**ISTQB Certification**

**What is software:**

--is a collection of computer programs that helps us to perform a task.

**Types of software:**

1. **System software:**

This type of software is responsible for managing and controlling the computer hardware and providing a platform for other software applications to run.

e.g device drivers, OS, servers, utilities etc

**2. Programming software:**

e.g compilers, debuggers, interpreters

**3. application software:**

Application software is designed to perform specific tasks or provide functionality to end-users.

e.g Industrial automation, business software, games, telecoms

**Project:**

If software application is developed foe specific customer requirements then it is called

**Product:**

If software application is developed for multiple customer requirements then it is called product.

**Software testing:**

--is a part of software development process

-- is an activity to detect and identify the defects in the software

-- objective of testing is to release quality product to client.

**Why Software testing needed:**

* Software testing is an essential part of the software development life cycle.
* It involves the process of evaluating a software system or application to identify defects, errors, or issues and ensure that it meets the specified requirements and quality standards.

1. Finding defects:
2. Ensuring quality and reliability
3. Enhancing user experience
4. Increasing software robustness
5. Compliance and industry standards
6. Cost and time efficiency

**What is software Quality:**

**Quality**: is defined as justification of all the requirements of a customer in a product.

**Quality software is reasonably:**

1. Bug free
2. Delivered on time
3. Within budget
4. Meets requirements and expectations
5. Maintainable.

**Error:**

An error, also known as a mistake or a bug, refers to a human action or decision that produces an incorrect or unexpected result in software.

**Defect:**

A defect, also referred to as a bug or a fault, is a flaw or an abnormality in the software that causes it to deviate from its expected behavior.

Defects occur when errors in software are not identified and corrected during the development process.

**Failure:**

A failure is an observable and unexpected behavior of the software when it does not perform its intended function.

It occurs when a defect is encountered during the execution of the software, resulting in incorrect or undesirable outputs.

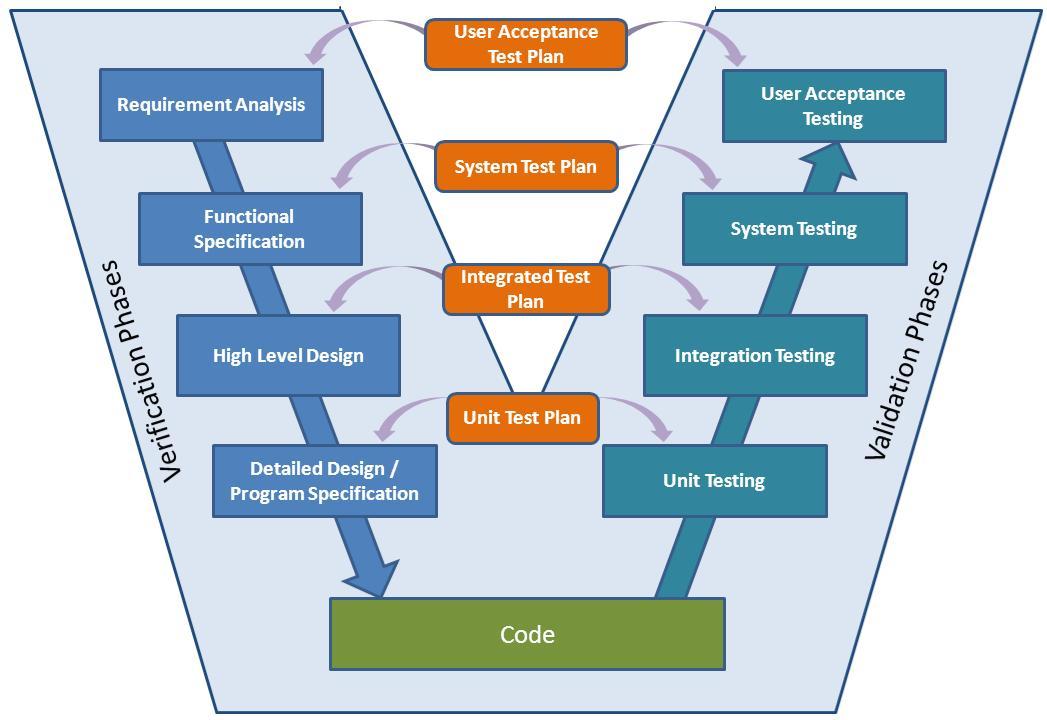
Failures are detected when the software is used or tested in real-world conditions.

**Why bugs in the software:**

1. Human Error
2. Miscommunication or Misunderstanding:
3. Complexity and Size of Software
4. Time and Resource Constraints
5. External Factors and Dependencies
6. Environment and Usage Variability

**SDLC VS STLC:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Sr no** | **parameters** | **SDLC** | **STLC** |
| **1** | **Full form** | Software Development Life Cycle) | Software Testing Life Cycle |
| **2** | **def** | SDLC is a structured approach or framework that outlines the phases and activities involved in the development of software.  It encompasses the entire process, from requirements gathering to software deployment and maintenance. | STLC is a process that defines the activities and tasks involved in testing the software to ensure its quality, functionality, and compliance with requirements. |
| **3** | **objective** | The primary objective of SDLC is to guide the development team in building software that meets the specified requirements, is of high quality, and is delivered within budget and time constraints. | The primary objective of STLC is to systematically plan, design, execute, and evaluate tests to identify defects and ensure that the software meets the desired quality standards. |
| **4** | **phases** | SDLC typically consists of phases such as requirements gathering and analysis, system design, coding/development, testing, deployment, and maintenance. | STLC typically includes phases such as test planning, test design, test execution, defect tracking, and test closure.  It may also involve activities like test environment setup, test data creation, and test result analysis. |
| **5** | **focus** | SDLC focuses on the overall development process, including activities such as requirements analysis, design, implementation, and integration.  It considers factors like project planning, resource allocation, and risk management. | STLC focuses specifically on testing activities, including test strategy formulation, test case development, test execution, defect management, and test reporting.  It aims to verify and validate the software against specified requirements. |
| **6** | **involvement** | SDLC involves various stakeholders, including business analysts, project managers, developers, testers, and end-users. | STLC involves software testers and quality assurance professionals who are responsible for planning, designing, and executing tests |
| **7** | **Model** | Waterfall or linear | V model (verification and validation model) |



**Static vs dynamic testing:**

**Static:** is an approach to test project documents in the form of reviews, walkthroughs and inspections

**Dynamic**: is an approach to test the actual software by giving inputs and observing results.

|  |  |  |  |
| --- | --- | --- | --- |
| **Sr no** | **parameters** | **Static (verification)** | **Dynamic (validatation)** |
| **1** | **def** | Static testing refers to the process of evaluating software or its documentation without executing the code | contrast to static testing |
| **2** | **involve** | It involves reviewing and analyzing the software artifacts to identify defects, errors, or issues | involves the execution of the software to observe its behavior and evaluate its performance |
| **3** | **goal** | to find problems early in the development process and prevent them from manifesting in later stages or during runtime. | focuses on verifying that the software functions correctly and meets the specified requirements during runtime. |
| **4** | **techniques** | techniques such as code reviews, walkthroughs, inspections, and static analysis.  techniques focus on examining the software's source code, requirements, design documents, and other artifacts for defects, ambiguities, inconsistencies, and adherence to coding standards. | techniques such as functional testing, performance testing, usability testing, security testing, and integration testing  assesses the software's actual behavior, responses, and outputs under different conditions, inputs, and scenarios |
| **5** | **stage** | helps identify issues at an early stage when they are easier and less costly to fix. | validates the software against the specified requirements, ensuring that it meets the desired functionality, performance, usability, security, and other quality attributes. |

**Levels of testing:**

1. **Unit Testing: white box**

* Unit testing involves testing individual units or components of the software, such as functions, methods, or classes, in isolation.
* The purpose of unit testing is to verify that each unit functions correctly and meets its specified requirements.
* Unit tests are typically written by developers and are executed frequently during the development process.
* **Unit testing techniques:**

1. basic path testing

2. control structure testing:

1. conditional coverage

2. loops coverage

3. mutation testing

1. **Integration Testing: white box**

* Integration testing verifies the interactions and interfaces between different components or modules of the software.
* It aims to uncover defects that may arise due to the integration of multiple units.
* Integration testing can be performed at different levels, such as module-level integration, subsystem integration, or system integration, depending on the complexity of the software system.

1. **System Testing: black box**

* System testing involves testing the entire software system as a whole, ensuring that all integrated components work together correctly and meet the defined requirements.
* It focuses on validating the behavior of the software in a real-world or production-like environment.
* System testing includes functional testing, performance testing, security testing, usability testing, and other types of testing to assess the system's compliance with specified criteria.

1. **Acceptance Testing: black box**

* Acceptance testing is performed to determine whether the software meets the business requirements and is acceptable to stakeholders, including end-users or customers.
* It validates that the software is ready for deployment and meets the defined acceptance criteria.
* Acceptance testing can include user acceptance testing (UAT), where end-users test the software in their own environment, or regulatory compliance testing, ensuring adherence to specific industry standards or regulations.
* **Levels in acceptance testing:**

1.alpha testing

2. beta testing

These are the primary levels of testing, but it's worth noting that some projects or organizations may have additional levels or variations based on their specific needs or development methodologies.

For example, some commonly used additional levels of testing include component testing, regression testing, performance testing, and security testing.

The selection and scope of testing levels depend on factors such as project requirements, complexity, and available resources.

Top of Form

**Testing methodology:**

**White box testing:**

* White box testing, also known as **clear box testing or structural testing**, is a testing approach where the tester has full knowledge of the internal structure, design, and implementation of the software system being tested.
* Test cases are designed based on this internal knowledge, focusing on exercising specific paths, statements, conditions, and branches within the code.
* The objective of white box testing is to ensure that the code functions correctly and that all logical paths are tested, including edge cases and boundary conditions.
* This technique is commonly used to verify the internal integrity of the code and to achieve high code coverage.

**Black box testing:**

* Black box testing is an approach where the tester has no knowledge of the internal workings of the software system being tested.
* Test cases are designed based on the system's specified requirements and functionalities, without considering the internal implementation.
* The tester treats the system as a "black box" and focuses on validating the inputs, outputs, and behaviour of the system without any knowledge of the internal code or structure.
* The objective of black box testing is to assess the system's functionality, usability, performance, and adherence to requirements from an end-user or external perspective.

**Grey box testing:**

* Grey box testing is a hybrid approach that combines elements of both white box testing and black box testing.
* In grey box testing, the tester has partial knowledge of the internal workings of the software system being tested. This limited knowledge could include access to the code, database schemas, or architectural information.
* The objective of grey box testing is to validate the functionality, performance, and security of the system while also leveraging internal knowledge to enhance the effectiveness and efficiency of the tests.

**Verification:**

* Verification is the process of evaluating a system or component to determine whether it meets specified requirements.
* It involves activities that focus on checking the design, development, and implementation of the software to ensure that it has been built correctly.
* Verification activities typically include reviews, inspections, walkthroughs, and static analysis.
* The goal of verification is to identify and fix defects, errors, or inconsistencies early in the development process, before the system is fully built or deployed.
* Verification helps ensure that the software is being developed according to the defined standards, guidelines, and requirements.

**Validation:**

* Validation is the process of evaluating a system or component during or at the end of the development process to determine whether it satisfies the specified requirements and meets the intended purpose.
* It involves activities that focus on the dynamic behaviour and performance of the software in real-world or production-like environments.
* Validation activities typically include dynamic testing, functional testing, performance testing, usability testing, and other forms of testing.
* The goal of validation is to assess the software's behaviour, functionality, performance, and compliance with user needs and expectations.
* Validation helps ensure that the software meets the intended requirements and provides value to the end-users or stakeholders.

**System testing types:**

1. GUI testing
2. Usability testing
3. Functional testing
4. Non functional testing

**1.GUI testing:**

* GUI (Graphical User Interface) testing focuses on verifying the graphical elements and user interactions within a software application's interface.
* It ensures that the user interface components, such as buttons, menus, forms, and visual elements, function correctly and provide a positive user experience.

**2.Usability testing:**

* Usability testing is a method of evaluating a software application, website, or product to determine its user-friendliness and overall user experience.
* The primary goal of usability testing is to identify any usability issues and gather feedback from real users to improve the design and usability of the product.
* It helps ensure that the final product meets user expectations, enhances user satisfaction, and increases user adoption.

**3.Functional testing:**

* Functional testing is a type of testing that focuses on verifying the functional behavior of a software system or application.
* It ensures that the system's functionalities, as defined in the requirements, are implemented correctly and perform as expected.
* Functional testing is crucial for verifying that the system meets the specified requirements and functions correctly from a user's perspective.
* By identifying and resolving functional issues early in the development lifecycle, functional testing helps ensure the overall quality and reliability of the software system.
* **Here are the key aspects of functional testing:**
  1. Requirement Analysis
  2. Test Case Design
  3. Test Execution
  4. Functional Coverage
  5. Positive Testing
  6. Negative Testing
  7. Boundary Testing
  8. Error Handling
  9. Regression Testing

**4.Non functional testing:**

* Non-functional testing focuses on evaluating the aspects of a software system or application that are not directly related to its specific functionalities but impact its overall performance, usability, security, and other non-functional attributes.
* Here are the key aspects of non-functional testing:

1. **Performance Testing(speed testing)**
2. Load Testing
3. Stress Testing
4. Volume testing

assesses how well the system performs under various conditions, such as high loads, concurrent users, or heavy data volumes

1. **Security Testing:**

* assesses the system's ability to protect data, prevent unauthorized access, and withstand security threats.
* It involves testing various aspects, such as authentication, authorization, data encryption, vulnerability scanning, and penetration testing, to identify vulnerabilities and ensure the system's security measures are effective.

1. **Usability Testing:**

* evaluates the system's user-friendliness and overall user experience.
* It focuses on factors such as ease of use, intuitiveness, navigation, clarity of instructions, and overall user satisfaction

1. **Compatibility Testing:**

* ensures that the system works correctly across different platforms, browsers, operating systems, and devices

1. **Scalability Testing:**

evaluates the system's ability to handle increasing workloads and data volumes.

1. **Reliability Testing:**

focuses on assessing the system's ability to perform consistently and reliably over an extended period.

1. **Maintainability Testing:**

assesses the ease of maintaining and supporting the system over time.

1. **Disaster Recovery Testing:**

* verifies the system's ability to recover and resume operations after a catastrophic event or system failure.
* It involves testing backup and restore procedures, data recovery mechanisms, and failover capabilities to ensure business continuity.

**Smoke testing vs sanity testing:**

both types of initial testing performed on software builds or releases.

|  |  |  |  |
| --- | --- | --- | --- |
| **Sr no** | **parameters** | **Smoke testing** | **Sanity testing** |
| 1 | **Objective** | ensure that the critical functionalities of the software are working properly and the build is stable enough for further testing. | check whether specific changes or fixes have been successfully implemented and to verify the areas of the software affected by those changes. |
| 2 | **Scope** | Smoke testing is a broad and shallow type of testing that covers the major functionalities of the software. | Sanity testing is narrow and focused, targeting specific functionalities or areas impacted by recent changes or fixes. |
| 3 | **Timing** | performed after a new build or release is deployed to check if it is stable enough for more detailed testing. | performed after a build or release to validate specific changes or fixes made to the software. |
| 4 | **Depth** | Smoke testing does not delve into extensive testing or validation of each functionality. It focuses on quickly identifying showstopper defects or major issues that would prevent further testing. | Sanity testing involves more in-depth testing than smoke testing.  It may include detailed testing of specific functionalities or modules affected by recent changes. |
| 5 | **Pass Criterion** | if the smoke test passes, it indicates that the software build is stable and can proceed to more comprehensive testing | The sanity test passes if the specific changes or fixes being tested are verified as working as intended |
| 6 | **Fail criterion** | If the smoke test fails, it suggests critical issues that need to be addressed before further testing can continue. | If the sanity test fails, it indicates that the changes or fixes have introduced issues and need to be addressed. |
| 7 | **summary** | smoke testing is a broader, high-level test to ensure overall stability | sanity testing is a narrower, focused test to verify specific changes or fixes |
| 8 | **Performed by** | Developer or tester | testers |
| 9 | **Subset of** | Accepatance testing | Regression testing |
| 10 | **Documented/**  **scripted** | done | Not done |

**Regression testing vs re-testing:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Sr no** | **parameters** | **Regression testing** | **Re-testing** |
| 1 | **objective** | validate that the software's existing functionalities continue to work as expected after changes or enhancements have been made to the system. | validate that the defects or issues reported in previous testing cycles have been successfully resolved. |
| 2 | **scope** | covers a wide range of functionalities to ensure that the changes haven't caused any unintended issues or regressions. | Retesting is narrower in scope compared to regression testing. It typically focuses on the specific areas or functionalities that were affected by the reported issues. |
| 3 | **Test selection** | Regression test cases are selected from a pre-existing test suite, including both functional and non-functional tests | selected based on the defects or issues that were previously identified and reported.  The focus is on verifying that the fixes or changes have resolved the reported problems. |
| 4 | **automation** | Regression testing often involves the use of automation tools to efficiently execute a large number of test cases and compare the actual results with the expected results. Automation helps in speeding up the regression testing process and ensuring consistent test coverage. | Retesting can involve automation, particularly when the original test cases are automated  Automated retesting ensures that the fixes have been implemented correctly and that the previously failed test cases now pass. |
| 5 | **frequency** | performed after every significant change or release, including bug fixes, enhancements, and new feature implementations | Retesting is conducted immediately after the fixes have been made to verify their effectiveness and ensure that the reported issues no longer exist. |

**Positive vs Negative testing:**

validate different aspects of a system's behaviour

|  |  |  |  |
| --- | --- | --- | --- |
| **Sr no** | **parameters** | **Positive testing** | **Negative testing** |
| 1 | **Objective** | validate the expected behaviors and functionality of the system under normal operating conditions. | verify that the system correctly detects, handles, and recovers from error conditions or invalid inputs. |
| 2 | **Test scenarios** | involves designing test scenarios and test cases that cover typical, valid usage patterns.  It aims to verify that the system handles standard inputs, operations, and flows correctly. | involves designing test scenarios and test cases that deliberately test the system's response to invalid or unexpected inputs, error conditions, and exceptional situations. |
| 3 | **Test data** | uses valid, appropriate test data that conforms to the expected input formats and ranges defined by the system's requirements. | Uses invalid, incorrect, or unexpected test data that falls outside the normal, valid input ranges defined by the system's requirements. |
| 4 | **Expected result** | expects the system to produce the correct outputs or results according to the expected behaviors defined in the requirements or specifications. | expects the system to handle error conditions appropriately, such as displaying meaningful error messages, rejecting invalid inputs, or gracefully recovering from errors. |
| 5 | **Pass criterion** | if the system behaves as expected and produces the correct outputs | if the system handles error conditions correctly and responds appropriately |
| 6 | **Fail criterion** | if the system deviates from the expected behavior or produces incorrect results. | if the system does not handle errors as expected or exhibits unexpected behavior in the face of invalid inputs or exceptional situations. |

**Globalisation vs localisation testing:**

Two distinct types of testing that focus on ensuring software applications are suitable for different target markets and cultural contexts.

|  |  |  |
| --- | --- | --- |
| **Sr no** | **Globalization testing** | **Localization testing** |
| 1 | objective of globalization testing is to validate the application's ability to function correctly across various cultures and locales, without any language, date, time, currency, or cultural barriers. | objective of localization testing is to validate that the application functions correctly in the target locale, including language, cultural norms, and regional requirements. |
| 2 | focuses on ensuring that the application is designed and developed with localization in mind, making it easier to adapt to specific target markets. | focuses on verifying the accuracy of translations, ensuring that text, labels, and messages are correctly localized and linguistically appropriate for the target market. |
| 3 | includes testing for internationalization features such as encoding support, character sets, locale-specific formatting, multilingual support, and proper handling of date, time, and currency formats. | involves checking that the application is culturally adapted, taking into account factors such as date formats, time formats, number formats, currency symbols, and regional preferences. |
| 4 | involves testing the user interface to ensure that it can accommodate and display different languages, scripts, and character sets correctly. | verifies that the localized application behaves as expected in terms of functionality, including validation of localized user interface elements, localized data formats, and proper functioning of localized features. |
| 5 | checks that the application complies with localization best practices, such as separating code from text, supporting language resource files, and implementing proper string handling and formatting mechanisms. | includes evaluating the usability and user experience of the application in the target locale, ensuring that it meets the expectations and preferences of the local users. |
| 6 | also known as internationalization testing or I18N testing | also known as L10N testing |

**End to end testing:**

* End-to-end testing is a comprehensive testing approach that verifies the entire software system's behaviour from start to finish, mimicking real-world user scenarios.
* It aims to validate the interactions and integration between various components, subsystems, and dependencies within the system.
* E.g login---add new customer—edit customer—delete customer—logout

**Adhoc testing vs monkey testing vs exploratory testing:**

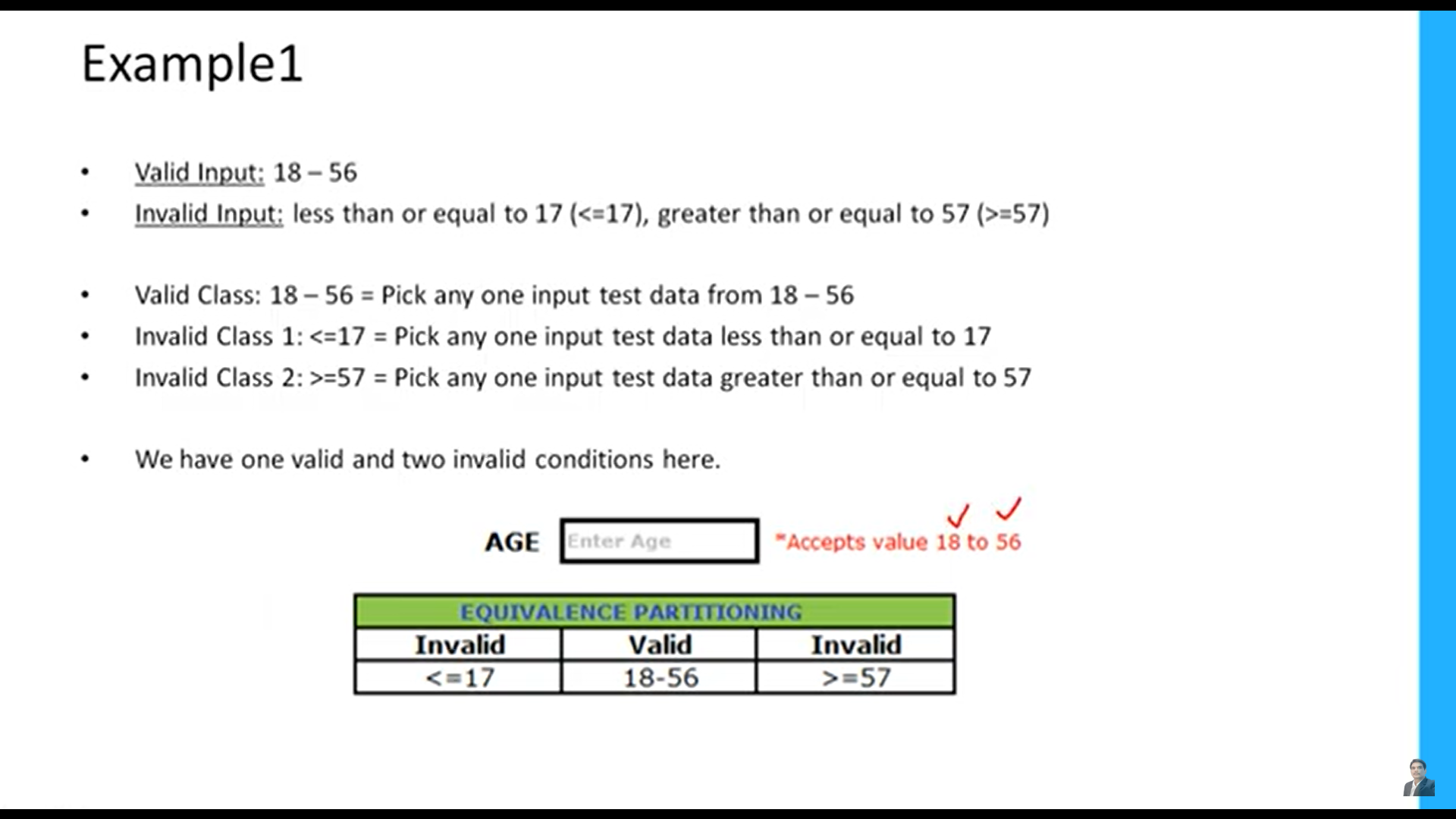
|  |  |  |  |
| --- | --- | --- | --- |
| **Sr no** | **Adhoc testing** | **Monkey testing** | **Exploratory testing** |
| 1 | No documentation | No documentation | No documentation |
| 2 | No plan | No plan | No plan |
| 3 | Informal testing | Informal testing | Informal testing |
| 4 | Tester should know application functionality | Tester doesn’t know application functionality | Tester doesn’t know application functionality |
| 5 | Random testing | Random testing | Random testing |
| 6 | Intension is to break the application/ find out corner defects | Intension is to break the application/ find out corner defects | Intension is to learn or explore functionality of application |
| 7 | Any applications | Gaming applications | Any application which is new to tester |

**Test design techniques:**

* Test design techniques, also known as **test design methodologies or test design approaches**, are systematic approaches used to determine how test cases should be created and organized.
* These techniques help ensure that the testing process is thorough, efficient, and covers the relevant aspects of the system under test.
* Here are some commonly used test design techniques:

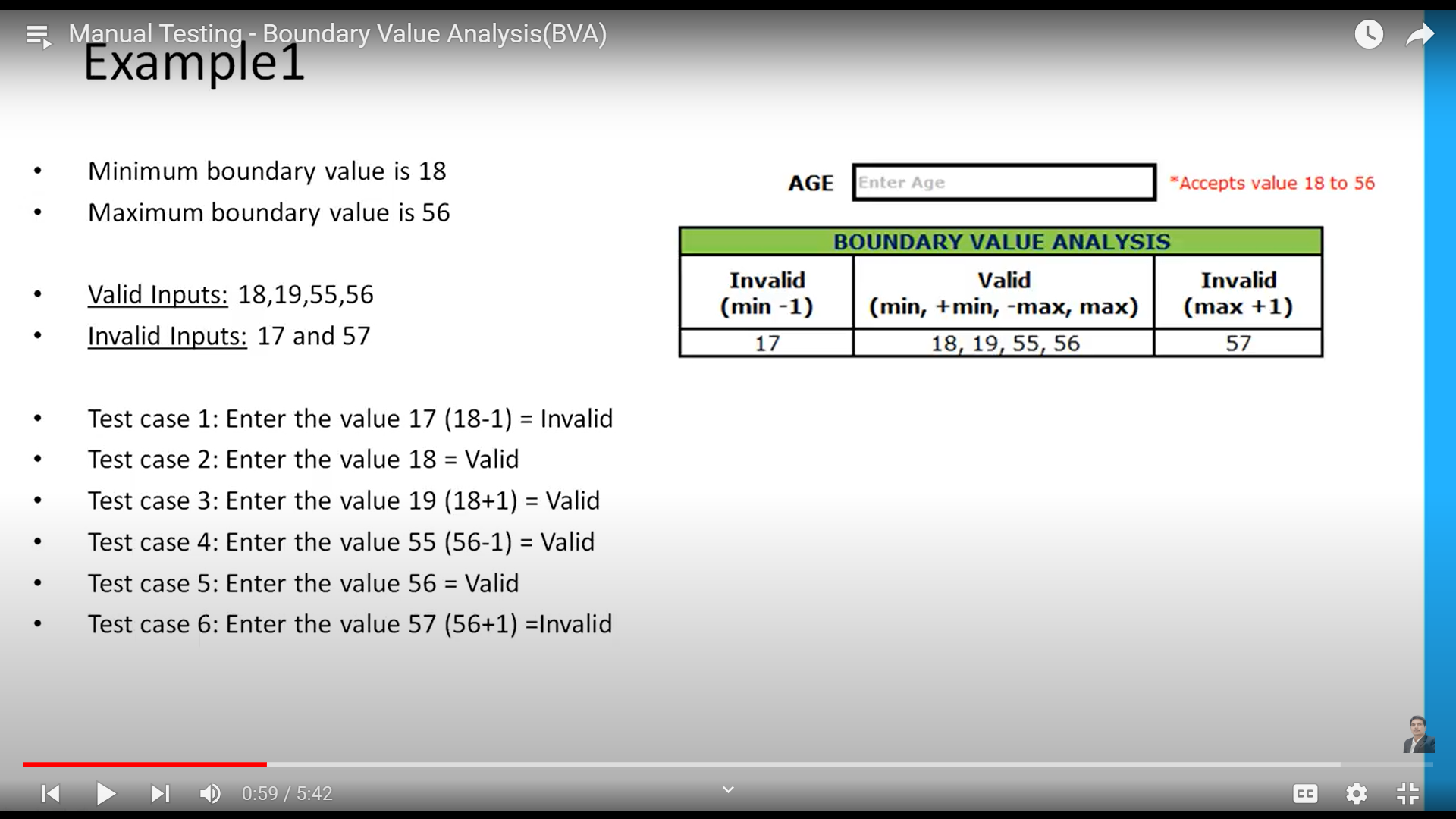
1. **Equivalence partitioning:**

* This technique divides the input domain into groups or partitions and selects representative values from each partition as test cases.
* The idea is that if one value in a partition behaves correctly, other values in the same partition should behave similarly.



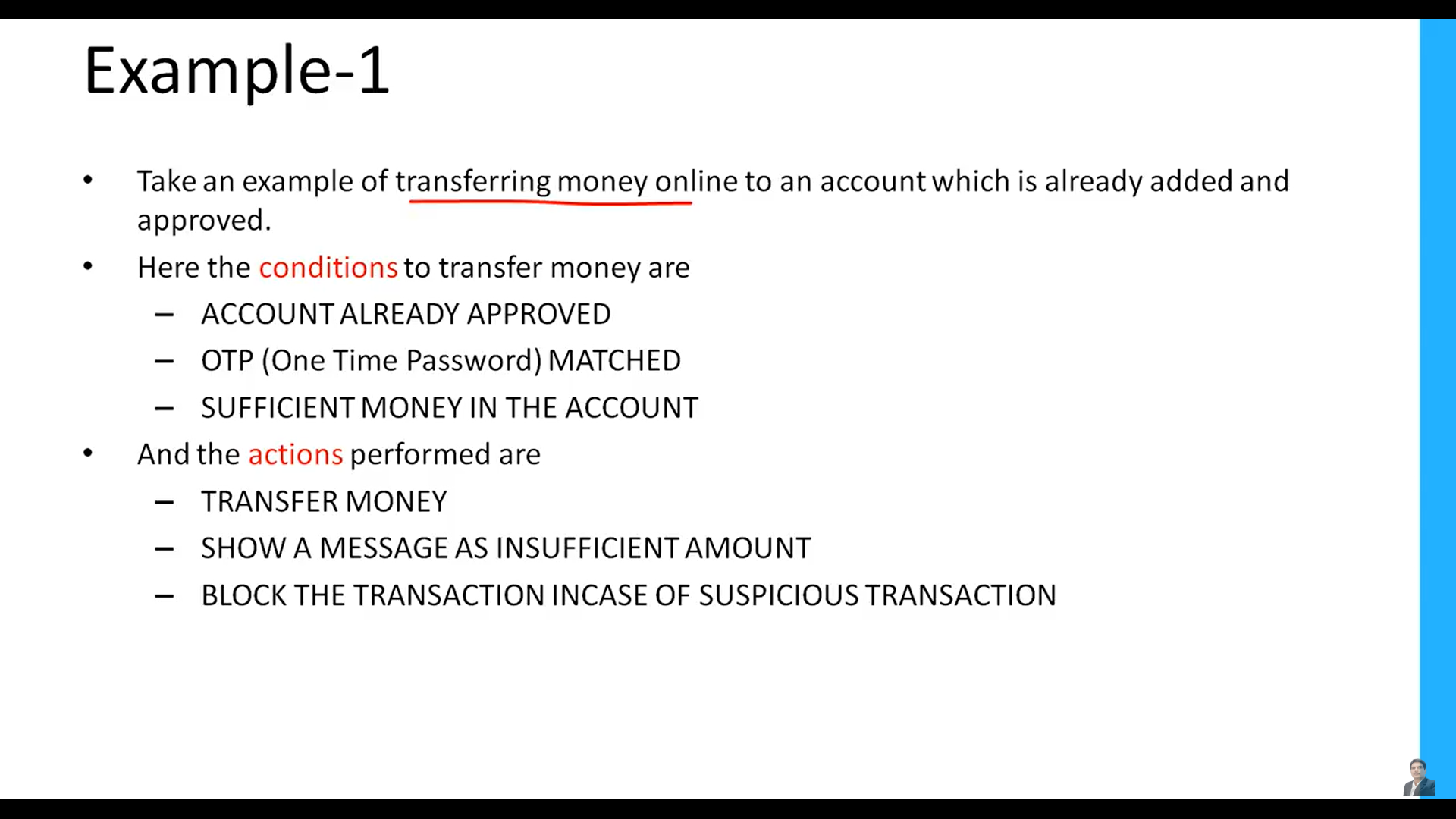
1. **Boundary value analysis:**

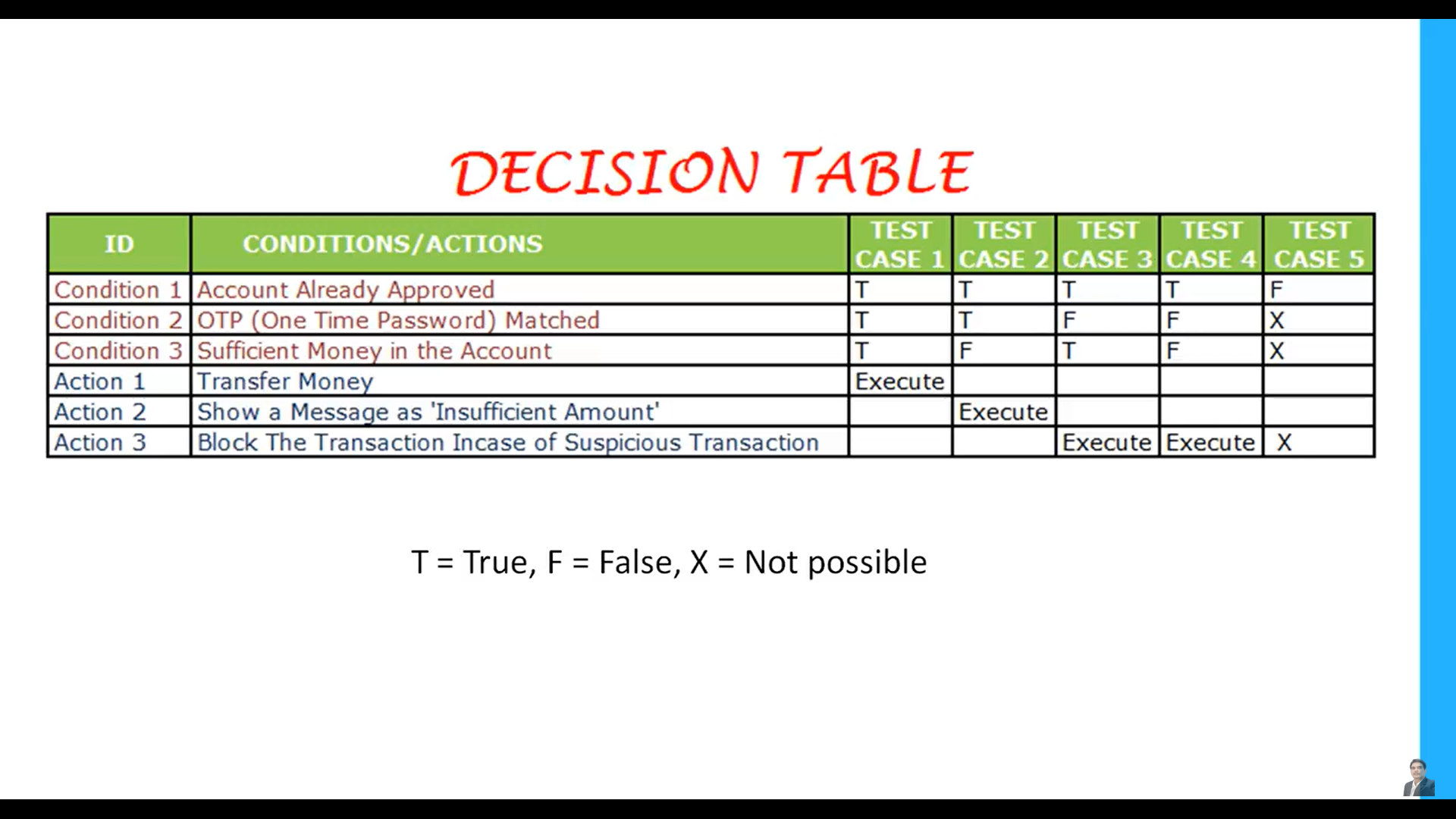
* focuses on selecting test cases at the boundaries or limits of input domains.
* It aims to identify defects that are often found near the boundaries rather than in the middle of the input range.



1. **Decision table testing:**

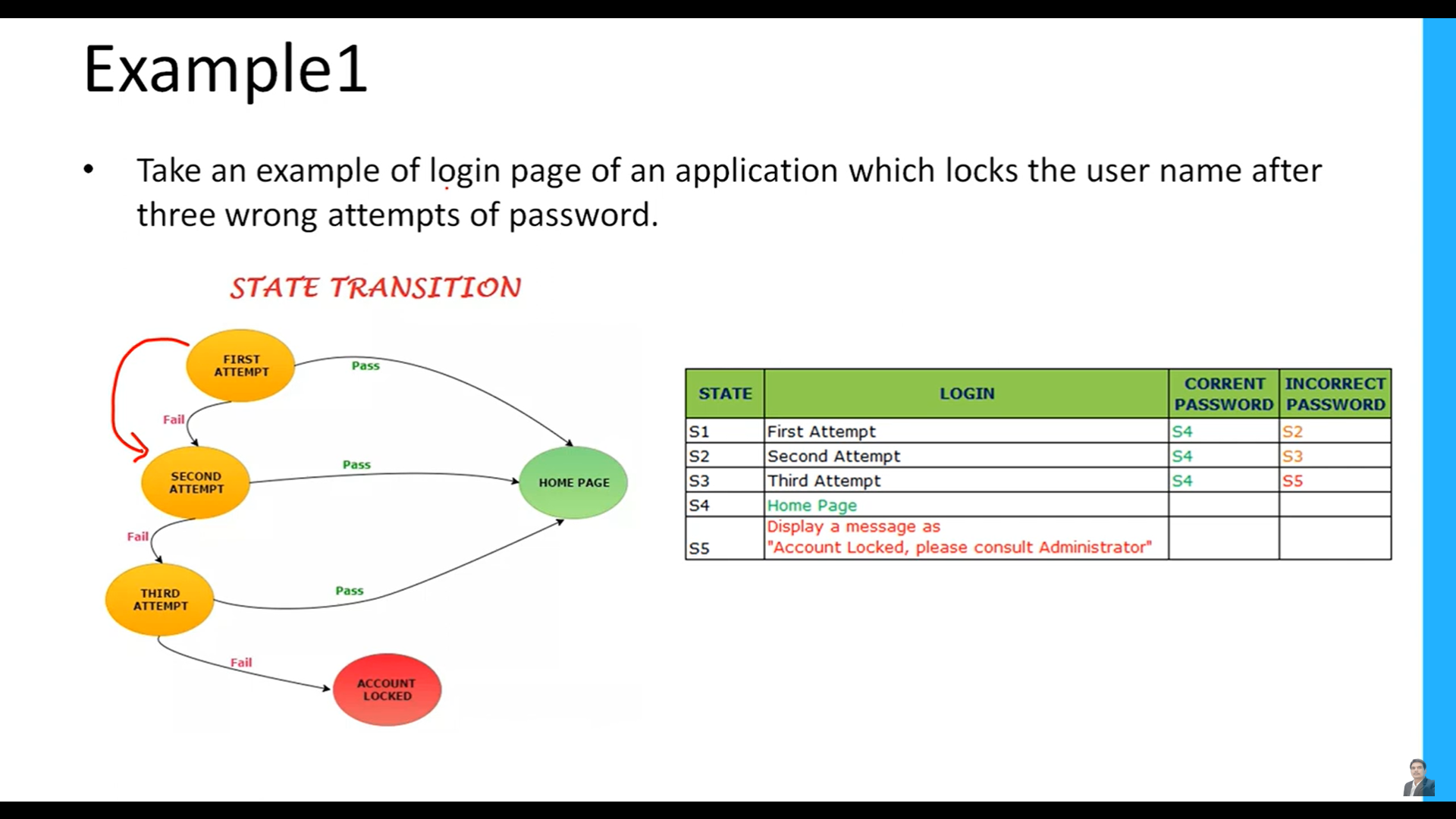
* technique used for testing systems with complex business rules or logic.
* It involves creating a table that maps different combinations of inputs and corresponding expected outputs, allowing testers to systematically identify and test various scenarios.





1. **State transition testing:**

* is particularly useful for testing systems with distinct states and transitions.
* Test cases are designed to cover different state changes and ensure that the system behaves correctly during transitions.



1. **Pairwise testing:**

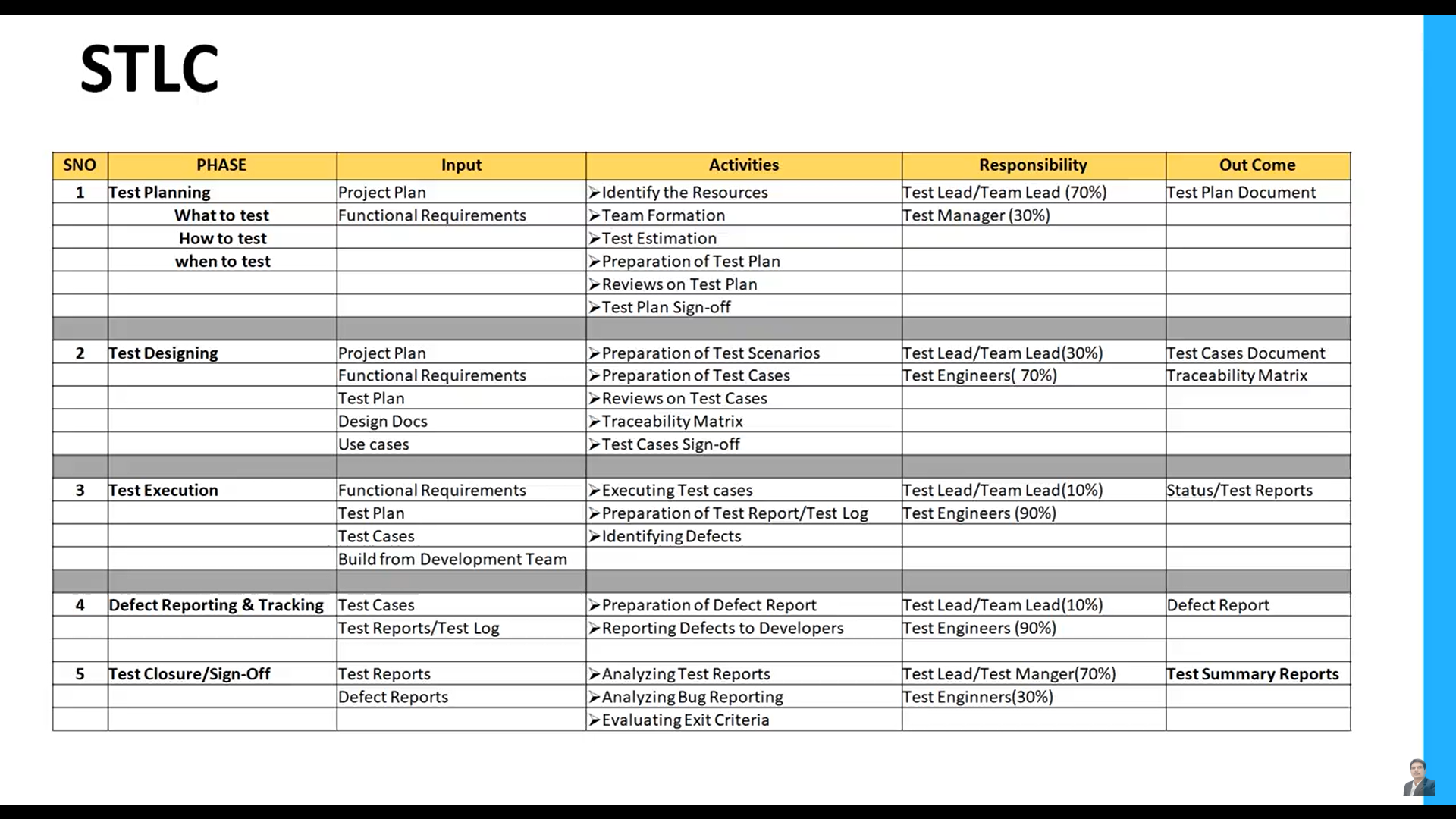
* also known as **all-pairs testing,**
* is an approach that selects a minimum set of test cases to cover all possible combinations of input parameters.
* This technique aims to achieve a high level of test coverage with a reduced number of test cases.

1. **Risk-based Testing:**

* involves prioritizing test cases based on the perceived risks associated with different system features or functionalities.
* It ensures that testing efforts are focused on areas that are more likely to have defects or where defects would have a significant impact.

1. **Domain Testing:**

* focuses on selecting test cases based on the understanding of the system's domain or industry.
* It considers the specific knowledge and rules associated with the domain to ensure that the system handles domain-specific scenarios correctly.



**Test plan document:**

* A test plan document is a formal document that outlines the approach, scope, objectives, and activities planned for testing a software application or system.
* It serves as a comprehensive guide for the testing team, stakeholders, and other project members, providing a roadmap for how testing will be conducted.
* Here are the key components typically included in a test plan document:

1. **Introduction:**

* This section provides an overview of the test plan, including its purpose, objectives, and scope.
* It describes the software system being tested, identifies the stakeholders, and outlines any assumptions and constraints.

1. **Test Objectives:**

* The test objectives section defines the specific goals and objectives of the testing effort.
* It may include goals such as validating functionality, verifying system performance, ensuring compatibility, or achieving specific quality criteria.

1. **Test Approach:**

* This section describes the overall approach and strategies that will be used to conduct the testing.
* It outlines the test levels (such as unit testing, integration testing, system testing, etc.) and the corresponding techniques or methodologies to be employed.

1. **Test Deliverables:**

* The test plan specifies the deliverables that will be produced during the testing process.
* This can include test cases, test scripts, test data, test reports, defect reports, and any other relevant documentation.

1. **Test Environment:**

* This section describes the required test environment, including hardware, software, network configurations, and any dependencies.
* It also addresses any specific setup or configuration needed for testing.

1. **Test Schedule:**

* The test plan includes a timeline or schedule that outlines the testing activities, milestones, and estimated effort for each phase of testing.
* It helps in coordinating testing efforts with other project activities and ensures proper resource allocation.

1. **Test Entry and Exit Criteria:**

* The entry and exit criteria define the conditions that must be met before testing can begin (entry criteria) and the conditions that indicate when testing can be considered complete (exit criteria).
* These criteria are typically based on factors such as test readiness, availability of test items, and defect thresholds

1. **Test Execution and Reporting:**

* This section provides details on how the tests will be executed, including the test procedures, test data, and any specific test environments or configurations.
* It also outlines the reporting mechanisms for documenting test results, defects, and overall test status.

1. **Test Risks and Contingencies:**

* The test plan addresses the potential risks and challenges associated with the testing effort.
* It identifies the risks, their potential impact, and mitigation strategies or contingency plans to minimize their effects on the testing process.

1. **Test Team and Responsibilities:**

* This section outlines the roles and responsibilities of the individuals involved in the testing effort, including testers, test leads, stakeholders, and any other relevant team members.
* It clarifies the organizational structure, communication channels, and coordination mechanisms.

1. **Approvals:**

* The test plan document concludes with a section for obtaining approvals from relevant stakeholders, such as project managers, development leads, and quality assurance managers.

**Test case document: (how to test)**

* test case document is a detailed document that describes individual test scenarios or test cases that need to be executed to validate specific features, functions, or behaviours of a software application or system.
* It provides step-by-step instructions for executing the tests, along with the expected results and any necessary test data.
* Here are the key components typically included in a test case document:

1. **Test Case ID:**

* A unique identifier assigned to each test case for easy reference and tracking.

1. **Test Case Title:**

* A descriptive title that summarizes the objective or focus of the test case.

1. **Test Objective:**

* A clear statement of the specific objective or goal of the test case.

1. **Test Pre-conditions:**

* The necessary preconditions or setup required before executing the test case, such as specific data, system configuration, or prerequisite actions.

1. **Test Steps:**

* Step-by-step instructions for executing the test, including any specific inputs or actions to be performed.
* Each step should be clear, concise, and unambiguous.

1. **Test Data:**

* The specific test data required for the test case, such as input values, expected outputs, or reference data.

1. **Expected Results:**

* The expected outcome or behavior of the system when the test is executed correctly.
* It should be measurable, observable, and verifiable.

1. **Actual Results:**

* The actual outcome or behavior observed during test execution.
* Testers document the actual results to compare them against the expected results.

1. **Pass/Fail Criteria:**

* Criteria or conditions for determining whether the test case has passed or failed based on the comparison between the actual results and the expected results.

1. **Test Environment:**

* The specific test environment or configuration required for executing the test case, including hardware, software, operating systems, browsers, or network conditions.

1. **Test Dependencies:**

* Any dependencies or relationships with other test cases or prerequisites that need to be considered.

1. **Test Execution Notes:**

* Additional notes, comments, or observations related to the test execution process or any issues encountered.

1. **Test management tool:**

* Load test cases finally in to test management applications include tools like Jira, TestRail, qTest, Zephyr, and HP ALM (Application Lifecycle Management)

**Test scenario: (what to test)**

* A test scenario is a broader concept that defines a high-level test condition or situation that needs to be tested.
* It describes the end-to-end flow or interaction between different components or functionalities of a system.
* Test scenarios are typically derived from user requirements, use cases, or business processes.
* They capture real-life usage scenarios and help ensure that the software application meets the desired objectives and functionality.
* Test scenarios provide a context for creating more specific test cases that cover different aspects and variations of the scenario.

**Use case:**

* A use case is a concept from software development and requirements analysis that describes a specific interaction or behaviour of a system from the perspective of an actor (user or system).
* It outlines a sequence of steps or actions that represent a typical flow of events to achieve a particular goal or result.
* Use cases are used to capture functional requirements and define how users or actors interact with the system to accomplish specific tasks.
* They focus on the system's behaviour and the interactions between users and the system.

**RTM:**

* RTM stands for **Requirements Traceability Matrix**.
* It is a document that provides a mapping or traceability between the requirements and the test cases.
* The RTM ensures that each requirement specified for a software system has associated test cases to validate its implementation.
* The primary purpose of an RTM is to establish a clear link between the requirements and the corresponding test cases.
* It helps in ensuring that all requirements are thoroughly tested and that there is complete test coverage for the system.
* The RTM also facilitates requirements change management, as any changes to requirements can be traced to the impacted test cases, enabling effective impact analysis.

An RTM typically includes the following information:

1. Requirement ID/Number: A unique identifier for each requirement.
2. Requirement Description: A description or statement of each requirement.
3. Test Case ID/Number: A unique identifier for each test case.
4. Test Case Description: A description of each test case.
5. Test Result: The outcome or status of the test case (e.g., Pass, Fail, Not Executed).
6. Remarks/Comments: Any additional notes or comments related to the requirement or test case.

**Defect metrics:**

* Defect metrics, also known as **defect metrics or defect management metrics**, are measurements used to evaluate and track the performance and effectiveness of defect management activities in a software development project.
* These metrics provide quantitative information about the quality of the software and the efficiency of the defect management process.
* They help in identifying trends, patterns, and areas for improvement in the software development and testing processes.
* **Defect rejection ratio:**

DRR=(No of defects rejected/ no of defects raised) \*100

No of defects rejected= it means defects we reported are wrong , it is mistake in our testing

* **Defect leakage ratio:**

DLR= (number of defect missed / total defects in software) \* 100

No of defect missed= testing team not able to detect defect from actual defect available

**Severity:**

* Severity refers to the degree of impact or seriousness of a defect or issue on the functionality or usability of the software.
* Severity levels are typically categorized as follows:

1. **Critical/Blocker:**

* Defects that completely prevent the system from functioning or cause a critical failure.
* These defects must be resolved immediately, as they significantly impact the core functionality and may lead to system crashes or data loss.

1. **Major/High:**

* Defects that have a major impact on the functionality or usability of the system but do not result in a complete failure.
* These defects may cause significant disruption or inconvenience to users and require prompt attention.

1. **Medium:**

* Defects that have a moderate impact on the system's functionality or usability.
* They may not cause critical issues but still need to be addressed to ensure a satisfactory user experience.

1. **Minor/Low:**

* Defects that have a minor impact on the system's functionality or usability.
* These defects do not significantly affect the core features but may cause minor inconveniences or cosmetic issues.

**Priority:**

* indicates the order in which defects or issues should be addressed based on their importance, business value, and urgency.
* It helps in managing the sequence of defect resolution activities. Priority levels can be assigned as follows:

1. **Immediate/Urgent:**

Defects or issues that require immediate attention and resolution, typically due to their critical impact on the business, customer experience, or regulatory compliance.

1. **High: (P1)**

* Defects or issues that should be resolved with high priority but may not be as time-sensitive as immediate issues.
* They have a significant impact on the system's functionality or business operations.

1. **Medium: (P2)**

* Defects or issues that need attention but are not as critical as immediate or high priority ones.
* They may have a moderate impact on the system and can be resolved within a reasonable timeframe.

1. **Low: (P3)**

* Defects or issues that have a lower impact on the system's functionality or business operations.
* They can be resolved in later stages or when higher priority issues are addressed.

**Conditions:**

* Critical or high severity defects often receive a higher priority because they have a significant impact on the system's functionality or usability.
* defect with lower severity but high business impact or customer visibility is given a higher priority.

**Bug life cycle:**

* also known as the **defect life cycle**, describes the different stages that a bug or defect goes through from its identification to its resolution.
* The bug life cycle helps in tracking and managing defects effectively.
* Although the specifics can vary depending on the organization and project, here is a generalized representation of the bug life cycle:

1. **New/Open:**

* A bug is identified and reported by a tester or any stakeholder.
* At this stage, the bug is logged into the bug tracking system with all the relevant details such as its description, severity, priority, and any supporting documentation.

1. **Assigned:**

* The bug is assigned to a developer or a development team responsible for fixing the bug.
* The assigned developer reviews the bug report and begins analyzing it to understand the root cause.

1. **In Progress:**

* The developer starts working on fixing the bug.
* They debug the code, identify the cause of the defect, and make the necessary code changes or fixes to address the bug.

1. **Fixed:**

* Once the developer believes that they have resolved the bug, they mark it as "Fixed" or "Resolved" in the bug tracking system.
* The bug fix is usually accompanied by a comment or explanation of the changes made.

1. **Ready for Retest:**

* After the bug is fixed, it is assigned back to the testing team for retesting.
* The tester verifies whether the bug has been successfully resolved and confirms if the fix has not introduced any new issues.

1. **Retest:**

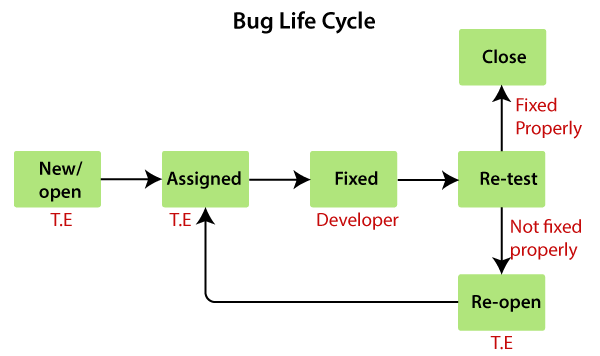
* The testing team executes the relevant test cases or test scenarios to validate the bug fix.
* They confirm if the issue has been resolved and no regressions or new defects have been introduced.

1. **Closed/Verified:**

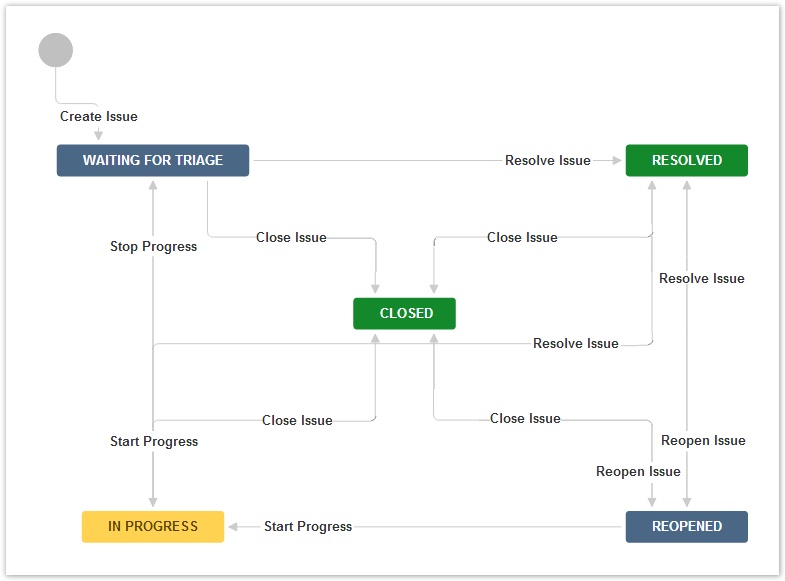
* If the bug fix passes the retest and meets the defined criteria for closure, it is marked as "Closed" or "Verified" in the bug tracking system.
* The bug is considered resolved, and no further action is required

1. **Reopened:**

* In case the bug is found to be not fixed or if the issue resurfaces during retesting, it is marked as "Reopened" in the bug tracking system.
* It is then reassigned to the developer to readdress the defect.



**jira bug life cycle:**



**Defect triage:**

* Defect triage is a process that tries to do the re-balancing of the process where test team faces the problem of limited availability of resources
* When there are large number of the defects and limited testers to verify them, defect triage helps trying to get as many defects resolved based on defect parameters like severity and priority.

**Process :**

1. Defect review
2. Defect assessment
3. Defect assignment

**Defect resolution:**

* After receiving the defect report from the testing team, development team conduct a review meeting to fix defects.
* Then they send resolution type to the testing team for further communication.

**Resolution type:**

1. Accept
2. Reject
3. Duplicate
4. Enhancement
5. Need more information
6. Not reproducible
7. Fixed
8. As designed

**Defect report:**

* A defect report, also known as a **bug report or an issue report**, is a formal document that describes a defect or bug identified in software during the testing or development process.
* It serves as a means of communication between testers, developers, and other stakeholders involved in the defect resolution process.
* The defect report typically includes the following:

1. **Defect Identifier:**

* A unique identifier or number assigned to the defect for easy reference and tracking.

1. **Defect Title/Summary:**

* A concise and descriptive title or summary that captures the essence of the defect.

1. **Description:**

* A detailed description of the defect, including the steps to reproduce it, the observed behavior, and any relevant supporting information.

1. **Environment:**

* Information about the environment or configuration in which the defect was discovered, such as the operating system, browser, hardware, or network setup.

1. **Severity:**

* The severity level or impact of the defect on the software's functionality or usability.
* It helps in assessing the criticality of the defect.

1. **Priority:**

* The priority assigned to the defect, indicating its relative importance or urgency for resolution based on business needs, customer impact, and project timelines.

1. **Attachments:**

* Any supporting documents, screenshots, logs, or other artifacts that help in understanding or reproducing the defect.

1. **Reported By:**

* The name or identifier of the person who reported the defect, along with the date of reporting.

1. **Assigned To:**

* The person or team responsible for addressing the defect, typically the developer or development team.

1. **Status:**

* The current status of the defect, indicating whether it is open, in progress, fixed, retested, or closed.

1. **Comments/History:**

* A log of comments, updates, or discussions related to the defect, including any interactions between testers, developers, and other stakeholders.

1. **Test Case/Scenario:**

* The test case or test scenario that initially uncovered the defect, if applicable.

Defect reports provide crucial information about the identified defects, allowing developers to understand, reproduce, and resolve the issues effectively