be fixed later; minimal impact | Typo in footer text |

**Priority vs Severity (often confused!):**

|  |  |  |
| --- | --- | --- |
| **Attribute** | **Priority** | **Severity** |
| Who decides? | Usually **Product Manager / Business Analyst** | Usually **Tester** |
| Based on | Business needs | Technical impact |
| Answers | “**How soon?”** | “**How bad?”** |
| Example | Logo missing → **Low Severity, High Priority** | App crash → **High Severity, High Priority** |

1. **What is severity?**

* **Severity** refers to **how serious or critical a defect is in terms of the application's functionality.**
* It describes the **impact** of a bug on the system, regardless of how soon it needs to be fixed.

**Types of Severity Levels:**

|  |  |  |
| --- | --- | --- |
| **Severity Level** | **Meaning** | **Example** |
| Critical (S1) | System crash or data loss; no workaround | App crashes on login |
| High (S2) | Major feature broken, but app still runs | Payment not processing |
| Medium (S3) | Minor functionality issue, workaround available | "Save" button takes too long |
| Low (S4) | UI bug or spelling mistake, doesn’t affect usage | Typo in About page |

**Severity vs Priority (Quick Recap):**

|  |  |  |
| --- | --- | --- |
| **Attribute** | **Severity** | **Priority** |
| Decided by | Tester | Project Manager/BA |
| Focus | Technical impact | Business urgency |
| Answers | “**How bad is the bug?”** | “**How soon should we fix it?**” |
| Example | Crash = High Severity | Logo issue before launch = High Priority |

1. Bug categories are ....

* In **software testing, bugs (or defects)** can be grouped into different **categories** based on their nature, cause, or how they affect the application.

**Common Bug Categories in Software Testing**

|  |  |  |
| --- | --- | --- |
| **Category** | D**escription** | **Example** |
| Functional Bugs | When the system doesn’t behave as expected | Login fails with correct credentials |
| UI/UX Bugs | Issues in the user interface or user experience | Misaligned button, wrong colour scheme |
| Performance Bugs | Application is slow or unresponsive under load | App takes 10+ seconds to load a page |
| Security Bugs | Vulnerabilities that can be exploited | User can access admin data without permission |
| Compatibility Bugs | Issues across different devices, browsers, or OS | Works on Chrome but breaks on Firefox |
| Usability Bugs | Makes the app hard to use or confusing | No error message for empty login field |
| Logical Bugs | Errors in business logic or workflows | Discount applied even when cart is empty |
| Boundary/Validation Bugs | Invalid input not handled or wrong output at boundaries | Accepts 1000 characters when only 100 allowed |
| Database Bugs | Data-related issues (storage, retrieval, etc.) | Wrong data displayed, failed data saving |
| Crash Bugs | When the app crashes during use | App closes when clicking on settings |
| Configuration Bugs | Wrong setup of software, hardware, or environments | Wrong URL in the test environment |
| Localization/Internationalization Bugs | Issues with language, currency, date/time formats, etc. | Text not translated in Spanish version |

1. **Advantages of Bugzilla.**

* **Bugzilla** is an **open-source bug tracking system** developed by Mozilla. It helps testers and developers report, track, and manage bugs or issues during software development and testing.

**Advantages of Bugzilla**

|  |  |
| --- | --- |
| **Advantage** | **Description** |
| Open Source & Free | No cost to use, which is great for small teams or start-ups. |
| Efficient Bug Tracking | Easy to report, search, and manage bugs with different statuses (New, Assigned, Resolved, etc.) |
| Customizable Fields | You can define your own workflows, fields, and bug statuses based on your project needs. |
| **Email Notifications** | Automatically sends email updates when bugs are reported, modified, or resolved. |
| Advanced Search & Filters | Powerful search features to find bugs by status, priority, assignee, etc. |
| Role-based Access Control | Allows different levels of access (admin, developer, tester, etc.) |
| Reporting & Charts | Provides graphs and reports to analyse bug trends and team performance. |
| Integration | Can be integrated with version control systems like Git and test case tools. |
| Web-Based | Can be accessed from anywhere through a web browser. |
| Comment & History Tracking | Keeps detailed history of bug changes and team communication. |