Generated By Syed Tabish Ishtiaq
Email:tabishishtiaq5@gmail.com
+923343308650

What is Software?

Software is a set of instructions, data or program used to operate computer.

What are types of Software?

System software helps the user, hardware and application software to communicate with each other. Examples Operating system, Bios, Device -drivers etc.

- Programming Software → IDE
- Driver software → Hardware Driver software

Application software is a pre-ready software to use, helps end users to collect notes, data, research Example: Microsoft word, Calculators, Browser.

What is Web Applications?

Web applications are an application programs that is stored on a remote- servers and delivered over the internet through a browser interface. Example online forms, shopping carts, video streaming, social media, games etc.

What are the types of web application?

Types of web applications

- Static web application. ...
- Dynamic web application. ...
- E-Commerce web application. ...
- Single-page web application. ...
- Portal web application. ...
- Content management system web application. ...
- Animated web applications. ...
- Rich Internet web applications.

What is SDLC?

SDLC is a defined process which is used to develop a software.

What are the stages of SDLC?

- Requirement Analysis
- Design
- Implementation
- Verification or Testing
- Maintenance

Generated By Syed Tabish Ishtiaq
Email:tabishishtiaq5@gmail.com
+923343308650

What happens in SDLC Phases?

1. Requirements Gathering Phase:
   - Entry Criteria: Business case approved, project charter in place, and stakeholders identified.
   - Activities: Gather user requirements, conduct interviews and workshops, document functional and non-functional requirements.
   - Exit Criteria: Approved requirements document, signed-off user requirements.
   - Deliverables: Requirements document, use cases, user stories.
2. System Analysis Phase:
   - Entry Criteria: Completed requirements gathering phase, approved requirements document.
   - Activities: Analyze requirements, define system architecture, identify system components and interfaces.
   - Exit Criteria: System architecture document, system design specifications.
   - Deliverables: System architecture document, system design specifications.
3. System Design Phase:
   - Entry Criteria: Completed system analysis phase, approved system design specifications.
   - Activities: Develop detailed system design, create database schema, design user interface, define system modules and functions.
   - Exit Criteria: Detailed system design document, approved database schema and user interface designs.
   - Deliverables: Detailed system design document, database schema, user interface designs.
4. Coding/Development Phase:
   - Entry Criteria: Completed system design phase, approved detailed system design.
   - Activities: Write code, develop software components, perform unit testing, integrate modules.
   - Exit Criteria: Developed and tested software modules, integrated system.
   - Deliverables: Code files, software modules, integrated system.
5. Testing Phase:
   - Entry Criteria: Completed coding/development phase, integrated system.
   - Activities: Develop test cases, perform functional and non-functional testing, execute test cases, log and track defects.
   - Exit Criteria: Completed test cases, resolved defects, acceptable test coverage.
   - Deliverables: Test cases, test results, defect reports.
6. Deployment/Release Phase:
   - Entry Criteria: Completed testing phase, approved software.
   - Activities: Prepare for deployment, create installation packages, perform user acceptance testing, deploy software.
   - Exit Criteria: Successfully deployed software, acceptance from users.
   - Deliverables: Installed software, user acceptance sign-off.
7. Maintenance and Support Phase:
   - Entry Criteria: Completed deployment/release phase.
   - Activities: Provide ongoing maintenance and support, address user feedback and issues, perform bug fixes and updates.

- Exit Criteria: Sustained system performance, satisfied user requirements.
- Deliverables: Bug fixes, system updates, user support.

What are the SDLC Models?

1. Waterfall is a sequential and linear approach, where each phase of SDLC is completed before moving to next phase, It is suitable for projects which have stable and well defined requirements.
2. Iterative model is a process which involves repetitive cycles of development each cycle includes requirements gathering, design, implementation and testing. After each iteration a working product increment delivered.
3. Spiral Model is a combined element of both the waterfall and iterative model. It follows a spiral pattern approach for development means where each iteration consist of following phases
    - Determine Objectives → Project objectives and requirements are defined.
    - Identify and Resolve risk → Focused an identifying potential risk and finding ways to mitigate them.
    - Development and Testing → software developed based on the defines objectives and requirements includes coding, integration and testing.
4. The V-Model is an extension of the waterfall model that follows an approach of aligning the testing phase with its corresponding development phase.

5. Rapid Application Development Model(RAD) is a adaptive software development model based on prototyping and quick feed back with specific planning.

6. DevOps Model is a model where development and operations teams work together across the entire software/application development lifecycle, from development and testing through deployment to operation.

What is Agile?

Agile is a project management approach that follows a set of principles known as Agile methodologies for development of software.

What are Agile Principles?

- Customer Satisfaction
- Welcome Change
- Deliver Frequently
- Work Together
- Motivated Team
- Face-to-face
- Working Software

- Constant Pace
- Good Design
- Simplicity
- Self-Organization
- Reflect and Adjust

What is Scrum?
Scrum is an agile framework for managing and organizing complex projects particularly in software development follows agile principle.

What are Scrum Roles?
Product Owner → The Product Owner presents to the stakeholders and is responsible for managing and prioritizing complex projects, particularly in software development. They follow Agile principles.
Scrum Master → Facilitates the Scrum process, removes any obstacles for team performance, and ensures adherence to Scrum principles.
Development Team → is a self-organized and responsible for delivering the product increments.

What are the Scrum Outputs/artifacts?

1. Product Backlog: The Product Backlog is a prioritized list of user stories, features, and requirements that need to be implemented in the product. It is managed and maintained by the Product Owner.
2. Sprint Backlog: The Sprint Backlog is a subset of the Product Backlog and contains the list of tasks and user stories that the Development Team commits to complete within a Sprint.
3. Increment: The Increment is the sum of all the completed and potentially shippable product backlog items at the end of a Sprint. It represents a usable and potentially releasable version of the product.
4. Sprint Goal: The Sprint Goal is a short statement that describes the objective or purpose of the Sprint. It provides a clear focus and direction for the Development Team during the Sprint.

How many scrum meeting's/ events occured?

1. Sprint Planning: This meeting marks the beginning of the sprint and involves the Product Owner and Development Team. They collaborate to determine which backlog items will be worked on during the sprint and create a sprint goal.
2. Daily Scrum: Also known as the daily stand-up, this is a short daily meeting for the Development Team to synchronize their activities. Each team member answers three questions: What did I do yesterday? What will I do today? Are there any obstacles in my way?
3. Sprint Review: At the end of the sprint, the Development Team presents the completed work to the stakeholders and gathers feedback. The Product Owner assesses the work completed against the sprint goal.
4. Sprint Retrospective: This meeting occurs after the Sprint Review and is an opportunity for the Scrum Team to reflect on the sprint and identify areas for improvement. They discuss what went well, what could be improved, and define action items for the next sprint.

Generated By Syed Tabish Ishtiaq
Email:tabishishtiaq5@gmail.com
+923343308650

What is Sprint?

Sprint is a process of s a time-boxed iteration during which the Scrum Team works to complete a set of prioritized sprint backlog items.

How many scrum charts?

1. Burndown Chart: Tracks remaining work over time in a sprint or project, helping the team monitor progress and stay on track.
2. Velocity Chart: Shows average work completed by the team in each sprint, aiding in future work estimation and project timeline planning.
3. Cumulative Flow Diagram: Visualizes work flow through different stages, identifies bottlenecks, and evaluates overall workflow efficiency.
4. Release Burndown Chart: Tracks progress of completing backlog items or features over time, ensuring alignment with planned scope and release date.
5. Sprint Goal Chart: Illustrates the sprint goal and key objectives, keeping the team focused and aligned throughout the iteration.
6. Team Task Board: Visualizes tasks or user stories within a sprint, promoting transparency and collaboration among team members.

What meant by User story and user point in scrum?

1. User Stories: User stories are brief descriptions of a feature or functionality written in a user-centric format. They capture the user's role, the desired feature, and the benefit it provides. User stories promote collaboration and understanding between the development team and stakeholders.
2. Story Points: Story points are a relative measure used to estimate the effort required to complete a user story. They represent the complexity and size of the work. Story points are assigned based on the team's collective judgment, using a scale like the Fibonacci sequence. They help in planning and prioritizing tasks, and provide a basis for measuring the team's productivity and velocity.

What is manual testing?
Manual Testing is a process of software testing in which we execute the test cases manually without using any automated testing too.

What are approaches of manual testing?

1. White-box Testing →White-box testing examines the internal structure, design, and code of a software application. Testers have access to the system's internal details and create test cases based on this knowledge. The goal is to validate the internal logic and flow of the software.
2. Black-box Testing → Black-box testing focuses on the external behavior of a software application. Testers have no knowledge of the internal implementation and treat the software as a "black box." Test cases are created based on specified requirements to validate if the software functions correctly without considering internal details.

3. Grey-box Testing → Grey-box testing combines elements of white-box and black-box testing. Testers have partial knowledge of the internal structure and code of the software. They leverage this knowledge to design more effective test cases. Grey-box testing strikes a balance between internal understanding and requirement-based testing.

What are Software testing Levels?

Unit Testing → Unit testing focuses on testing small, individual parts of the software to ensure they work correctly. It helps identify bugs or errors early on and improves the overall quality of the software.
Integration Testing → Integration testing verifies how different parts of the software interact with each other. It ensures that integrated components function as intended and transfer data accurately. Integration testing helps uncover and resolve issues that may arise from the integration of components.
System Testing → System testing evaluates the entire software system to ensure it functions properly. It checks if all components and subsystems work together seamlessly, meet the specified requirements, and deliver the intended functionality. System testing covers various aspects, including performance and reliability.
Acceptance Testing → Acceptance testing assesses whether the software meets the business requirements and fulfills user expectations. It involves testing real-world scenarios and user inputs to validate the software's functionality. Acceptance testing is typically conducted by end-users or stakeholders to determine if the software is ready for production use.

What are the software test types?

Functional Testing →Functional testing is the process of checking if software works as intended and meets the specified requirements.
Non-Functional Testing →Non-functional testing evaluates how well software performs under different conditions, focusing on aspects like speed, usability, security, reliability, and compatibility.
Structural Testing → Structural testing, also known as white-box testing, examines the internal structure and code of software to ensure its logical flow and relationships between components are correct.
Change Related Testing →Change-related testing is performed when software is modified or enhanced, aiming to test the affected areas and ensure that the changes do not introduce new issues or disrupt existing functionality.

What are the types of Functional Testing?

Unit Testing → Unit testing is when individual parts or components of software are tested to ensure they work correctly in isolation.

Generated By Syed Tabish Ishtiaq
Email:tabishishtiaq5@gmail.com
+923343308650

Exploratory Testing → Exploratory testing is an approach where testers explore the software application without predefined test cases. They learn, test, and adapt their testing as they interact with the system, uncovering defects and gaining a deeper understanding of the software.

Ad-hoc Testing → Ad-hoc testing is an informal and unplanned testing approach. Testers perform tests based on their experience, intuition, and domain knowledge, without following any specific test documentation or predefined steps.

Component Testing → Component testing focuses on testing the different parts or modules of software to make sure they function properly and fit together seamlessly.
Smoke Testing → Smoke testing is a quick initial test that checks if the essential features of an application work without going into extensive testing.
Sanity Testing → Sanity testing is a brief and focused test performed to quickly verify if recent changes or fixes in the software have not caused major issues or bugs.
Regression Testing → Regression testing involves retesting specific areas of software to confirm that previous functionality has not been impacted by changes or updates.
Integration Testing → Integration testing verifies how well different components or modules of software work together and interact with each other.
API Testing → API testing is the evaluation of the application programming interfaces (APIs) to ensure they function correctly, are reliable, secure, and meet the specified requirements.
UI Testing → UI testing examines the user interface of software to ensure it is user-friendly, visually appealing, and operates correctly.
System Testing → System testing assesses the entire software system to ensure that all components function together as intended and meet the specified requirements.
White-box Testing → White-box testing is a method that looks into the internal structure and code of software to validate its logic and functionality.
Black-box Testing → Black-box testing is a technique that assesses the functionality of software without considering its internal code or structure. It focuses on inputs and outputs.

Acceptance Testing → Acceptance testing is performed to determine if the software meets the user's requirements and is ready for acceptance or deployment.

Alpha Testing → Alpha testing involves selected users or testers at the development site checking for defects and providing feedback before releasing the software to the public.
Beta Testing → Beta testing involves releasing the software to a limited group of external users or the public to gather feedback, identify bugs, and evaluate overall performance.
Production Testing → Production testing is conducted on the live or production environment to ensure the software functions correctly and meets performance and reliability expectations.

What are the types of Non-Functional Testing?

1. Performance Tests →Performance tests evaluate software performance under specific conditions to ensure it meets requirements and runs efficiently.
2. Load Tests → Load tests simulate real-world usage to check if the software can handle expected user loads and identify performance issues.

3. Stress Tests → Stress tests push software to its limits to assess stability and behavior under extreme conditions, finding its breaking point.
4. Volume Tests → Volume tests evaluate software performance and scalability with large data or transactions, ensuring it handles growth without performance degradation.
5. Security Tests → Security tests assess software resistance to vulnerabilities and attacks, identifying weaknesses and protecting sensitive data.
6. Upgrade & Installation Tests: Upgrade and installation tests verify smooth transitions between software versions, ensuring upgrades, updates, and installations don't impact functionality.
7. Recovery Tests → Recovery tests evaluate software's ability to recover from failures, restoring data and functionality after unexpected events.

What are the types of Structural Testing?

1. Statement Coverage Testing → This testing ensures that every line of code is executed at least once during testing, covering all executable statements.
2. Branch Coverage Testing → This testing ensures that all possible branches or decision points in the code, including both true and false branches, are tested.
3. Path Coverage Testing → This testing aims to cover every possible path through the code, ensuring that all sequences of statements and branches are executed.
4. Condition Coverage Testing →This testing verifies that all conditions in the code, including both true and false evaluations, are tested to ensure all possible outcomes are evaluated.
5. Loop Coverage Testing → This testing focuses on testing loops in the code, including zero iterations, one iteration, multiple iterations, and boundary cases.
6. Data Flow Testing → This testing examines how data is processed and propagated within the code, testing variables, assignments, and data dependencies to identify any issues related to data flow.
7. Mutation Testing → This testing involves making intentional modifications or mutations in the code to assess the effectiveness of test cases in detecting and identifying these mutations.
8. Boundary Value Testing →This testing checks the input and output boundaries of code segments to ensure that the code behaves correctly at the edges of valid and invalid inputs.
9. Equivalence Partitioning → This testing involves dividing input data into equivalent groups to reduce the number of test cases while ensuring that each group is adequately covered.

What are the types of Change related Testing?

1. Regression Testing → Regression testing validates that the existing software functionality is unaffected by changes. It involves retesting previously tested features to ensure they still work correctly.

Generated By Syed Tabish Ishtiaq
Email:tabishishtiaq5@gmail.com
+923343308650

2. Integration Testing → Integration testing verifies that different software components or modules work together seamlessly after changes. It ensures proper interaction and functionality integration.
3. Retesting → Retesting focuses on retesting specific areas or functionalities that have been modified or fixed to confirm that the reported issues have been resolved and the affected functionality works as expected.
4. Smoke Testing → Smoke testing is a quick initial test conducted after changes to ensure that critical software functionalities are working correctly before proceeding with more detailed testing.
5. Impact Analysis Testing → Impact analysis testing assesses the potential impact of changes on different areas of the software. It identifies affected areas and guides the selection of appropriate test cases for testing.
6. Configuration Testing → Configuration testing ensures that the software functions correctly with different configurations, setups, and environments after changes. It tests compatibility and performance under various configurations.
7. Patch Testing → Patch testing validates the correct application of software patches or hotfixes. It ensures that the patches address reported issues without introducing new problems.
8. Compatibility Testing → Compatibility testing verifies that changes to the software have not affected its compatibility with different platforms, browsers, operating systems, or hardware configurations.
9. Impact on Performance Testing → Impact on performance testing evaluates the effect of changes on the software's performance. It identifies performance improvements or degradation resulting from the changes.
10. Data Migration Testing → Data migration testing is performed when changes involve transferring data from one system or format to another. It ensures accurate and reliable data transfer without loss or corruption.

what mean by software test design technique?

- Software test design techniques → are systematic methods for creating test cases that ensure comprehensive coverage of the software. They help testers identify what to test—inputs, conditions, and actions—to design well-organized test cases. These techniques maximize coverage while minimizing test cases. Examples include equivalence partitioning, boundary value analysis, and decision table testing. They improve efficiency, prioritize critical areas, and uncover defects, ensuring high-quality software.

What are types/ categories of software test design Techniques?

1. Static Design Techniques → Static design techniques evaluate software design without executing the code. They find design flaws, inconsistencies, and issues early in development. Examples include design reviews, inspections, and code inspections.
2. Static Analysis → Static analysis analyzes source code or software artifacts without executing the program. Tools scan the code for defects, vulnerabilities, violations, and issues. It improves code quality, identifies errors, and enhances reliability and security.
3. Dynamic Test Design Techniques → Dynamic test design techniques create test cases based on software behavior during execution. They focus on scenarios, inputs, and paths.

Examples include equivalence partitioning, boundary value analysis, state transition testing, and decision table testing. These techniques ensure thorough testing and detect defects during runtime.

What are static Test Design Techniques?

1. Informal Review → Informal review is a flexible and casual approach where a group provides feedback and identifies potential issues in software artifacts without following a strict process or predefined roles.
2. Walkthrough → Walk-through is a collaborative approach where the author guides participants through a software artifact to gather feedback, clarify concepts, and promote understanding among stakeholders.
3. Technical Review → Technical review is a structured evaluation conducted by experts to assess technical aspects of software artifacts, such as design or code, to identify defects, risks, or improvements.
4. Inspection → Inspection is a rigorous technique where experts systematically examine software artifacts to identify defects, inconsistencies, or violations of standards, aiming to improve quality and correctness.
5. Static analysis → is a technique that analyzes software artifacts, like source code or documentation, without running the program. It uses tools to identify defects, vulnerabilities, violations, or other issues. It improves code quality, finds errors, and strengthens software reliability and security.

What are types/categories of Dynamic test design technique?

Specification based techniques

1. Equal Partitioning: Equal partitioning is a test design technique that groups input data into partitions, reducing the number of test cases needed. Each partition is tested at least once, covering different input value ranges or categories.

Example of Equal Partitioning:-

Consider a system that requires a user to input their income for a tax calculation. Instead of testing every possible income value, we can apply equal partitioning to group the income values into partitions. Let's assume we divide the income range into three partitions: Low, Medium, and High.

- Low Income Partition: $0 to $30,000
- Medium Income Partition: $30,001 to $70,000
- High Income Partition: Above $70,000

Using equal partitioning, we can select test cases from each partition to ensure coverage across different income ranges. For example, we might choose test cases such as $10,000 from the Low Income partition, $40,000 from the Medium Income partition, and $80,000 from the High Income partition.

By testing at least one value from each partition, we can assess how the system handles different income levels and identify any issues related to specific income ranges. This approach helps optimize the number of test cases required while ensuring adequate coverage of the input space.

2. Boundary Value Analysis: Boundary value analysis tests the boundaries of input values by selecting test cases at or just beyond those boundaries. It helps identify defects that often occur at the edges of valid and invalid input ranges.

Example of BVA:-

Consider a system that requires the user to input a number of items for an online shopping cart. The system allows a minimum of 1 item and a maximum of 10 items in the cart. Applying boundary value analysis, we would select test cases at the boundaries and just beyond them:

1. Minimum Value (Boundary): Test the system with 1 item in the cart. This checks if the system correctly handles the minimum valid input.
2. Just Below Minimum (Invalid): Test the system with 0 items in the cart. This ensures that invalid inputs below the minimum value are rejected.
3. Maximum Value (Boundary): Test the system with 10 items in the cart. This checks if the system correctly handles the maximum valid input.
4. Just Above Maximum (Invalid): Test the system with 11 items in the cart. This ensures that invalid inputs above the maximum value are rejected.

By selecting test cases at the boundaries and just beyond them, boundary value analysis helps identify defects that often occur at the edges of valid and invalid input ranges. It focuses on critical areas where issues are more likely to arise, providing thorough test coverage while minimizing the number of test cases required.

3. Decision Table Testing: Decision table testing systematically tests various combinations of conditions and corresponding actions using a tabular format. Testers create test cases to cover all possible combinations, ensuring comprehensive coverage and identifying defects related to decision logic.
Example of Decision Table:-

Consider a system that calculates the shipping cost for an online shopping platform based on the weight and destination of the package. The decision table for this scenario might look as follows:

Condition 1: Weight	**Condition 2: Destination**	**Action: Shipping Cost Calculation**
Below 1kg	Local	$5.00
Below 1kg	International	$15.00
1kg and above	Local	$10.00
1kg and above	International	$25.00

Using decision table testing, we can create test cases that cover all possible combinations of conditions and corresponding actions:

Generated By Syed Tabish Ishtiaq
Email:tabishishtiaq5@gmail.com
+923343308650

1. Test case 1: Weight below 1kg, Destination: Local. The expected action is a shipping cost of $5.00.
2. Test case 2: Weight below 1kg, Destination: International. The expected action is a shipping cost of $15.00.
3. Test case 3: Weight 1kg and above, Destination: Local. The expected action is a shipping cost of $10.00.
4. Test case 4: Weight 1kg and above, Destination: International. The expected action is a shipping cost of $25.00.

By creating test cases that cover all possible combinations, decision table testing ensures comprehensive coverage of the system's decision logic. It helps identify defects or issues related to decision-making within the system, ensuring accurate and reliable results based on different combinations of conditions.

4. State Transition Testing: State transition testing focuses on testing the behavior of systems with different states and transitions. Test cases cover various state transitions, ensuring correct system functionality as it moves between states.

Example for State Transition Testing:-

Consider a system for a traffic signal that has three states: "Red," "Yellow," and "Green." State transition testing focuses on testing the behavior of the system as it moves between these states. Let's define the valid state transitions:

- From "Red," the signal transitions to "Green."
- From "Green," the signal transitions to "Yellow."
- From "Yellow," the signal transitions to either "Red" or "Green."

Based on these state transitions, we can design test cases to cover various scenarios:

1. Test case 1: Start with the traffic signal in the "Red" state. Transition the signal to "Green" and verify if the system updates the state correctly.
2. Test case 2: Start with the traffic signal in the "Green" state. Transition the signal to "Yellow" and verify if the system updates the state correctly.
3. Test case 3: Start with the traffic signal in the "Yellow" state. Transition the signal to "Red" and verify if the system updates the state correctly.
4. Test case 4: Start with the traffic signal in the "Yellow" state. Transition the signal to "Green" and verify if the system updates the state correctly.

By designing test cases that cover various state transitions, state transition testing ensures that the system behaves correctly as it moves between different states. This technique helps identify any issues related to state changes, ensuring the system's functionality aligns with the expected behavior of a traffic signal.

5. Use Case Testing: Use case testing validates system functionality based on specified use cases. Test cases are designed to verify if the system behaves as expected during execution of different use cases, ensuring compliance with use case requirements.

Generated By Syed Tabish Ishtiaq
Email:tabishishtiaq5@gmail.com
+923343308650

Example of Use Case Testing:-

Consider a system for an online shopping platform that includes various use cases. Let's focus on the "Add to Cart" use case. The steps for this use case might include:

1. User logs in to the system.
2. User searches for a product.
3. User selects a product.
4. User adds the product to the cart.
5. System updates the cart with the added product.

To perform use case testing for the "Add to Cart" use case, we can design test cases that cover different scenarios:

1. Test case 1: Start with a logged-in user. Search for a specific product, select it, and verify that it is added to the cart correctly.
2. Test case 2: Start with a logged-in user. Search for a product that is out of stock, select it, and verify that the system displays an appropriate message indicating the unavailability of the product.
3. Test case 3: Start with a logged-out user. Attempt to add a product to the cart and verify that the system prompts the user to log in.
4. Test case 4: Start with a logged-in user. Add multiple products to the cart, ensuring that the system updates the cart with all selected products correctly.

By designing test cases that correspond to different use case scenarios, use case testing ensures that the system behaves as expected during the execution of specific use cases. This technique helps validate the system's compliance with the requirements outlined in the use cases and ensures a satisfactory user experience for each use case

Structured based T\techniques:-

1. Statement Coverage: Ensures that every statement in the source code is executed at least once during testing.
   Example of Statement Coverage:-

   Consider a simple function in a programming language that calculates the square of a number:
   ```
   python
   def calculate_square(number):
       result = number * number
       return result
   ```
   To achieve statement coverage, we would design test cases that ensure every statement in the code is executed at least once during testing. Here's an example set of test cases:

   1. Test case 1: Test the function with input `2`. The statement `result = number * number` is executed, and the result is returned.
   2. Test case 2: Test the function with input `-5`. The statement `result = number * number` is executed, and the result is returned.
   3. Test case 3: Test the function with input `0`. The statement `result = number * number` is executed, and the result is returned.

By designing these test cases, we can ensure that each statement in the source code, including the calculation and return statements, is executed at least once during testing. This provides coverage and helps identify any potential issues or bugs in the function's implementation.

2. Decision Coverage: Ensures that every decision or branch in the source code is executed at least once during testing.
   Example for Decision Coverage:-

   Consider a simple function in a programming language that determines whether a given number is positive or negative:

```
python
def check_sign(number):
    if number > 0:
        result = "Positive"
    else:
        result = "Negative"
    return result
```

   To achieve decision coverage, we would design test cases that ensure every decision or branch in the code is executed at least once during testing. Here's an example set of test cases:

   1. Test case 1: Test the function with input `5`. The condition `number > 0` evaluates to true, and the code executes the positive branch. The result `"Positive"` is returned.
   2. Test case 2: Test the function with input `-3`. The condition `number > 0` evaluates to false, and the code executes the negative branch. The result `"Negative"` is returned.

   By designing these test cases, we can ensure that every decision point or branch in the source code, in this case, the if-else statement, is executed at least once during testing. This provides coverage and helps identify any potential issues or bugs related to the decision-making logic of the function.

3. Condition Coverage: Ensures that each condition in a decision statement is tested, including both true and false evaluations.
   Example for Condition Coverage:-

   Consider a simple function in a programming language that determines whether a given number is positive, negative, or zero:

```
python
def check_number(number):
    if number > 0:
        result = "Positive"
    elif number < 0:
        result = "Negative"
    else:
        result = "Zero"
    return result
```

Generated By Syed Tabish Ishtiaq
Email:tabishishtiaq5@gmail.com
+923343308650

To achieve condition coverage, we would design test cases that ensure each condition in the decision statement is tested, including both true and false evaluations. Here's an example set of test cases:

1. Test case 1: Test the function with input `5`. The condition `number > 0` evaluates to true, and the positive branch is executed. The result `"Positive"` is returned.
2. Test case 2: Test the function with input `-3`. The condition `number > 0` evaluates to false, and the condition `number < 0` evaluates to true. The negative branch is executed, and the result `"Negative"` is returned.
3. Test case 3: Test the function with input `0`. Both conditions `number > 0` and `number < 0` evaluate to false. The else branch is executed, and the result `"Zero"` is returned.

By designing these test cases, we can ensure that each condition in the decision statement, including both true and false evaluations, is tested during testing. This provides coverage and helps identify any potential issues or bugs related to the conditions and their corresponding branches in the function.

4. Multiple Condition Coverage: Ensures that all possible combinations of conditions in a decision statement are tested, including both true and false evaluations.
   Example for Multiple Condition Coverage:-

Consider a simple function in a programming language that determines the category of a given student based on their marks and attendance:

```
python
def get_student_category(marks, attendance):
    if marks >= 90 and attendance >= 90:
        category = "Outstanding"
    elif marks >= 70 and attendance >= 80:
        category = "Good"
    elif marks >= 50 or attendance >= 50:
        category = "Average"
    else:
        category = "Below Average"
    return category
```

To achieve multiple condition coverage, we would design test cases that ensure all possible combinations of conditions in the decision statement are tested, including both true and false evaluations. Here's an example set of test cases:

1. Test case 1: Test the function with marks `95` and attendance `95`. Both conditions `marks >= 90` and `attendance >= 90` evaluate to true. The category `"Outstanding"` is returned.
2. Test case 2: Test the function with marks `80` and attendance `85`. The condition `marks >= 90` evaluates to false, but the condition `marks >= 70` and `attendance >= 80` evaluate to true. The category `"Good"` is returned.
3. Test case 3: Test the function with marks `60` and attendance `75`. Both conditions `marks >= 70` and `attendance >= 80` evaluate to false, but the condition `marks >= 50` or `attendance >= 50` evaluates to true. The category `"Average"` is returned.

Generated By Syed Tabish Ishtiaq
Email:tabishishtiaq5@gmail.com
+923343308650

4. Test case 4: Test the function with marks `40` and attendance `40`. All conditions `marks >= 70`, `attendance >= 80`, and `marks >= 50` or `attendance >= 50` evaluate to false. The category `"Below Average"` is returned.

By designing these test cases, we can ensure that all possible combinations of conditions in the decision statement, including both true and false evaluations, are tested during testing. This provides coverage and helps identify any potential issues or bugs related to the complex decision-making logic of the function.

Experience Based Techniques:-

Error Guessing: Testers use their experience and intuition to guess and target potential errors or defects in the software, based on common mistakes, past issues, and error-prone areas. This technique helps uncover defects that may be missed by formal methods.

Example for Error Guessing:-

Imagine a software application that allows users to register for an online shopping platform. Testers with experience in similar registration processes can use Error Guessing to anticipate potential errors or defects. They may guess that users might encounter issues with:

1. Invalid email format: Testers can design test cases to intentionally enter email addresses with incorrect formats, such as missing the "@" symbol or including invalid characters.
2. Weak password validation: Testers can create test cases to test the application's response to weak passwords, such as using easily guessable passwords or passwords that do not meet the required complexity criteria.
3. Error handling: Testers can explore the system's response to unexpected scenarios, such as entering special characters or excessively long inputs in registration fields.
4. Duplicate username: Testers can intentionally attempt to register with a username that already exists in the system to verify if the application properly detects and handles such cases.

By using their experience and intuition, testers can design test cases specifically targeting these potential errors. Through Error Guessing, they can uncover defects or issues that might not be easily identified using other formal testing techniques. This technique helps improve the overall quality of the software by anticipating and addressing potential problems based on testers' experience.

Exploratory Testing: Testers dynamically design, execute, and evaluate tests based on their knowledge and intuition, without predefined test cases. They explore the software, perform ad-hoc tests, and adapt their approach based on immediate feedback. This technique helps quickly find defects and understand software behavior.

Example for Exploratory Testing:-

Imagine a mobile banking application that allows users to perform various transactions, such as checking account balance, transferring funds, and paying bills. Testers can apply

Generated By Syed Tabish Ishtiaq
Email:tabishishtiaq5@gmail.com
+923343308650

Exploratory Testing to gain insights into the application's behavior and identify potential defects.

1. Testers launch the mobile banking application and start exploring its user interface and features. They interact with different screens, buttons, and menus to familiarize themselves with the application's functionality.
2. Testers perform ad-hoc tests by trying different combinations of actions and inputs. For example, they may transfer funds from one account to another while simultaneously checking the account balance to ensure that the transaction is reflected correctly.
3. While performing the tests, testers pay attention to the application's response, including any error messages, delays, or unexpected behaviors. They use their knowledge and intuition to identify potential issues or defects, such as incorrect calculations, missing validation checks, or UI inconsistencies.
4. Testers adapt their testing approach based on immediate feedback and insights gained during testing. If they discover an issue, they may investigate further, perform additional tests, or modify their test scenarios to uncover the root cause or related defects.
5. Throughout the exploratory testing session, testers document their observations, findings, and any defects encountered. They can provide detailed feedback to the development team, helping them understand the issues and improve the software's quality.

Exploratory Testing allows testers to dynamically explore and evaluate the software based on their knowledge and intuition. By performing ad-hoc tests and adapting their approach, they can quickly identify defects, gain a better understanding of the software's behavior, and contribute valuable insights to improve the application's overall quality.

What is test planning?

Test Planning is the process of defining the overall approach and strategy for testing a software application or system.

What is STLC?

STLC, or Software Testing Life Cycle, is a structured approach to conducting software testing activities. It consists of a set of phases and activities that are designed to ensure the quality, reliability, and functionality of the software being developed. The primary goal of STLC is to identify defects, validate the software against predefined requirements, and ensure that it performs as intended.

What happens in all STLC Phases?

1. Requirement Analysis Phase:

    - Entry Criteria: Detailed requirements, functional specifications, and system design documents are available.
    - Activities: Review and understand the requirements, identify testable features, clarify ambiguities, and create a requirement understanding document.

- Exit Criteria: Requirement understanding document, requirement traceability matrix.
- Deliverables: Requirement understanding document, requirement traceability matrix.

2. Test Planning Phase:
    - Entry Criteria: Completed requirement analysis phase, testable software/application, and test environment setup.
    - Activities: Identify test objectives, define test strategies, determine test scope, estimate effort and resources, create a test plan, and identify test deliverables.
    - Exit Criteria: Test plan document, test estimation, test schedule, and test deliverables identified.
    - Deliverables: Test plan document, test estimation, test schedule, and test deliverables identified.
3. Test Case Development Phase:
    - Entry Criteria: Completed test planning phase, test environment setup, and available test scenarios.
    - Activities: Identify test scenarios, design test cases, review and refine test cases, and prepare test data and test scripts.
    - Exit Criteria: Test cases created, reviewed, and approved. Test data and test scripts prepared.
    - Deliverables: Test cases, test data, and test scripts.
4. Test Environment Setup Phase:
    - Entry Criteria: Test environment requirements defined, hardware/software infrastructure available, and test data prepared.
    - Activities: Set up the required test environment, install necessary hardware and software, configure test tools, and prepare test data.
    - Exit Criteria: Test environment ready for execution, including necessary hardware, software, and test data.
    - Deliverables: Configured test environment, test data.

5. Test Execution Phase:
    - Entry Criteria: Completed test case development phase, test environment setup, and availability of test scripts and test data.
    - Activities: Execute test cases, record test results, log defects, perform regression testing, and track progress.
    - Exit Criteria: Test execution completed, test results logged, and defects logged in the defect tracking system.
    - Deliverables: Test execution reports, defect reports.

6. Test Cycle Closure Phase:
    - Entry Criteria: Completed test execution phase, identified and resolved defects, and test execution report ready.
    - Activities: Analyze test results, generate test closure reports, document lessons learned, and archive test artifacts.

Generated By Syed Tabish Ishtiaq
Email:tabishishtiaq5@gmail.com
+923343308650

- Exit Criteria: Test closure report prepared, lessons learned documented, and test artifacts archived.
- Deliverables: Test closure report, lessons learned document, archived test artifacts.

What are the important tasks in the Test Planning stage?

- Define Testing Objectives: Clearly state testing goals aligned with project requirements and stakeholder expectations.
- Identify Scope and Test Coverage: Determine which features, functionalities, and modules to test and establish coverage criteria.
- Create a Test Strategy: Develop an overarching approach for testing, including levels, types, techniques, and any constraints.
- Estimate Test Effort and Resources: Assess the effort needed for testing considering software complexity, available resources, and time/budget constraints. Allocate resources accordingly.
- Define Test Deliverables: Identify and document test deliverables, such as Test Plans, Test Cases, Test Scripts, and Test Data, specifying their format, structure, and content.
- Create a Test Schedule: Establish a detailed timeline for testing, considering dependencies and allowing sufficient time for test execution, defect management, and retesting.
- Identify Test Environments and Tools: Determine the required test environments, including hardware, software, and network configurations, and identify relevant test tools and frameworks.
- Define Test Entry and Exit Criteria: Establish criteria for starting testing (entry criteria) and determining completion (exit criteria), including development completion, environment readiness, and quality thresholds.
- Analyze and Mitigate Risks: Identify potential risks affecting testing, assess their severity and likelihood, and define strategies to minimize their impact.
- Establish Communication and Collaboration: Set up effective channels to communicate with stakeholders, development team members, and other relevant parties, ensuring awareness of test planning activities, timelines, and expectations.

What are the reference documents for Test Planning?

1. Requirements Documents
2. Business or Functional Specifications
3. Use Cases or User Stories
4. Design Documents
5. Test Strategy
6. Test Case Templates
7. Test Data Requirements
8. Project Schedule
9. Defect Management Process
10. Test Environment Specifications
11. Test Exit Criteria

What are the considerable factors for Test Estimations?

1. Size and Complexity of the Software

2. Scope of Testing
3. Availability of Resources
4. Time Constraints
5. Budget Constraints
6. Test Environment Setup and Configuration
7. Dependencies on Other Teams or Activities
8. Level of Automation
9. Risk Assessment
10. Historical Data and Metrics

Who is the author of the Test Plan Document?

The Test Lead or Team Lead is typically responsible for authoring the Test Plan Document.

Who approves the Test Plan document?

Generally, the Project Manager approves the Test Plan Document after the review process.

What is Test Point Analysis?

Test Point Analysis is a formula-based test estimation method based on function point analysis.

What is Software Test Process?

The Software Test Process encompasses various activities, including test planning and control, test analysis and design, test implementation and execution, evaluating exit criteria and reporting, and test closure activities.

What is Function Point Analysis?

Function Point Analysis is a method used to measure the size of the functionality of an information system. It is independent of the technology used and can be used for productivity measurement, resource estimation, and project control.

What is the Entry criteria in Test Plan?

Entry criteria in a Test Plan refer to the set of generic and specific conditions that must be met before a process, such as the test phase, can proceed. Entry criteria help prevent starting a task that would require more effort to fix than the effort needed to address the failed entry criteria.

What is Exit criteria in Test Plan?

Exit criteria in a Test Plan are the set of generic and specific conditions agreed upon with stakeholders to determine when a process can be considered officially completed. Exit criteria prevent considering a task complete if there are outstanding parts that have not been finished. Exit criteria are used for reporting and planning when to stop testing.

What is a Test deliverable?

A Test deliverable refers to any test (work) product that must be delivered to someone other than the author of the test (work) product.

What are the Test deliverables in Software Test Process?

Test deliverables in the Software Test Process include the Test Plan document, Test cases, Test design specifications, Tools and their outputs, Simulators, Static and dynamic generators, Error logs and execution logs, Problem reports, and corrective actions.

Generated By Syed Tabish Ishtiaq
Email:tabishishtiaq5@gmail.com
+923343308650

What is a Master Test Plan?

A Master Test Plan is a test plan that typically addresses multiple test levels within a project.

What is Wide Band Delphi?

Wide Band Delphi is an expert-based test estimation technique that aims to make accurate estimations by leveraging the collective wisdom of the team members.

What is difference between STLC and SDLC?

SDLC (Software Development Life Cycle)	**STLC (Software Testing Life Cycle)**
Focus	Focus
Covers the entire software development process, from gathering requirements to deploying and maintaining the software.	Specifically concentrates on the testing phase of software development.
Scope	Scope
Includes all stages of development, from the initial concept to the final product.	Encompasses activities related to software testing, including test planning, test case development, test execution, defect tracking, and test closure.
Activities	Activities
Analyzing requirements, designing the system, coding, integrating components, testing, deploying the software, and maintaining it.	Planning tests, designing test cases, executing tests, tracking and managing defects, and closing the testing phase.
Objective	Objective
Main goal is to deliver a high-quality software product that meets the specified requirements within the given time and budget.	Ensure the quality, reliability, and functionality of the software by identifying defects and validating it against the specified requirements.
Stakeholders	Stakeholders
Various stakeholders, such as business analysts, developers, project managers, and end-users.	Stakeholders include test analysts, test engineers, QA leads, and other members of the testing team.

Write test cases for pen?

1. Verify that the ballpoint pen writes smoothly on a standard sheet of paper.
2. Test the pen's writing performance on recycled paper.
3. Check the pen's performance on glossy paper.
4. Test the pen's writing performance on a rough or textured surface.

Generated By Syed Tabish Ishtiaq
Email:tabishishtiaq5@gmail.com
+923343308650

5. Verify that the pen writes consistently at different writing speeds.
6. Test the pen's writing performance at different ambient temperatures.
7. Verify that the pen can write underwater.
8. Test the pen's performance on oily or greasy surfaces.
9. Verify that the pen can write at different angles (e.g., upright, tilted, or upside-down).
10. Check the pen's performance when writing with light pressure.
11. Test the pen's performance when writing with heavy pressure.
12. Verify that the pen writes smoothly without skipping or clogging.
13. Test the pen's performance on a vertical surface (e.g., writing on a whiteboard).
14. Verify that the pen's ink does not smudge or smear easily after drying.
15. Test the pen's performance on different types of fabric (e.g., cotton, polyester).
16. Verify that the pen can write on plastic surfaces without smearing or scratching.
17. Test the pen's performance on glass surfaces.
18. Verify that the pen's ink is resistant to fading over time.
19. Test the pen's performance on a variety of colored papers.
20. Verify that the pen can write on dark-colored paper with good visibility.
21. Test the pen's performance at high altitudes (e.g., in an airplane).
22. Verify that the pen's ink is resistant to water damage or smudging when exposed to water.
23. Test the pen's performance when writing for an extended period without interruption.
24. Verify that the pen's ink does not leak or drip.
25. Test the pen's performance after being left uncapped for an extended period.
26. Verify that the pen's ink is easy to erase (if applicable).
27. Test the pen's performance when writing in extreme humidity.
28. Verify that the pen's ink is resistant to fading when exposed to sunlight.
29. Test the pen's performance when writing on thermal paper.
30. Verify that the pen's ink does not bleed through thin or delicate paper.
31. Test the pen's performance when writing on carbon paper.
32. Verify that the pen's ink is smudge-proof after drying.
33. Test the pen's performance when writing on different types of envelopes.
34. Verify that the pen's ink does not dry out quickly.
35. Test the pen's performance on coated paper surfaces.
36. Verify that the pen's ink is resistant to chemicals (e.g., alcohol, cleaners).
37. Test the pen's performance when writing on a sticky or adhesive surface.
38. Verify that the pen's ink does not transfer onto the user's hands during writing.
39. Test the pen's performance when writing on thermal labels.
40. Verify that the pen's ink remains visible even after being exposed to water.
41. Test the pen's performance when writing on metal surfaces.
42. Verify that the pen's ink is resistant to fading when exposed to heat.
43. Test the pen's performance on different types of cardboard.
44. Verify that the pen's ink does not dry out when left unused for an extended period.
45. Test the pen's performance when writing on transparent plastic sheets.
46. Verify that the pen's ink does not smudge when highlighted over.
47. Test the pen's performance when writing on rough wood surfaces.
48. Verify that the pen's ink does not transfer onto the opposite side of thin paper.
49. Test the pen's performance when writing on chalkboards or blackboards.
50. Verify that the pen's ink does not bleed when exposed to moisture.

Generated By Syed Tabish Ishtiaq
Email:tabishishtiaq5@gmail.com
+923343308650

51. Test the pen's performance on adhesive labels.
52. Verify that the pen's ink does not clump or blob during writing.
53. Test the pen's performance when writing on different types of rubber surfaces.
54. Verify that the pen's ink is resistant to fading when exposed to air pollution.
55. Test the pen's performance when writing on laminated surfaces.
56. Verify that the pen's ink is quick-drying to prevent accidental smudging.
57. Test the pen's performance when writing on different types of leather.
58. Verify that the pen's ink does not transfer onto other documents when stacked.
59. Test the pen's performance when writing on textured plastic surfaces.
60. Verify that the pen's ink does not leak during air travel (e.g., changes in cabin pressure).
61. Test the pen's performance when writing on vinyl surfaces.
62. Verify that the pen's ink remains visible even after being exposed to extreme cold temperatures.
63. Test the pen's performance when writing on ceramic surfaces.
64. Verify that the pen's ink does not bleed through multiple layers of thin paper.
65. Test the pen's performance when writing on magnetic surfaces.
66. Verify that the pen's ink is resistant to smearing when highlighted with different types of highlighters.
67. Test the pen's performance when writing on stone or concrete surfaces.
68. Verify that the pen's ink does not clump or dry out when exposed to air.
69. Test the pen's performance when writing on silicone surfaces.
70. Verify that the pen's ink remains visible even after being exposed to extreme heat temperatures.
71. Test the pen's performance when writing on food packaging materials (e.g., plastic wrap, aluminum foil).
72. Verify that the pen's ink does not transfer onto the user's fingers when writing for an extended period.
73. Test the pen's performance when writing on wax or wax-coated surfaces.
74. Verify that the pen's ink does not smudge or smear when subjected to friction or rubbing.
75. Test the pen's performance when writing on erasable surfaces (e.g., whiteboard, chalkboard).
76. Verify that the pen's ink does not dry out or evaporate quickly in dry environments.
77. Test the pen's performance when writing on rubberized surfaces.
78. Verify that the pen's ink is resistant to fading when exposed to ultraviolet (UV) light.
79. Test the pen's performance when writing on polystyrene or Styrofoam surfaces.
80. Verify that the pen's ink does not bleed through carbonless copy paper.
81. Test the pen's performance when writing on electronic touchscreens (e.g., smartphones, tablets).
82. Verify that the pen's ink does not transfer onto the user's clothing when clipped to a pocket.
83. Test the pen's performance when writing on wood veneer or laminate surfaces.
84. Verify that the pen's ink does not smudge or blur when subjected to erasing or correction fluid.
85. Test the pen's performance when writing on plastic-coated or laminated photo paper.
86. Verify that the pen's ink remains visible even after being exposed to harsh cleaning agents.
87. Test the pen's performance when writing on plastic packaging materials (e.g., blister packs, shrink wrap).

Generated By Syed Tabish Ishtiaq
Email:tabishishtiaq5@gmail.com
+923343308650

88. Verify that the pen's ink does not fade or deteriorate when exposed to humidity and moisture.

Generate test cases for Mobile Phone?

1. Power On/Off Test:
- Verify that the mobile phone powers on correctly when the power button is pressed.
- Ensure that the mobile phone powers off correctly when the power button is held.
2. Screen Display Test:
- Check that the mobile phone's screen displays content clearly and without any dead pixels.
- Verify that the screen brightness adjusts properly in different lighting conditions.
3. Touchscreen Test:
- Test the responsiveness and accuracy of the touchscreen by tapping, swiping, and pinching gestures across the entire screen.
4. Home Button Test:
- Verify that pressing the home button returns the user to the home screen from any application.
- Ensure that the home button functions consistently and without delays.
5. Volume Control Test:
- Test the volume control buttons to verify that they adjust the volume levels correctly.
- Check if the volume buttons function properly during phone calls, media playback, and notifications.
6. Silent Mode Test:
- Test the functionality of the silent mode switch or option to ensure it effectively mutes all sound notifications.
7. Camera Test:
- Test the front and rear cameras to verify that they capture photos and videos with the correct resolution and quality.
- Check if camera settings (e.g., flash, HDR, autofocus) work as intended.
8. Call Functionality Test:
- Verify that the mobile phone can make and receive calls without any issues.
- Test call clarity, speakerphone functionality, and the ability to switch between audio sources.
9. Messaging Test:
- Test the messaging app to ensure that sending and receiving text messages, multimedia messages, and group chats work properly.
- Verify that attachments (e.g., images, videos, documents) can be sent and received without errors.
10. Internet Connectivity Test:
- Test Wi-Fi connectivity to ensure the mobile phone can connect to different networks and browse the internet without interruptions.
- Test cellular data connectivity (3G, 4G, 5G) to verify proper data transfer speeds and stability.
11. GPS Functionality Test:
- Test the GPS functionality of the mobile phone by using location-based services and mapping applications.
- Verify that the phone accurately tracks and displays the user's location.
12. App Installation and Uninstallation Test:

Generated By Syed Tabish Ishtiaq
Email:tabishishtiaq5@gmail.com
+923343308650

- Test the ability to install and uninstall applications from app stores or external sources.
- Verify that apps install and launch correctly without any compatibility issues.

13. Battery Life Test:
- Test the mobile phone's battery life by using it under various usage scenarios (e.g., browsing, gaming, video playback).
- Verify that the battery drains at an expected rate and charges correctly.

14. App Performance Test:
- Test the performance and responsiveness of pre-installed and downloaded applications under different usage conditions.
- Verify that apps load quickly, function smoothly, and do not crash unexpectedly.

15. Storage Capacity Test:
- Test the mobile phone's storage capacity by filling it with various file types (e.g., photos, videos, documents) until it reaches its limit.
- Verify that the phone properly handles storage limitations and prompts the user to free up space.

16. Audio Playback Test:
- Test the mobile phone's ability to play audio files (e.g., music, podcasts, audiobooks) through the built-in speaker and headphones.
- Verify that the audio quality is clear and without distortion.

17. Video Playback Test:
- Test the mobile phone's ability to play different video formats and resolutions.
- Verify that videos play smoothly, without lag or audio/video synchronization issues.

18. Bluetooth Connectivity Test:
- Test the mobile phone's Bluetooth functionality by connecting to various devices (e.g., headphones, speakers, car systems) and transferring files.
- Verify that the connections are stable, and data transfer is successful.

19. External Ports Test:
- Test the functionality of external ports, such as the charging port, audio jack, and any other available ports.
- Verify that the ports work correctly and are compatible with respective devices.

20. Operating System Updates Test:
- Test the mobile phone's ability to receive and install operating system updates.
- Verify that updates install correctly and do not cause any compatibility issues with existing apps or settings.

21. Security Features Test:
- Test the mobile phone's security features, such as fingerprint sensors, facial recognition, or PIN/password locks.
- Verify that the security features work as intended and provide the necessary protection.

22. Accessibility Features Test:
- Test the accessibility features of the mobile phone, such as screen readers, color adjustments, and text resizing.
- Verify that the accessibility features function correctly and improve usability for users with disabilities.

23. Multitasking Test:
- Test the mobile phone's ability to handle multiple applications running simultaneously.

Generated By Syed Tabish Ishtiaq
Email:tabishishtiaq5@gmail.com
+923343308650

- Verify that switching between apps is smooth, and background apps do not excessively drain system resources.

24. Mobile Hotspot Test:

- Test the mobile phone's ability to function as a Wi-Fi hotspot and share the internet connection with other devices.
- Verify that the hotspot connection is stable and secure.

25. Mobile Payments Test:

- Test the mobile phone's ability to perform mobile payments using NFC or other technologies.
- Verify that payment transactions are successful and secure.

26. SIM Card Functionality Test:

- Test the mobile phone's ability to detect and recognize SIM cards from different carriers.
- Verify that SIM cards are properly recognized, and cellular network connectivity is established.

27. Network Roaming Test:

- Test the mobile phone's ability to switch to a different network when roaming internationally.
- Verify that the phone connects to available networks seamlessly without any issues.

28. USB Connectivity Test:

- Test the mobile phone's USB connectivity by connecting it to a computer and verifying data transfer capabilities.
- Ensure that the phone is recognized by the computer and can transfer files.

29. NFC Functionality Test:

- Test the mobile phone's NFC (Near Field Communication) functionality by performing transactions or transferring data with compatible devices.
- Verify that NFC connections are established reliably and that transactions or data transfers are successful.

30. App Permissions Test:

- Test the mobile phone's app permission system by granting and revoking permissions for various applications.
- Verify that apps request permissions correctly and function properly based on the granted permissions.

31. Ingress Protection (IP) Rating Test:

- Test the mobile phone's resistance to dust and water based on its IP rating.
- Verify that the phone can withstand specified dust levels and water immersion according to its rating.

32. Sensor Functionality Test:

- Test the functionality of various sensors in the mobile phone, such as the accelerometer, gyroscope, proximity sensor, and ambient light sensor.
- Verify that the sensors work accurately and provide the necessary input to applications.

33. Gaming Performance Test:

- Test the mobile phone's performance and graphics capabilities by running demanding games.
- Verify that games run smoothly without lag or frame rate drops.

Generated By Syed Tabish Ishtiaq
Email:tabishishtiaq5@gmail.com
+923343308650

34. Screen Rotation Test:

- Test the mobile phone's ability to detect screen rotation and adjust the display accordingly.
- Verify that the screen rotates smoothly and without delay when the device is rotated.

35. User Interface (UI) Responsiveness Test:

- Test the responsiveness of the mobile phone's user interface, including app launches, menu navigation, and system interactions.
    - Verify that the UI responds promptly to user inputs without noticeable delays.