SOFTWARE TESTING ASSIGNMENT

MODULE – 2

DARSHAN GOHIL

WHAT IS 7 – KEY PRINCIPLES? EXPLAIN IN DETAIL.

- These Principles guide software testers in ensuring that testing is efficient, effective, & valuable.
- By applying them properly, teams can improve software quality, reduce risks, & enhance user satisfaction.
- There are 7 Principles as follows:
  - Testing Shows Presence of Defects.
  - Absence of Errors Fallacy.
  - Exhaustive Testing isn't Possible.
  - Early Testing.
  - Defect Clustering.
  - Pesticide Paradox.
  - Testing is Context Dependent.

7 – KEY PRINCIPLES IN DETAILS

TESTING SHOWS PRESENCE OF DEFECTS.

- The main goal of testing is to find defects, not to prove that the software is perfect.
- Even if a system passes all test cases, it doesn't guarantee that the software is bug-free.
- **Example:** Suppose you are testing an e-commerce website.
- You test different scenarios & all tests passes.
- However, later a customer finds a critical bug where applying a specific discount code leads to incorrect pricing.
- This shows that testing helped uncover defects but didn't prove that no defects exists.

ABSENSE OF ERROR FALLACY.

- A system can be bug-free but still fail if it doesn't meet user requirement.
- Testing should ensure that the software functions as expected & delivers value to errors.
- **Example:** A company develops an attendance tracking app for employees.
- The app works perfectly, with no technical bugs, but it lacks: An option to track remote employees, Mobile compatibility, A User-friendly interface.
- Employees reject the app because it doesn't meet their needs, proving that just having error-free software isn't enough.

7 – KEY PRINCIPLES IN DETAILS

EXHAUSTING TESTING ISN'T POSSIBLE.

- It isn't practical to test all possible inputs, conditions, & combinations, as this would take too much time & effort.
- Testers focus on the most common & critical scenarios.
- **Example:** Testing A Login Page.
- Different usernames
- Various password combinations.
- Browser compatibility.
- Mobile VS Desktop versions.
- Multiple network conditions.

EARLY TESTING.

- Finding defects in the requirement gathering or design phase in much cheaper than finding them after deployment.
- The cost of fixing a bug increases as the software progresses through the development lifecycle.
- **Example:** Suppose a banking application is under development.
- If a security flaw is found after deployment, fixing it may require extensive changes to the database, code, & user authentication system.
- If the same flaw is identified during the design phase, it can be fixed early with minimal cost & effort.

7 – KEY PRINCIPLES IN DETAILS

DEFECT CLUSTERING.

- A small number of modules usually contain most of the defects.
- This follows the 80/20 rule: 80% of the defects are found in 20% of the software components.
- **Example:** In food delivery app, tester find that most bugs occur in:
- Payment module.
- Order tracking feature.
- Cart system.
- Since these areas are more prone to defects, testers prioritize them for thorough testing.

PESTICIDE PARADOX.

- Running the same test cases repeatedly will eventually become ineffective.
- Bugs that were not covered in those test cases will remain undetected.
- To improve effectiveness, test cases need to be updated regularly.
- **Example:** A social media app has a feature to upload profile pictures.
- Testers repeatedly test only the standard JPEG & PNG formats.
- However, a user uploads, HEIF image, & the app crashes.
- If testers had updated their test cases with new file formats, they could have caught this issue earlier.

7 – KEY PRINCIPLES IN DETAILS

- **TESTING IS CONTEXT DEPENDENT.**
  - Different application require different testing strategies based on their purpose, audience, & risk level.
  - **Example:** Testing for a medical device software involves:
    - Strict compliance with health regulations.
    - Security & accuracy validation.
    - Extensive performance & reliability testing.
  - **Example:** Testing for a gaming app focuses on:
    - Graphics performance & rendering speed.
    - Multiplayer server stability.
    - User experience & smooth animations.

WHAT IS ERROR, DEFECT, BUG & FAILURE?

ERROR:

- An error is a mistake made by a developer while designing or coding the software.
- These mistakes occur due to misunderstanding requirements, incorrect logic, or human oversight.
- **Example:** A developer writes **X = Y/0;** which causes a divide by zero error.

DEFECT:

- A defect is a flaw in the software that causes it to behave incorrectly.
- It is found during testing before the product is released to users.
- **Example:** A Login button doesn't work as expected when clicked.

BUG:

- A bug is another term for a defect, commonly used by developers & testers.
- It is reported in bug-tacking tools like JIRA, Bugzilla, etc.
- **Example:** A mobile app crashes when a user tries to upload a profile picture.

FAILURE:

- A failure occurs when a defect escapes testing & is found by the end user.
- It happens in a real-world environment, affecting customers & business operations.
- **Example:** A banking app incorrectly transfers money to the wrong account due to an undetected defect.

CATEGORIES OF DEFECTS

CATEGORY	DESCRIPTION	EXAMPLE
Functional Defects	When the Software doesn't behave as per the functional requirements.	Login button doesn't redirect to dashboard after correct login.
Performance Defects	Issues affecting speed, load time or response under normal or peak load.	Page takes 10 seconds to load instead of 2 seconds.
Usability Defects	Poor user experience due to confusing or non-intuitive UI/UX.	No error message shown for empty required field.
Compatibility Defects	The app fails on different devices, OS or browsers.	Works on Chrome, but layout breaks on Safari.
Security Defects	Vulnerabilities that allow unauthorized access or data breaches.	A user can access admin page via direct URL.
Database Defects	Errors in data handling, storage or integrity.	Submitting a form twice creates duplicate entries.
Edge Case Defects	Failures when input goes beyond expected range or limits.	Entering 1000 characters in a 255-character field causes crash.
Logical Defects	Flaws in the underlaying logic or algorithm used.	Discount calculation is wrong due to a formula error.
Localization Defects	Incorrect translation, date, time or currency formatting for different locales.	Date format not changing for US vs. UK settings.
Installation Defects	Problems during setup or installation process.	Installer skips dependency checks & fails silently.

Difference between Verification & Validation

Verification:

- Are we building the product right?
- Ensures the software is being developed correctly as per requirements.
- Process oriented – checking documents, design, code.
- Focus on Activities like reviews, walkthroughs, inspections, static, analysis.
- Performed each development phase – requirements, design, coding.
- Testing type – static testing without executing the code.
- Done by developers, business analyst & designers.
- Checking if system design follows SRS.

Validation:

- Are we building the right product?
- Ensure the final product meets user needs & expectations.
- Product oriented – checking functionality, performance & usability.
- Focus on Functional testing, system testing, UAT.
- Performed after development, during the testing phases.
- Testing type – dynamic testing with executing the code.
- Done by testers & end-users.
- Running the software to ensure it meets customer needs.

DIFFERENCE BETWEEN QA – QC - TESTER

QA - QUALITY ASSURANCE:

- A proactive process that ensures quality in software development.
- Preventing defects by improving processes.
- Nature is process oriented.
- Activities like Process improvements, audits, documentation, training.
- Before development starts & continues throughout.
- Responsibility QA engineers, process managers.
- Defining coding standards & review processes.

QC – QUALITY CONTROL:

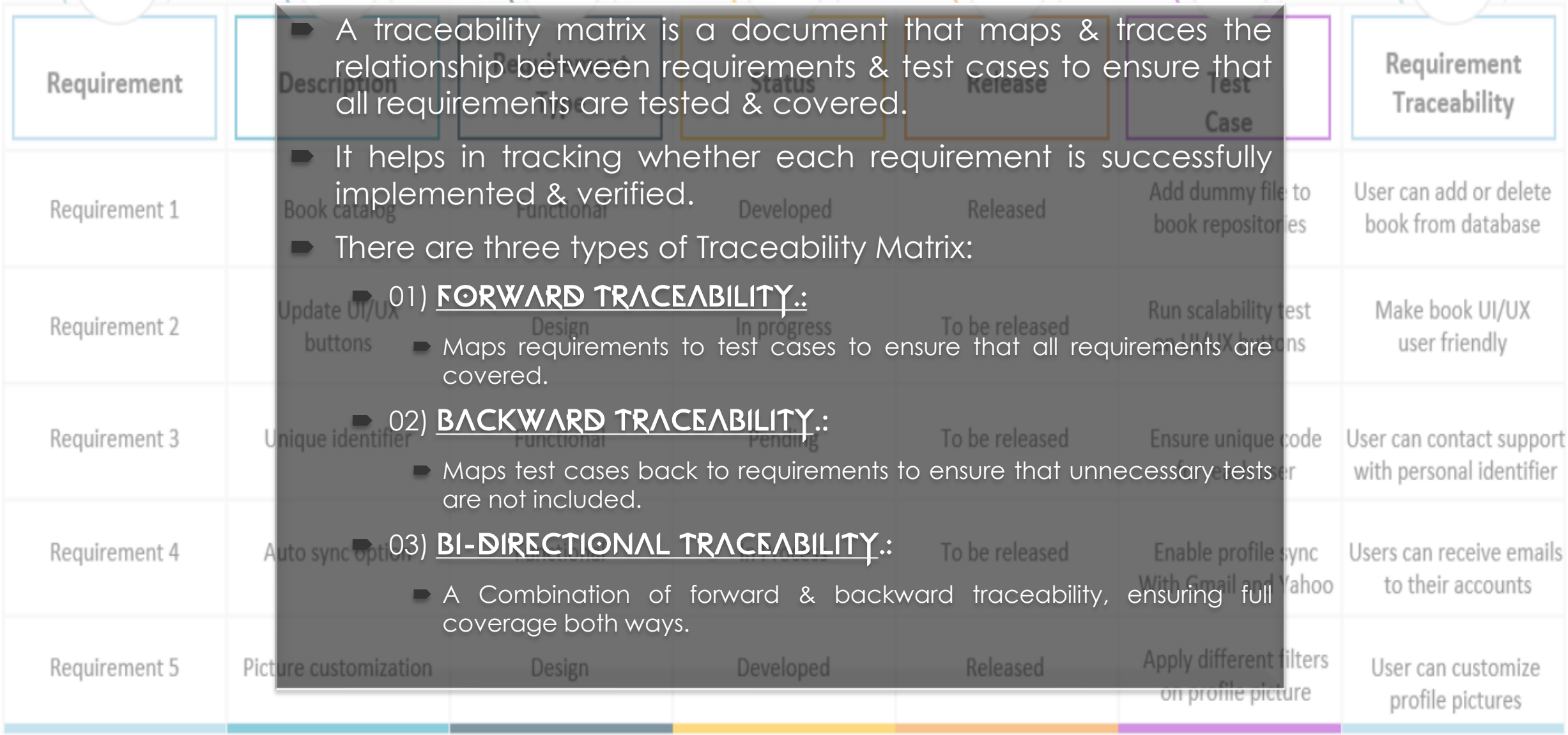
- A reactive process that checks the final product for defects.
- Identifying & fixing defects in the final product.
- Nature is product oriented.
- Activities like inspection, functional testing & regression testing.
- After development, during testing phases.
- Responsibility QC Engineers, Testers.
- Checking a final product for compliance with requirements.

TESTER:

- An individual responsible for executing tests to find defects.
- Finding bugs & verifying the software works as expected.
- Nature is task oriented.
- Activities like writing test cases, executing tests, reporting bugs.
- During testing phases – unit, integration, system, UAT.
- Responsibility software testers, QA analyst.
- Running test cases & logging defects.

WHAT IS TRACEABILITY MATRIX?

- A traceability matrix is a document that maps & traces the relationship between requirements & test cases to ensure that all requirements are tested & covered.
- It helps in tracking whether each requirement is successfully implemented & verified.
- There are three types of Traceability Matrix:
  - 01) **FORWARD TRACEABILITY.:**
    - Maps requirements to test cases to ensure that all requirements are covered.
  - 02) **BACKWARD TRACEABILITY.:**
    - Maps test cases back to requirements to ensure that unnecessary tests are not included.
  - 03) **BI-DIRECTIONAL TRACEABILITY.:**
    - A Combination of forward & backward traceability, ensuring full coverage both ways.



WHAT IS TRACEABILITY MATRIX – WITH EXAMPLE PREVIOUS MENTIONED GAME - GOD OF WAR

REQUIREMENT ID	REQUIREMENT DESCRIPTION	TEST CASE ID	TEST CASE DESCRIPTION	STATUS
R – 01	The player should be able to control Kratos' movement (walk, run, jump)	TC - 01	Verify Kratos moves as expected using controller inputs.	PASSED
R – 02	Kratos should be able to attack enemies using light & heavy attacks.	TC - 02	Verify that Kratos' attacks deal damage to enemies.	PASSED
R – 03	The Leviathan Axe should return when recalled.	TC - 03	Verify that pressing the recall button brings back the Axe.	FAILED
R – 04	The player should be able to upgrade weapons using XP.	TC – 04	Verify that spending XP upgrades weapons & abilities.	FAILED
R – 05	Health & Rage Meters should function correctly.	TC – 05	Verify that health decreases when hit & rage fills up when attacking.	PASSED
R – 06	The game should save progress at checkpoints.	TC - 06	Verify that restarting the game loads from the last checkpoint.	FAILED

WHAT DETERMINES THE LEVEL OF RISK?

- Risk in software testing refers to the possibility of a problem occurring that may impact the quality, functionality or performance of the software.
- It arises due to uncertainties in requirements, development or external factors, leading to defects, project delays or failures.
- There are two types of RISK:
  - **01) PROJECT RISK.**
  - **02) PRODUCT RISK.**

DIFFERENCE BETWEEN PROJECT RISK & PRODUCT RISK

PROJECT RISK:

- Project risks are uncertainties that affect the SDLC, leading to project delays, budget overruns or failure to meet deadlines.
- **Example:**
  - Unclear Requirements:
    - Frequent changes in requirements can delay development.
  - Lack of Skilled Resources:
    - Not enough experienced developers & testers.
  - Time & Budget Constraints:
    - Insufficient time for testing leads to defects in production.
  - Technical Challenges:
    - Using new or unstable technologies increases risk.
  - Communication Gaps:
    - Poor coordination between teams affects project delivery.

PRODUCT RISK:

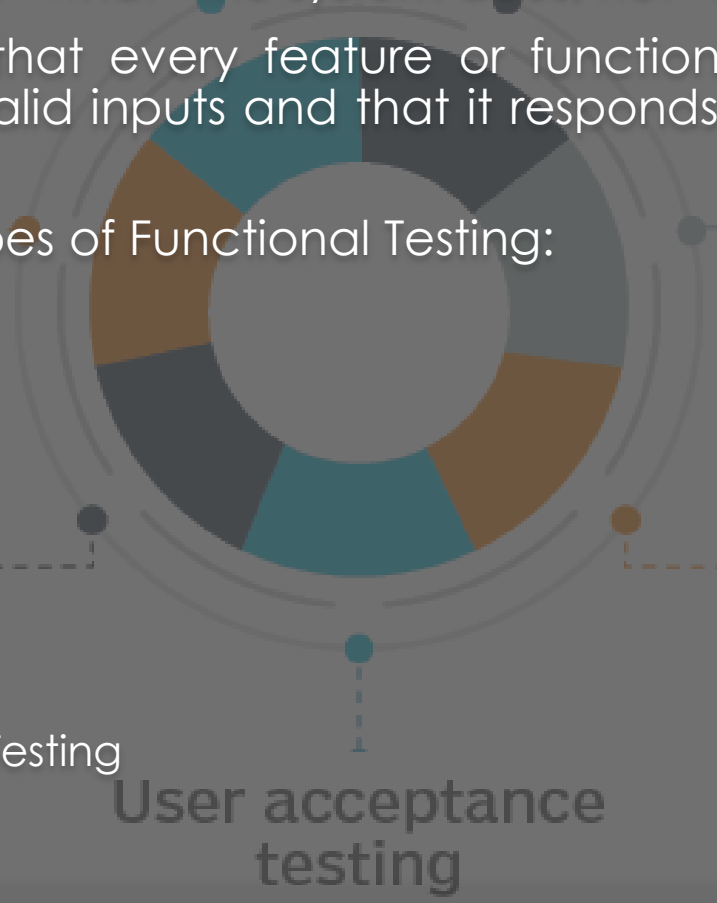
- Product risks are uncertainties that impact the software functionality, performance or usability after release.
- **Example:**
  - Critical Functionality Failure:
    - A payment gateway in an e-commerce app not working.
  - Security Vulnerabilities:
    - A bug exposing user data in a banking app.
  - Performance Issues:
    - The game "GOD OF WAR" crashing due to memory leaks.
  - Compatibility Issues:
    - The software not running properly on a different device.

WHAT IS THE PURPOSE OF EXIT CRITERIA?

- The purpose of exit criteria is to define when testing should stop b ensuring that the software meets quality standards & is ready for release or the next development phase.
- It helps prevent premature releases & ensures a smooth user experience.
- Confirms that all planned test cases have been executed.
- Ensure that no high-priority bugs are left unresolved.
- Ensure that the software meets performance & security standards.
- Ensure that half build or buggy features are not shipped.
- Ensures that the software is stable & meets user expectations.
- Detecting & fixing major bugs early prevents expensive post-release patches.

WHAT IS FUNCTIONAL TESTING?

- Functional testing focuses on verifying whether the software or application performs according to its defined requirements.
- In simple terms it checks “what” the system does, not “how” it does it.
- The goal is to ensure that every feature or function of the software behaves correctly when given valid inputs and that it responds appropriately to invalid or unexpected inputs.
- There are total eight types of Functional Testing:
  - 01) Unit Testing
  - 02) Smoke Testing
  - 03) Sanity Testing
  - 04) Integration Testing
  - 05) Black Box Testing
  - 06) White Box Testing
  - 07) User Acceptance Testing
  - 08) Regression Testing



WHAT IS COMPONENT (UNIT) TESTING?

- Component Testing (Unit Testing) is the process of testing individual components or modules of a software system in isolation to verify that they work correctly before integration with other parts.
- It is performed at the developer level before integration testing.
- It ensures that each module behaves as expected.
- Mock data or stubs may be used if a component depends on other parts of the system.
- **Example:**
  - The Previous game "GOD OF WAR", Recall the axe Function.
  - What it Does: When Kratos throws the Axe, pressing the recall button should bring it back.
  - Why Test Separately: If the recall mechanic is broken, the game will be unplayable since the axe is a core weapon.

WHAT IS COMPONENT TESTING?

- HOW IT TESTED:
  - **Through The Axe:**
    - Check if the axe lands correctly in an object or enemy.
  - **Press Recall Button:**
    - Verify if the axe returns properly.
  - **Check Animation & Sound Effect:**
    - Ensure smooth motion & correct recall sound.
  - **Verify Edge Cases:**
    - Test if the axe returns when Kratos is moving, climbing, or fighting.

WHAT IS INTEGRATION TESTING?

- Integration testing is the process of testing how different components, modules or systems work together after they have been individually tested.
- It ensures that interconnected game mechanics function properly when combined.
- It helps find bugs in interactions between different systems.
- It is performed after component testing & before system testing.
- **Example:**
  - The Previous game "GOD OF WAR", Recall the axe Function.
  - **Modules involved:**
    - Axe Recall System.
    - Combat System.
    - Enemy AI System.
    - Animation System.

WHAT IS INTEGRATION TESTING?

- HOW IT TESTED:
  - **Throw The Axe at an Enemy:**
    - Ensure the enemy gets hit and takes damage.
  - **Press Recall Button During Combat:**
    - The Axe should return while Kratos is fighting.
  - **Check Enemy Reactions:**
    - If the Axe returns through an enemy, it should deal damage.
  - **Verify Animation & Sound Effects:**
    - Kratos should have a smooth recall animation while attacking.
  - **Test Extreme Scenarios:**
    - What happens if the enemy dodges the axe return?
    - Does the axe get stuck if Kratos is climbing while recalling it?
    - Does the enemy still take damage if the axe recall is delayed?

WHAT IS BLACK BOX TESTING?

- Black box testing is a software testing technique where the tester evaluates the functionality of an application without knowing the internal code or logic.
- Instead of focusing on the how system works internally, black box testing checks whether the system produces the correct output for a given input based on its requirements.
- It is widely used for functional, usability & system testing.
- There are 6 technique of black box testing:
  - 01) Equivalence Partitioning
  - 02) Boundary Value Analysis
  - 03) Decision Tables
  - 04) State Transition Testing
  - 05) Use-case Testing
  - 06) Syntax or Pattern Testing

BLACK BOX TESTING TECHNIQUE EQUIVALENCE PARTITIONING

- In this technique where input data is divided into partitions that are expected to behave the same.
- The idea is that if one value in a group works, the others in that group will likely behave the same way – so you only need to test one value from each group, not all possible values.
- **Example:**
  - Let's say there's a form field that accepts ages from **18 to 60.**
  - We can divide the inputs into these equivalence partitions:
    - Valid Partition: 18 to 60
    - Invalid Partition 1: Less than 18
    - Invalid Partition 2: Greater than 60
  - Now we can pick just one value from each group to test:
    - From Valid: 25 (Any value between 18 to 60).
    - From Invalid Partition 1: 16
    - From Invalid Partition 2: 65
  - You don't need to test every single number, testing just one from each group is enough to catch the majority of logical input errors.

BLACK BOX TESTING TECHNIQUE
BOUNDARY VALUE ANALYSIS

- It focuses on testing the edges of input ranges.
- The idea is that bugs are more likely to occur at the boundaries of input values, rather than in the middle.
- **Example:**
  - Valid Range: **18 to 60**.
  - Lower boundary: 18
  - Upper Boundary: 60
  - Let's apply BVA.

Test Case	Value	Reason
TC – 01	17	Just Below the lower boundary (invalid)
TC – 02	18	Exactly at the lower boundary (valid)
TC – 03	19	Just Above the lower boundary (invalid)
TC – 04	59	Just Below the upper boundary (invalid)
TC – 05	60	Exactly at the upper boundary (valid)
TC – 06	61	Just Above the upper boundary (invalid)

BLACK BOX TESTING TECHNIQUE DECISION TABLE

- In this technique used to test systems that respond to a combination of inputs or conditions.
- It helps ensure that all possible input combinations and their corresponding actions (outputs) are tested.
- It is especially useful when the system behavior depends on logical rules or business decisions.
- Rows represent conditions (input) & actions (output).
- Column represent rules – combinations of input values.
- For each rule, the decision table defines what action should be taken.
- **Example:** Login System
  - Condition 1: Is the username correct?
  - Condition 2: Is the password correct?
  - Expected Action:
    - Allow or Deny Access.

- Here's how the Decision Table would look:

Rule	Username Correct	Password Correct	Action
01	Yes	Yes	Allow Access
02	Yes	No	Deny Access
03	No	Yes	Deny Access
04	No	No	Deny Access

BLACK BOX TESTING TECHNIQUE DECISION TABLE

- It is used when a system's behavior depends on its current state & the events or input that cause it to change from one state to another.
- It focuses on testing how the system transitions between different states based on inputs or actions.
- It's especially useful or systems that have finite states, like login processes, vending machines or online shopping carts.
- **Example:** ATM Machine

Current State	Input	Next State	Action
Idle	Insert Card	Card Inserted	Ask for PIN
Card Inserted	Enter Correct PIN	Pin Verified	Show Transaction Options
Pin Verified	Select Withdraw	Transaction Processing	Process Withdrawal
Any State	Remove Card	Idle	Return to Start

BLACK BOX TESTING TECHNIQUE
USE CASE TESTING

- In this technique where test cases are derived from use cases – real – world scenarios that describe how a user interacts with a system to achieve a specific goal.
- It focuses on testing the flow of the application from the user's point of view.
- This technique is particularly useful in ensuring that the system works correctly in practical usage situations, especially with multiple steps & user actions.
- **Example:** Online Shopping (Place Order Use-case)
  - Main Flow:
    - Login to account.
    - Browse product.
    - Add item to cart.
    - Go to checkout.
    - Enter address & payment info.
    - Confirm Order.
  - Alternate Flow:
    - If payment fails → Show Error message.
    - If cart is empty → Don't allow checkout.

BLACK BOX TESTING TECHNIQUE
SYNTAX OR PATTERN TESTING

- It is used to validate structured inputs against predefined rules or format.
- It checks whether the input follows the correct syntax or pattern expected by the system.
- **Example:** Email Field Validation.
  - If the system requires a valid email format like: username@domain.com
  - Syntax Would Check:
    - Is there an **"@"** symbol?
    - Is domain name present?
    - Does it end with valid extension?
  - Valid Input:
    - darshan.gohil@gmail.com
  - Invalid Input:
    - darshan.gohilgmail.com → Missing "@"
    - darshan@.com → Missing domain name.
    - @gmail.com → Missing username.

WHAT IS SMOKE TESTING?

- It checks whether the basic & critical functionalities of a system work properly after a new build or update.
- It is a quick & shallow test to ensure that the software is stable enough for deeper testing.
- If smoke testing passes, the build moves on to detailed testing.
- If it fails, the build is rejected.
- It's often called a "Build Verification Test."
- Example: Smoke Testing in GOD OF WAR game.
  - Let's say a new game build is released. A smoke test might include checking.:
    - Game launches successfully without crashing.
    - Main menu loads correctly.
    - Player can start a new game.
    - Kratos can move, jump, & attack.
    - Sound & graphic load properly.
    - Game saves & loads without error.
- If any of these basic feature fail, testers will reject the build & send it back to developers for fixing before doing deeper testing like combat mechanics, boss fight, or side quests.

DIFFERENCE BETWEEN SMOKE & SANITY TESTING?

Aspect	Smoke Testing	Sanity Testing
Purpose	To check whether the basic & critical functionalities of the app work or not.	To verify that a specific bug fix or feature change works as expected.
Level of Testing	Broad & Shallow	Narrow & Deep
Performed When	After receiving a new software build.	After receiving a bug fix or minor update in a stable build.
Covers	Entire application at a high level.	Specific module or functionality.
Goal	To determine if the build is stable for further detailed testing.	To check that specific changes didn't break anything.
Example	Open the game → check if it launches → start new game → basic menu navigation.	After fixing a bug in the inventory system → check only-inventory-related features.
Automation	Often automated.	Can be manual or automated.
Build Acceptance	Used to accept or reject the build before deeper testing starts.	Used to validate a fix without doing full regression.
Documentation	Usually well-documented & part of CI/CD pipelines.	Often quick & informal, especially in agile setups.

WHAT IS UAT TESTING?

- UAT full form is **"User Acceptance Testing."**
- It is the final phase of software testing, where real users or stakeholders test the system to make sure it works as expected in real-world scenarios.
- It answers the question: "Does this product do what the user actually needs?"
- UAT validates the functionality, usability, & performance from the user's perspective, rather than just from a technical standpoint.
- **UAT Process:**
  - A select group of users.
  - QA team members who simulate real user behavior.
  - Product managers checking feature alignment with requirements.
- **Expected Outcome:**
  - Understand the system easily.
  - Don't experience bugs or confusion.
  - And feel the feature adds value to software.
- Then the feature passes UAT and is ready to go live.

WHAT IS REGRESSION TESTING?

- It is a type of software testing that ensure that new code changes haven't accidentally broken any existing functionality in the software.
- Whenever a new feature is added, a bug is fixed, or code is modified- regression testing helps confirm that everything that used to work still works correctly after those changes.
- **Purpose:**
  - Catch unexpected side effects of code changes.
  - Make sure existing features continue to work properly.
  - Maintain software stability after updates.
- **How it works:**
  - Identify impacted areas:
    - Based on the code change, determine what parts of the system could be affected.
  - Run old test case again:
    - Especially ones that were previously passing.
  - Automate when possible:
    - Regression tests are often automated for efficiency & speed.
  - Analyze results:
    - If something that used to work now fails, developers investigate.

WHEN SHOULD REGRESSION TESTING PERFORM?

- Regression testing should be performed whenever changes are made to the software's codebase, to ensure that existing functionality remains intact.
- This includes after bug fixes, when new features are added, or when enhancements and optimizations are implemented.
- Even small modifications in the code can unintentionally impact other areas of the application, so regression testing helps catch those side effects early.
- It is also essential during regular testing cycles, especially in agile or iterative development, where changes happen frequently—usually at the end of each sprint or before delivering a new build.
- Most importantly, regression testing must be carried out before releasing the software to production, as it acts as a final safety net to ensure that new updates haven't broken anything that was previously working.

WHAT IS WHITE BOX TESTING?

- White box testing, also known as clear box, glass box, or structural testing, is a software testing method where the tester has full visibility into the internal structure and logic of the code.
- Unlike black box testing, which focuses only on inputs and outputs, white box testing allows the tester to examine the actual code, algorithms, and control flow.
- This type of testing is typically done by developers or testers with programming knowledge, as it involves testing individual functions, loops, conditions, and paths within the code to ensure everything works as intended.
- The main goal is to verify the internal operations and make sure all possible execution paths are covered, leading to optimized and bug-free code.
- White box testing is often used in unit testing and is especially useful for catching logical errors, unreachable code, or security vulnerabilities early in the development cycle.

WHAT IS NON-FUNCTIONAL TESTING?

- Non-functional testing is a type of software testing that focuses on how the system performs under certain conditions rather than what it does.
- Unlike functional testing, which verifies specific features and behaviors of the software, non-functional testing evaluates aspects like performance, usability, reliability, scalability, security, and compatibility.
- The goal of non-functional testing is to ensure the software system meets the required quality standards and behaves correctly under stress, load, or unusual situations.
- Non-functional testing is crucial because even if a system works functionally, poor performance or bad user experience can lead to failure in real-world use.
- There are total nine types of Functional Testing:
  - Performance Testing.
  - Load Testing.
  - Volume Testing.
  - Stress Testing.
  - Security Testing.
  - Installation Testing.
  - Penetration Testing.
  - Compatibility Testing.
  - Migration Testing.

WHAT IS PERFORMANCE TESTING?

- Performance Testing is a type of software testing that evaluates how well a system performs in terms of speed, responsiveness, stability, scalability, and resource usage under a given workload.
- It helps ensure the application delivers a smooth and reliable user experience, even when many users are using it at the same time or when it's under stress.
- Ensure the system can handle expected traffic smoothly.
- Identify performance bottlenecks like slow database queries or memory leaks.
- Measure response times, throughput, and resource usage.
- Check how the system behaves under high load or over extended use.
- Ensure it meets performance-related requirements (e.g., "page should load in 2 seconds").
- There are six types of performance testing:
  - Load Testing.
  - Stress Testing.
  - Spike Testing.
  - Endurance Testing.
  - Scalability Testing.
  - Volume Testing.

WHAT IS LOAD TESTING?

- Load Testing in Software Quality Assurance (SQA) is a type of non-functional testing that checks how a software application behaves under a specific expected load.
- The main goal is to evaluate the system's performance, stability, and response time when it is subjected to normal and peak usage conditions.
- In load testing, testers simulate multiple users accessing the system at the same time to see how it handles the traffic.
- This helps identify bottlenecks, slow responses, or performance issues before the system goes live.
- It ensures that the software can handle the anticipated number of users, transactions, or data volume without crashing or slowing down significantly.
- Load testing is commonly used in web applications, games, banking systems, and enterprise software, where user traffic is expected to fluctuate.
- It is typically performed using automated tools like JMeter, LoadRunner, or Locust.

- **Load Testing: NETFLIX APP**
- Imagine Netflix is about to release a highly anticipated show.
- The QA team needs to make sure that millions of users can stream the show at the same time without issues.
- To ensure this, they perform load testing on the Netflix platform.
- <u>What QA Team Do.:</u>
  - Simulate millions of users logging into the app simultaneously.
  - Trigger thousands of requests to play the same episode at once.
  - Test across multiple devices – smartphones, smart TVs, web browsers, etc.
  - Monitor how the server handle requests like login, browsing, and streaming.
- <u>What QA Team Check.:</u>
  - Response Time, Buffering & playback, Server performance, Error handling.

WHAT IS STRESS TESTING?

- Stress Testing is a type of non-functional testing used in Software Quality Assurance (SQA) to evaluate how a system behaves under extreme or beyond-normal conditions.
- The goal is to determine the system's breaking point and how gracefully it fails or recovers from high-pressure situations.
- While load testing checks performance under expected user traffic, stress testing goes a step further — it pushes the system beyond its limits to see what happens when resources like CPU, memory, bandwidth, or concurrent users exceed capacity.
- Call of Duty: Mobile is a fast-paced online multiplayer game played by millions of users worldwide.
- The developers want to make sure the game stays stable even during extreme conditions, such as a new season launch, a massive in-game event, or a global tournament.
- To prepare for unexpected spikes in traffic, they perform stress testing.

- **Stress Testing: CALL OF DUTY MOBILE GAME**
- <u>What QA Team Do.:</u>
  - Simulate 10s of millions of players logging in simultaneously.
  - Force massive matchmaking requests at once.
  - Generate a flood network traffic by simulating rapid firing, movement, voice chat, & killstreaks in thousands of matches & friend invites.
- <u>What QA Team Observe.:</u>
  - Do matches load successfully or get stuck on the loading screen?
  - Does the app crash or throw connection errors under pressure?
  - Can the game recover gracefully after the load is reduced?
  - Are there lag spikes, high ping, or rubber-banding during game play?

WHAT IS SPIKE TESTING?

- Spike Testing is a type of performance testing that evaluates how a system behaves when there is a sudden and extreme increase or decrease in user load.
- The goal is to see if the application can handle unexpected spikes in traffic and how quickly it recovers afterward.
- It's like simulating a surprise flash mob of users hitting your app — will it crash, slow down, or gracefully adapt?
- **Example: IPL Live Stream**
- During an IPL final match, especially when a last-over thriller is unfolding, millions of users may suddenly open the app or refresh the stream at the same time — either to catch the final moments, rewind a big shot, or join in from social media buzz.
- **What Spike Testing Would Simulate:**
  - A sudden traffic surge — like going from 500,000 to 10 million users in 1 minute
  - A massive burst of stream requests on the main match feed
  - Users hitting live, rewind, and replay functions rapidly
  - Simultaneous logins, notifications taps, and push updates right at a key moment
- **What Testers Check:**
  - Does the stream load instantly or is there buffering or failure?
  - Are live scores and commentary updating in real time?
  - Does the platform auto-scale its servers to handle the traffic?
  - Do any users get kicked out, face lag, or stream crashes?
  - How long does the system take to stabilize once the surge drops?

WHAT IS ENDUARNCE TESTING?

- Endurance Testing (also known as Soak Testing) is a type of performance testing where a system is tested under a normal or expected workload for an extended period of time to check how it behaves over the long run.
- **Example: WhatsApp**
- Simulate normal user activity over several days:
  - Sending/receiving messages
  - Making voice/video calls
  - Sending images, videos, documents
  - Creating and using groups
  - Switching networks (Wi-Fi to 4G/5G)
- Run this activity continuously for 48–72 hours on test devices

- What Testers Monitor:
  - Does the app become slower after prolonged use?
  - Are there memory leaks (e.g., app keeps consuming more RAM)?
  - Does battery drain faster over time due to background processes?
  - Do notifications stop working after hours of uptime?
  - Are media and call logs still saving and syncing correctly?

WHAT IS SCALABILITY TESTING?

- Scalability Testing is a type of performance testing used to determine a system's ability to scale up or scale down in response to increased load or demand — such as more users, data, or transactions.
- It helps assess how well your application performs when more resources (like CPU, RAM, servers, etc.) are added, or when user traffic grows over time.
- **Example: CALL OF DUTY MOBILE GAME**
- Call of Duty: Mobile is launching a new season update with a battle pass, new maps, and a limited-time Warzone mode.
- Millions of players rush to:
  - Download the update
  - Join matches
  - Access new rewards
  - Log into servers simultaneously

- What Scalability Testing Would Simulate:
  - Gradually increasing the number of simultaneous users from thousands to millions
  - Monitoring if matchmaking, map loading, and player stats syncing hold up
  - Checking if cloud infrastructure (like AWS or Azure) can auto-scale to serve more players
  - Verifying that lobbies are created quickly, even during peak times
- What Testers Observe:
  - Does server performance stay consistent as player load increases?
  - Are there delays in login, matchmaking, or profile loading?
  - Can the backend handle a surge in purchases (CP coins, skins, bundles)?
  - Is there a drop in frame rate or lag due to overloaded resources?

WHAT IS VOLUME TESTING?

- Volume Testing is a type of non-functional testing where the system is tested with a large volume of data to evaluate its performance, stability, and reliability.
- The goal is to check how well the system handles a high volume of data — such as large databases, huge file uploads, or millions of user records — and to identify any performance bottlenecks or failures caused by data size.
- **Example: GMAIL**
- Imagine a Gmail user has been using the same account for over 10 years. Over time, they've accumulated:
  - 100,000+ emails
  - Thousands of attachments (images, PDFs, videos)
  - Dozens of labels, filters, and folders
  - Archived and starred messages
  - Massive search and sort operations

- What Volume Testing Would Simulate:
  - Logging into an account with huge volumes of emails and attachments
  - Performing search operations like "from:xyz has:attachment before:2020"
  - Downloading or previewing large attachments from old emails
  - Loading the inbox and navigating through thousands of emails
  - Running bulk operations like deleting or archiving 10,000+ emails
- **What Testers Look For:**
  - Does Gmail slow down while loading or searching?
  - Is the search result accurate and fast, even with massive history?
  - Does the system crash, freeze, or time out?
  - How much memory or CPU is used when handling such a large dataset?

WHAT IS ADHOC TESTING?

- Adhoc Testing is an informal, unstructured type of software testing where testers explore the application without following any test cases or plans.
- It's usually performed spontaneously to find unexpected bugs by using creativity, intuition, and product knowledge.
- The main goal is to break the system or uncover hidden defects that structured testing might miss.
- **Example: NETFLIX APP**
  - Netflix tester finishes regular testing early and decides to perform some adhoc testing to see if they can uncover bugs that scripted tests missed.
  - They start exploring the app randomly, trying things like:
  - Rapidly switching between multiple user profiles
  - Clicking "Play" and "Pause" repeatedly during a stream
  - Changing subtitles and audio languages mid-stream several times
  - Jumping back and forth quickly between the home screen and playback screen
  - Trying to download and delete episodes at the same time
  - Logging out while a video is buffering
  - Testing offline mode by turning off Wi-Fi mid-stream

WHAT IS EXPLORATORY TESTING?

- Exploratory Testing is a hands-on, unscripted approach to software testing where testers actively explore the application to discover bugs, usability issues, or unexpected behaviors without following pre-defined test cases.
- It relies on the tester's creativity, domain knowledge, and experience to uncover hidden problems while learning about the software in real time.
- Example: **God Of War Ragnarök: Valhalla DLC**
  - The QA team is asked to explore the Valhalla DLC, which introduces new rogue lite-style mechanics, new enemy encounters, and evolving environments.
  - Instead of following strict test cases, they perform exploratory testing to uncover hidden bugs, design flaws, or gameplay issues.
- What the Tester Might Explore:
  - <u>Progression Flow:</u>
    - Enter Valhalla repeatedly to see if upgrades and buffs reset properly after death.
    - Purposely try to skip upgrades or boons to see if progression breaks.
    - Die mid-animation during a reward selection — does the game respond properly?

WHAT IS EXPLORATORY TESTING? CONT.

- Combat & Enemies:
  - Use a mix of Kratos' runic abilities, relics, and companions during chaotic fights to test animation syncing.
  - Try combining high-speed movement with heavy attacks in tight rooms — does the camera behave properly?
- Map & Environment:
  - Sprint into unreachable corners or roll into destructible walls repeatedly to check for collision bugs.
  - Pause the game in weird places like mid-cutscene or right before a gate opens.
- Unlockable and Customization:
  - Change Kratos' gear mid-run and rapidly switch weapons — does the UI update properly?
  - Equip a boon, then try to override it in a way the UI might not allow — does it crash or freeze?

WHAT IS GUI TESTING?

- GUI Testing (Graphical User Interface Testing) is a type of software testing that focuses on how the application looks and behaves from the user's perspective.
- It ensures that the visual elements — such as buttons, menus, icons, text boxes, colors, fonts, layouts, and navigation — work correctly and consistently across different devices and screen sizes.
- **Example: Instagram App.**
- The Instagram QA team is performing GUI Testing on the latest version of the app to make sure the user interface looks and behaves correctly across all devices and screen sizes.
- Buttons & Icons:
  - The heart (like) icon responds when tapped and shows the animation.
  - The camera icon opens the Story or Reels camera smoothly.
  - The send (DM) icon opens the chat screen correctly
- Visual Elements:
  - Profile pictures are displayed in the correct circular frame, not stretched or blurry.
  - The grid layout of posts on the profile page appears correctly in rows and columns.
  - All icons (like, comment, share, save) are visible and aligned properly.
- Input Fields & Navigation:
  - The search bar accepts text and shows suggestions.
  - Tapping the Explore tab loads trending content as expected.
  - The Story rings animate and show the right user stories on tap.
- Responsive Design:
  - Layout and elements adjust correctly on different screen sizes (phones, tablets).
  - No text or button is cut off or overlapping.

WHAT IS ALPHA TESTING?

- Alpha Testing is a type of internal acceptance testing performed by in-house testers (usually QA teams and developers) before the product is released to a limited external audience or to the public.
- It is one of the final testing phases done to identify bugs, usability issues, and stability problems in a controlled environment.
- It is conducted before beta testing.
- Let's say Instagram is developing a new feature — like **"Group Stories"**, where multiple users can contribute to a single story thread. Before releasing it to the public, the Instagram development team conducts Alpha Testing.
- <u>They test the core functionality:</u>
  - Can users create a group story?
  - Can multiple users add content to it?
  - Do the privacy settings work as expected?
  - Is the UI clear and responsive?
- <u>Testers simulate different scenarios:</u>
  - What happens if someone tries to upload a video in poor network conditions?
  - What if someone deletes the group mid-story?
  - How does it behave on older device or OS versions?
- <u>Bugs, Glitches & Design issues are reported:</u>
  - UI overlap when switching between group members.
  - Story upload fails silently in some cases.
  - Privacy setting not syncing across members.

WHAT IS BETA TESTING?

- Beta Testing is a type of external user acceptance testing where a selected group of real users uses the product in a real-world environment before the final release.
- It follows Alpha Testing and helps gather real user feedback on the product's performance, usability, and any remaining bugs.
- Previous Example of Instagram Group Stories.
- Feature Rollout:
  - A limited number of users (maybe 10,000 to 50,000) across different regions get access to "Group Stories."
  - These users are informed they are using a beta version and may experience bugs.
- User Behavior in Real Life:
  - Friends use the feature during events (like weddings or college parties) to create and contribute to shared stories.

- User Behavior in Real Life:
  - Friends use the feature during events (like weddings or college parties) to create and contribute to shared stories.

- Users test:
  - Adding photos/videos
  - Setting privacy rules (e.g., who can add/view)
  - Leaving or deleting the group
  - Receiving notifications for updates
- Feedback Collected:
  - One user reports that videos added to the group story don't always sync properly across all contributors.
  - Another notices that story contributors aren't notified when someone comments.
  - Some suggest a "highlight" option for the best moments in the group story.
  - Others complain about slow loading on older devices or poor networks.

WHAT IS BIGBANG TESTING?

- Big Bang Testing is a software testing approach where all components or modules of an application are integrated together at once, and then tested as a complete system in one go — without performing integration testing on smaller parts first.
- It's like building an entire machine, turning it on for the first time, and hoping everything works together.
- **Example: WhatsApp**
- Let's say the WhatsApp development team has been working on different modules independently:
  - Voice/Video Call Module
  - Chat Messaging Module
  - File Sharing Module
  - End-to-End Encryption Module
  - Contacts and Group Management Module
- The team waits until all the modules are complete, integrates them all at once, and then tests the entire app as a whole.

- **What Testers Might Do:**
- Start a group chat, send messages, make a voice call, and share media — all in one session.
- Check whether switching from a call to messaging works smoothly.
- Test if shared files are properly encrypted and received.
- Try adding and removing contacts while using all other features.
- Look for conflicts or bugs caused by interactions between modules.
- **Possible Issues Found:**
- Sending a file during a call causes the app to crash.
- Group management features don't update properly when media is shared.
- Encryption keys fail to sync when switching between devices.
- Video call quality drops when a large file is being uploaded in chat.

WHAT IS BUG LIFE CYCLE?

- The Bug Life Cycle, also known as the Defect Life Cycle, is a detailed process that describes the stages a software bug goes through from the moment it is discovered until it is completely resolved and closed.
- This cycle ensures that defects are properly tracked, managed, and addressed throughout the software development and testing phases, promoting effective communication between testers and developers and improving the overall quality of the software.
- The life cycle begins when a tester identifies a defect during testing and logs it into a bug tracking system.
- It is assigned to a developer for further analysis and resolution.
- After the developer has applied a fix, this does not mean the process is complete; the bug now goes back to the testing team for "Retesting."
- During this phase, testers check if the reported issue has been genuinely resolved and if the fix hasn't caused any side effects or new issues.
- If everything works as expected & the fix is confirmed, the bug is marked as "Closed," signifying the end of its life cycle.
- **THE CYCLE AS FOLLOWS:**
  - **NEW → ASSIGNED → OPEN → FIXED → RETEST → VERIFIED → CLOSED.**
  - **RETEST → REOPENED → FIXED → RETEST AGAIN… UNTIL → CLOSED.**

DIFFERENCE – PRIORITY & SEVERITY

ASPECT	PRIORITY	SEVERITY
Definition	Indicates how soon the defect should be fixed.	Indicates how serious the defect is in terms of system functionality.
Focus	Focuses on the urgency to fix the bug.	Focuses on the impact of the bug on the system.
Set By	Usually decided by the project manager or client.	Usually decided by the tester or QA team.
Related To	Business impact or customer need.	Technical impact on the application.
Change Frequency	Can change depending on release deadlines or customer needs.	Less likely to change unless the impact changes.
Example (High)	Issue found just before release – High Priority.	A crash on login page – High Severity.
Example (Low)	On a login page everyone use – High Priority.	Minor UI misalignment – Low Severity.

DIFFERENCE – SDLC & STLC

ASPECT	SDLC – SOFTWARE DEELOPMENT LIFE CYCLE	STLC – SOFTWARE TESTING LIFE CYCLE
Definition	Entire process of software development from planning to deployment.	Process focused specifically on software testing.
Scope	Covers the complete development cycle including testing.	Limited to testing & quality assurance activities.
Purpose	To develop a functional software product.	To ensure the software product is defect-free & meets quality standards.
Phases	Requirements → Analysis → Design → Implementation → Testing → Maintenance.	Requirement Analysis → Test Planning → Test Design → Test Execution → Test Closure.
Involves	Business analyst, Developers, Testers, Project Managers, Clients.	QA Testers, Test Leads, Test Managers.
Starts When	As soon as the business requirement is gathered.	After the software requirements are finalized.
Ends When	After the product deployed & enters in maintenance.	After testing is completed & a test closure report is submitted.
Main Deliverables	Functional software product, design docs, codebase, user manual.	Test plans, test cases, bug reports, test summary reports.
Dependency	Independent of STLC but includes testing as a phase.	Dependent on SDLC phases for input like requirement & code.

WHAT IS TEST PLAN? WHAT IS THE INFORMATION SHOULD BE COVERED?

- A Test Plan is a formal document in Software Testing that outlines the strategy, objectives, resources, schedule, and scope of testing activities for a software project.
- It serves as a blueprint for the entire testing process, ensuring that all stakeholders are aligned on how testing will be carried out, what will be tested, who will test it, and when it will happen.
- The main purpose of a test plan is to define the testing approach clearly and systematically so that the team can deliver a quality product that meets the requirements.
- It helps manage and control testing efforts, reduce risks, and ensure that defects are identified and fixed before the product goes live.
- **Key Information Covered in a Test Plan:**
- **Test Plan Identifier**
  - A unique ID or name to reference the test plan document.
- **Introduction**
  - A brief overview of the software being tested, along with background details and objectives of testing.
- **Test Items**
  - The features, modules, or components that will be tested.

WHAT IS TEST PLAN? WHAT IS THE INFORMATION SHOULD BE COVERED?

- **Features to be Tested**
  - A detailed list of functionalities or areas that will undergo testing.
- **Features Not to be Tested**
  - Specifies what is **out of scope** for the current testing cycle.
- **Test Objectives**
  - The overall goals of the testing process (e.g., to validate login functionality, verify performance under load).
- **Test Strategy / Approach**
  - Describes the type of testing to be performed (e.g., manual or automation, black-box or white-box, regression testing, etc.).
- **Test Deliverables**
  - Documents and outputs that will be produced during testing (e.g., test cases, test scripts, defect reports, test summary report).
- **Test Environment**
  - Details about the hardware, software, network, and tools needed to perform the testing.

WHAT IS TEST PLAN? WHAT IS THE INFORMATION SHOULD BE COVERED?

- **Roles and Responsibilities**
  - Who will be involved in the testing activities and what their roles are (e.g., testers, test leads, developers).
- **Schedule / Milestones**
  - Timeline for each phase of testing, including planning, execution, bug fixing, and closure.
- **Entry and Exit Criteria**
  - Entry: Conditions that must be met to start testing.
  - Exit: Conditions to be met before testing can be considered complete.
- **Risk and Contingencies**
  - Possible risks that may impact testing (e.g., resource unavailability, environment issues) and how to mitigate them.
- **Approval and Sign-off**
  - Names and signatures of stakeholders who approve the plan.

DIFFERENCE – TEST SCENARIOS, TEST CASES & TEST SCRIPT

ASPECT	TEST SCENATIO	TEST CASE	TEST SCRIPT
Definition	A high level description of a testable condition or situation.	A detailed document specifying steps, inputs & expected results for testing specific feature.	A set of automated instructions or code that executes test cases.
Purpose	To identify what needs to be tested.	To provide structured, detailed approach for testing specific functionality.	To automate the execution of test cases or scenarios.
Level of Detail	High – Level, general.	Detailed, with specific steps, inputs & expected outcomes.	Highly detailed, with actual code or command to run tests.
Scope	Broad; can cover multiple functionalities or features.	Focused on a specific feature or functionality.	Specific to the automation tool & languages used.
Example	"Test the login functionality for valid credentials."	"Enter valid username & password & verify successful login."	"Script that inputs a valid username & password into the login form, clicks submit & checks the response."
When to use	Early in the test planning process, to define broad area of testing.	During the test design phase to provide clear steps for testers.	When automating the testing process for efficiency.
Focus	What is being tested.	How it's being tested.	Execution of test cases through automation tools.

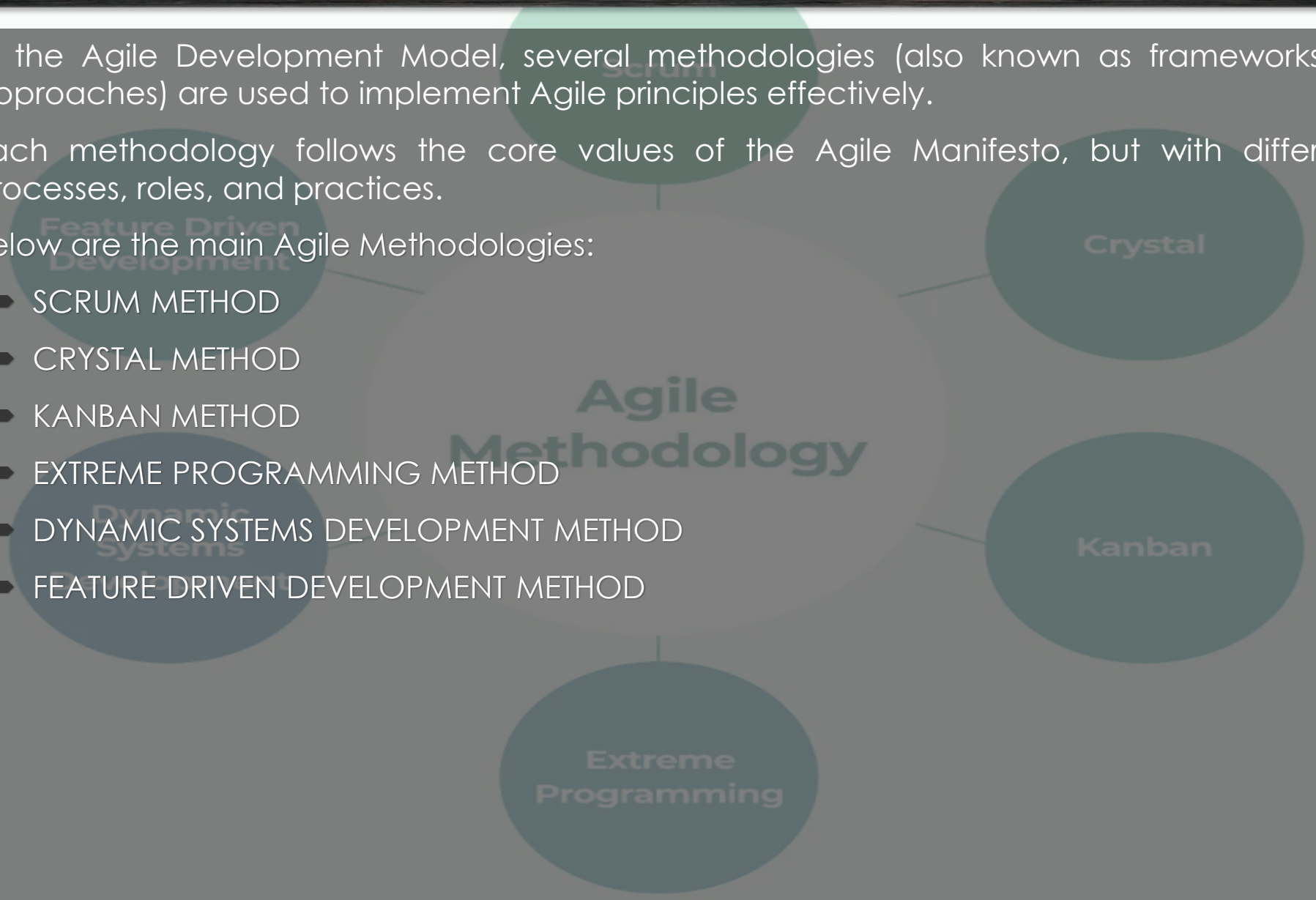
ADVANTAGES OF BUGZILLA SOFTWARE

Bugzilla is one of the most widely used open-source bug tracking systems. It was developed by Mozilla and is trusted by organizations worldwide for managing software defects efficiently.

ADVANTAGE	DESCRIPTION
Open Source & Free	It is completely free to use & open-source, with no licensing costs.
Powerful Bug Tracking	Allows detailed bug tracking with fields like severity, priority, status & history.
Advanced Search & Filters	Offers complex query capabilities with filters & saved searches for quick access.
Email Notifications	Send automatic alerts on bug updates, assignments or comments to keep the team informed.
Customisable Workflow	Users can define custom fields, statuses & workflows based on project requirements.
Role-Based Access Control	Supports user roles & permissions to ensure secure & role-specific access.
Time Tracking Feature	Enables tracking of estimated time, time spent & deadlines to manage project timelines.
Reporting & Charts	Generates visual reports & charts for better bug analysis & progress tracking.
Integration Capabilities	Can be integrated with other tools like version control systems & testing tools.
Active Community Support	Backed by a strong community & regular updates for bug fixes & new feature.

METHODOLOGIES IN AGILE MODEL.

- In the Agile Development Model, several methodologies (also known as frameworks or approaches) are used to implement Agile principles effectively.
- Each methodology follows the core values of the Agile Manifesto, but with different processes, roles, and practices.
- Below are the main Agile Methodologies:
  - SCRUM METHOD
  - CRYSTAL METHOD
  - KANBAN METHOD
  - EXTREME PROGRAMMING METHOD
  - DYNAMIC SYSTEMS DEVELOPMENT METHOD
  - FEATURE DRIVEN DEVELOPMENT METHOD



AGILE MODEL METHOD - SCRUM

- Scrum is a lightweight, iterative, and incremental framework used to manage complex software and product development.
- It is based on Agile principles and helps teams deliver high-value products in short development cycles called sprints. Scrum promotes transparency, inspection, and adaptation throughout the development process.
- Scrum divides work into time-boxed iterations called sprints (typically 1–4 weeks). Each sprint delivers a potentially shippable product increment.
- Scrum emphasizes team collaboration, continuous feedback, and customer satisfaction.
- EXAMPLE:
  - Imagine a team building a mobile app.
  - The product owner gathers user requirements and prioritizes them in the Product Backlog.
  - Every two weeks, the team selects a few high-priority features (like login, user profile, etc.) in Sprint Planning.
  - They work on them, meet daily in stand-ups, and at the end of the sprint, they showcase the working app. Feedback is taken, improvements are discussed, and the cycle repeats.

AGILE MODEL METHOD - KANBAN

- Kanban is an Agile methodology that focuses on visualizing workflow, limiting work in progress (WIP), and maximizing efficiency.
- Unlike Scrum, Kanban doesn't have fixed sprints or roles—instead, it aims for continuous delivery of tasks with a smooth and flexible workflow.
- Originally developed by Toyota for lean manufacturing, Kanban is now widely used in software development, IT operations, and many other industries.
- Kanban does not define specific roles like Scrum (e.g., Scrum Master, Product Owner). Teams may use existing roles or define their own based on need.
- The focus is on collaboration and shared responsibility.
- EXAMPLE:
  - Imagine a support team using Kanban.
  - Each incoming ticket is added to the To Do column.
  - A developer picks a task and moves it to In Progress, then to Review after testing, and finally to Done.
  - The WIP limit ensures only 2 tickets can be worked on at once, preventing overload.
  - Metrics like cycle time help improve performance.

AGILE MODEL METHOD - CRYSTAL

- Crystal is a family of Agile methodologies designed to be flexible and adaptive to different types of projects.
- It emphasizes the people, interactions, and processes involved in software development rather than rigid tools or techniques.
- Created by Alistair Cockburn, one of the original authors of the Agile Manifesto, Crystal recognizes that each project is unique, and therefore, the method used to develop it should also be tailored accordingly.
- Crystal believes that the skills, communication style, and team dynamics are more critical to success than any fixed set of rules or practices.
- EXAMPLE:
  - A small startup with a team of 5 developers building a mobile app might use Crystal Clear.
  - They work in the same office, release updates every 2 weeks, and hold informal reviews and retrospectives.
  - They value communication over heavy planning, and adjust their process as needed.

AGILE MODEL METHOD – EXTREME PROGRAMMING METHOD - XP

- Extreme Programming (XP) is an Agile software development methodology focused on producing high-quality software and improving responsiveness to changing customer requirements.
- XP encourages frequent releases in short development cycles, which improves productivity and introduces checkpoints at which new customer requirements can be adopted.
- EXAMPLE:
  - Let's say a team is building an online shopping website. Using XP:
  - They write unit tests first before coding a feature like "Add to Cart."
  - Developers work in pairs to write and review code.
  - The code is integrated and tested multiple times a day.
  - Features are released weekly for customer feedback.
  - The team refactors the code regularly to ensure it stays clean and maintainable.

AGILE MODEL METHOD – DYNAMIC DEVELOPMENT METHOD

- It is an iterative and incremental Agile method that emphasizes active user involvement, frequent delivery, and collaborative decision-making.
- It provides a comprehensive framework that covers project management as well as development, making it suitable for larger and more complex projects.
- EXAMPLE:
  - Suppose a government agency is developing a citizen portal. Using DSDM:
  - A feasibility study is conducted first.
  - Business and functional requirements are discussed with users.
  - Timeboxes are set for iterations, and work is prioritized using MoSCoW.
  - A working prototype is released in each iteration for feedback.
  - Final implementation is done with proper testing, training, and rollout.

AGILE MODEL METHOD – FEATURE DRIVEN DEVELOPMENT METHOD

- Feature Driven Development (FDD) is an Agile methodology focused on building and delivering software features frequently and efficiently.
- It emphasizes short iterations, regular progress tracking, and design before development.
- Deliver tangible, working software repeatedly in short timeframes.
- Focus on features that are valuable to the client.
- Encourage modular design and team collaboration.
- Maintain consistency and visibility of project progress.
- EXAMPLE:
  - A fintech company building a loan management system uses FDD.
  - The system is modeled upfront with input from domain experts.
  - A long list of features like “approve loan request” or “calculate EMI” is created.
  - These features are assigned to developers, built, reviewed, and integrated iteratively.
  - Progress is tracked feature by feature.

Difference – Authentication & Authorization in Web Testing

ASPECT	AUTHENTICATION TESTING	AUTHORIZATION TESTING
Goal	Ensure the system correctly verifies user identity.	Ensure that users can access only what they're allowed to.
Test Focus	Login Functionality, credential validation, session management.	Access control, permission validation, role-based restrictions.
What is Tested?	Login page behavior, Password encryption, Multi-factor auth, Session timeout.	User roles (admin, guest, user), Page level or API level restrictions, Direct URL access.
Example Test Case	Valid username / password allows login Invalid credentials block access.	Admin can access settings. Guest can't open / admin URL.
Tools Used	Postman, Selenium, Jmeter, Burm Suite (for session testing)	Postman, , Selenium, OWSAP ZAP, Custom test scripts.
Common Issues Found	Weak password policies, Session fixation, Broken login flows.	Broken access control, Privilege escalation, Insecure direct object references (IDOR).
Tester's Perspective	Can the user login securely?	Can this user do something they are not supposed to?
Security Relevance	Protects identity & data from being stolen.	Protects data & features from unauthorized use.

WHEN TO USE USABLITY TESTING?

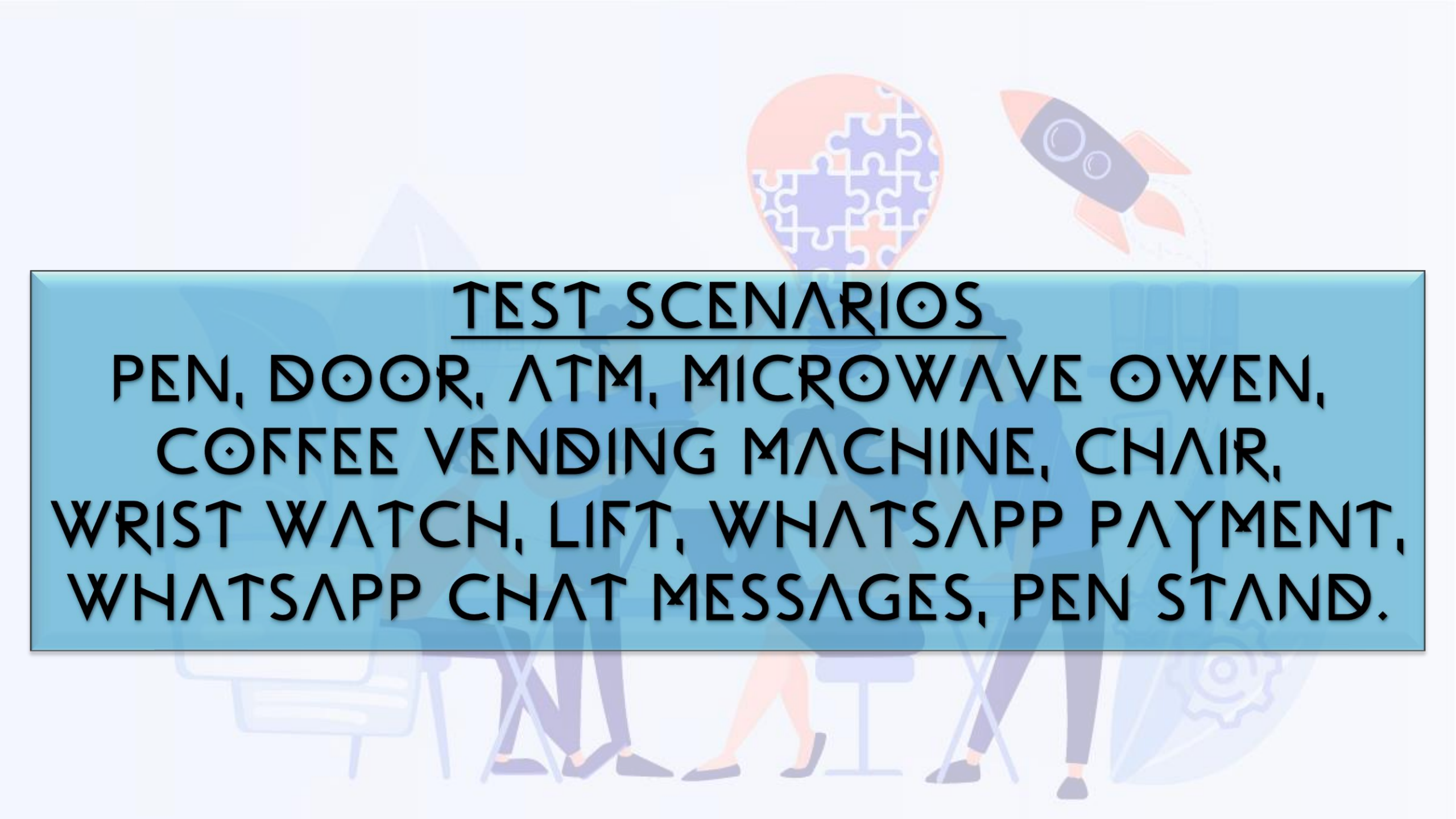
- Usability testing should be used whenever you want to evaluate how easy, intuitive, and user-friendly your product is for real users.
- It is especially valuable during the early design phase, even when you have just wireframes or low-fidelity prototypes.
- Testing at this stage helps identify navigation issues and user confusion before investing significant time and resources into full development.
- It is also crucial before a product launch to ensure that new users can complete key tasks smoothly, such as signing up, searching, checking out, or making a payment.
- Another important time to conduct usability testing is when you're receiving negative user feedback or noticing a high drop-off rate in analytics.
- If you're expanding to a new audience, region, or demographic, usability testing ensures your product meets the needs, expectations, and cultural preferences of that group.
- Lastly, even if your product is live and stable, usability testing is a key part of continuous improvement.

WHAT IS THE PROCEDURE FOR GUI TESTING?

- The procedure for GUI (Graphical User Interface) testing begins with a thorough understanding of the application's requirements and design specifications.
- Testers need to review the UI/UX documents to grasp the purpose and expected behavior of each interface element and user workflow.
- Once the requirements are clear, the next step is to set up the testing environment, which includes installing the application on the relevant platforms such as web browsers, mobile devices, or desktops, and preparing any required GUI testing tools like Selenium, TestComplete, or Appium.
- With the test cases in place, the next step is executing the tests, either manually or through automation.
- During this phase, testers interact with the GUI elements to verify if they perform as expected and look consistent with the design.
- Any bugs or issues discovered during testing are logged with detailed information including screenshots, environment details, and steps to reproduce the issue.
- Once the development team fixes the reported issues, testers perform re-testing to verify the fixes and ensure the problems no longer exist.

What is the procedure for GUI testing? cont.

- Regression testing is also conducted to confirm that the fixes haven't introduced new issues elsewhere in the GUI.
- Finally, a comprehensive review of the interface is done to ensure that it meets both visual and functional expectations.
- If everything is satisfactory, the GUI testing process is concluded with a formal sign-off.



TEST SCENARIOS

PEN, DOOR, ATM, MICROWAVE OWEN, COFFEE VENDING MACHINE, CHAIR, WRIST WATCH, LIFT, WHATSAPP PAYMENT, WHATSAPP CHAT MESSAGES, PEN STAND.

TEST SECNARIOS - PEN

- 01) Verify the pen type.
- 02) Verify the material used to make pen.
- 03) Verify the pen dimensions as per the mentioned in document.
- 04) Verify the pointer size like 0.5mm or 0.7 mm.
- 05) Verify the pen ink is not going to leak when the pen is in the reverse direction of the pointer.
- 06) Verify that the pan can write in any kind of paper (not with the layer of thin polyethene).
- 07) Verify the ink is not spreading while writing on paper.
- 08) Verify the ink is not going to be jammed if the pen is ideal for 2 to 3 days.
- 09) Verify the Ink is not going to be dry if pen is open for 2 to 3 long hours.
- 10) Verify that pen is working in angle of writing or not.

TEST SECNARIOS - DOOR

- 01) Verify the type of door. (Metal, Wooden, Aluminum, Glass).
- 02) Verify the door dimensions as per the details mentioned in specification.
- 03) Verify the door is single side openable or both side.
- 04) Verify that how many hinges are going to fix in the door.
- 05) Verify the size of hinges.
- 06) After fixing door verify that spacing between door and wall (floor & top wall) is clear so that door can move easily without jamming.
- 07) Verify the aldrop & handle is fixing properly.
- 08) Verify that aldrop & handle is fixing alternate to each other.
- 09) Verify the hole are the same diameter as alsrop's diameter to lock the door.
- 10) Verify at bottom of door the stop-clip is fixed properly.
- 11) Verify that stop-clip is stopping the door from closing.

TEST SECNARIOS - ATM

- 01) Verify the type pf ATM (Touch screen, both keypad buttons or both).
- 02) Verify the ATM can accept different ATM cards of different bank.
- 03) Verify that card is properly inserted or not, if inserted properly the show menu, if not show an error.
- 04) Verify that ATM screen is smooth & operational.
- 05) Verify that in screen user can choose different languages.
- 06) Verify that user can input pin number, if details are correct then go for next process, if wrong ATM coding allows only limited attempt of input pin number.
- 07) Check the pin shown in display is hidden always.
- 08) Verify that user can choose account type (savings, current, etc.).
- 09) Verify that user can check the balance of account.
- 10) Verify that user can't withdraw more amount than total available balance.
- 11) Verify if ATM is out of money it shows proper message.
- 12) Verify that user can get receipt of withdrawal.
- 13) Verify that user cant do any process with expired card.

TEST SECNARIOS - CHAIR

- 01) Verify the type of chair (Plastic, Fiber, Meta).
- 02) Verify the dimensions of chair meets the mentioned specification.
- 03) Verify that chair is comfortable for user.
- 04) Verify the load analysis on chair is perfect or not.
- 05) Verify that the weight of chair is mentioned in specification.
- 06) Verify that hand support of chair is comfortable or not.
- 07) Verify that cushion is provided on chair or not.
- 07) If cushion is applied then check the cushion is fixed with seat base of chair.
- 08) Verify that cushion is get affected by the water or any liquid.
- 09) Verify that cover of cushion is comfortable to user.
- 10) Verify that cover is perfectly fixed with cushion.

TEST SECNARIOS
COFFEE VENDING MACHINE

- 01)) Verify the machine powers on & working properly.
- 02) Verify user can select a coffee options.
- 03) Verify coffee is dispensed after selection.
- 04) Verify the machine stops dispensing once the quantity is reached.
- 05) Verify machine accepts various payment method.
- 06) Verify user can choose sugar level.
- 07) Verify user can choose milk level.
- 08) Verify machine shows alert when water is low, milk is low.
- 09) Verify machine shows alert if coffee beans are low.
- 10) Verify machine shows alert when some of the ingredients are empty.

TEST SECNARIOS – WRIST WATCH

- 01) Verify the type of wrist watch. (Analog or Digital.)
- 02) Verify the display is showing perfect time.
- 03) Verify that user can set manually time.
- 04) verify the dimensions of wrist watch as per specification.
- 05) Verify that belt are using for wrist watch it has multiple holes (if leather belt).
- 06) Verify that pins are attached with belt and watch machine body is perfect or not.
- 07) Verify that watch is waterproof or not.
- 08) Verify that user can watch time in dark (specially in analog).
- 09) Verify that watch is battery operated or chargeable battery operated.

TEST SECNARIOS – LIFT (ELEVATOR)

- 01) Verify lift material as per specification.
- 02) verify the rope wire quality & quantity is as per specification or not.
- 03) Verify lift can go up & down when floor number are selected.
- 04) Verify that lift can stop at right time at right floor perfectly.
- 05) Verify if any door is opened lift can't operate.
- 06) Verify that lift can accept only limited person (3 to 5 person).
- 07) Verify lift can show alert if number of person is exceeded.
- 08) Verify lift motor is working properly.
- 09) Verify ground floor lift stoppage spring is properly fixed or not.
- 10) Verify the fan and light switch is working properly or not.
- 11) Verify that during lift movement door can't be open automatically or manually.

TEST SECNARIOS – WHATS APP PAYEMNT

- 01) Verify user can access the Payments option in WhatsApp settings.
- 02) Verify phone number is verified via OTP during setup.
- 03) Verify correct bank is fetched using the phone number linked with it.
- 04) Verify UPI PIN is set or entered correctly.
- 05) Verify user can enter any amount within the allowed limit.
- 06) Verify UPI PIN prompt appears before sending money.
- 07) Verify failed transactions display a proper error message.
- 08) Verify notification is received after successful payment.
- 09) Verify user can view all past transactions in the Payments section.
- 10) Verify payment via QR code scans correctly.
- 11) Verify WhatsApp doesn't crash if the bank server is temporarily unavailable.

TEST SECNARIOS – MICROWAVE OWEN

- 01) Verify the microwave powers on when plugged in.
- 02) Verify the clock can be set correctly.
- 03) Verify user can set cooking time manually.
- 04) Verify microwave starts cooking when the start button is pressed.
- 05) Verify user can set power levels (low, medium, high).
- 06) Verify cooking ends automatically after the set time.
- 07) Verify beep sounds at the end of cooking.
- 08) Verify cooking pauses when the door is opened mid-cycle.
- 09) Verify microwave doesn't start if the door is open.
- 10) Verify door opens and closes smoothly.
- 11) Verify microwave beeps or shows message if there's a malfunction.

TEST SECNARIOS – WHATSAPP CHAT MESSAGES

- 01) Verify user can send a text message to an individual contact.
- 02) Verify user can receive a text message from a contact.
- 03) Verify sent message shows single tick (✓) when sent.
- 04) Verify double tick (✓✓) appears when message is delivered.
- 05) Verify blue double tick appears when message is read (if read receipts are on).
- 06) Verify message shows clock icon if sent while offline.
- 07) Verify user can send and receive images, videos, documents, and audio.
- 08) Verify media preview is shown before sending.
- 09) Verify user can stop media download mid-way.
- 10) Verify messages are end-to-end encrypted.
- 11) Verify user can delete a message for everyone (within time limit).
- 12) Verify "This message was deleted" placeholder appears after deletion.
- 13) Verify user receives notification when a new message arrives.
- 14) Verify no notification is received in muted chats.

TEST SECNARIOS – PENSTAND

- 01) Verify the pen stand can hold standard-sized pens and pencils.
- 02) Verify it can hold multiple pens without tipping over.
- 03) Verify items can be easily inserted and removed.
- 04) Verify the base is stable on flat surfaces.
- 05) Verify the height is sufficient to support long pens without falling.
- 06) Verify it fits comfortably on a standard desk or table.
- 07) Verify the pen stand can be cleaned easily with a cloth or water.
- 08) Verify it doesn't retain stains or ink marks easily.
- 09) Verify it can hold various stationery items like markers, highlighters, scissors, rulers, etc.
- 10) Verify the base is solid and doesn't wobble.
- 11) Verify it doesn't bend or break under regular use.
- 12) Verify the pen stand is made of durable material (plastic, wood, metal, etc.).

TEST CASES & HLR LINKS

INSTAGRAM FIRST PAGE HLR & TEST CASE

WHATSAPP WEB HLR & TEST CASE

INSTAGRAM CHAT FUNCTIONALITY HLR & TEST CASE

GMAIL COMPOSE MAIL FUNCTIONALITY TEST CASE

FACEBOOK FIRST PAGE HLR & TEST CASE

FLIPKART BUY PRODUCT TEST CASE

FACEBOOK CHAT FUNCTIONALITY HLR & TEST CASE

WHATSAPP GROUP CHAT FUNCTIONALITY TEST CASE