**Q1. What is Exploratory Testing?**

=> Exploratory testing is an approach to software testing that emphasizes the tester's freedom and creativity to explore the software application in order to discover defects, issues, and areas of potential improvement. It is a dynamic and ad-hoc testing technique that involves simultaneous test design, execution, and learning.

=> In exploratory testing, the tester does not rely on pre-defined test cases or scripts but instead relies on their experience, intuition, and domain knowledge to guide the testing process. The tester interacts with the software system as an end user would, exploring different features, inputs, and workflows in an unscripted manner.

**Q2.** **What is traceability matrix?**

=> A traceability matrix is a tool used in software development and testing to establish and track the relationships between various project artefacts, such as requirements, design documents, test cases, and defects. It provides a means to ensure that all requirements are appropriately addressed through the testing process and that any changes or issues can be traced back to their source.

=> The traceability matrix typically consists of a table or matrix format, where each row represents a requirement or a specific item being traced, and each column represents a project artefact, such as test cases or design documents. The intersections within the matrix indicate the relationships between the requirements and the associated artefacts.

=> By using a traceability matrix, project teams can improve transparency, manage changes effectively, ensure requirements coverage, and facilitate communication and coordination between different stakeholders involved in the software development lifecycle. It serves as a valuable reference and reporting tool throughout the project, helping to maintain traceability and support decision-making processes.

**Q3. What is Boundary value testing?**

=> Boundary value testing is a software testing technique that focuses on testing the boundaries or limits of input values to evaluate the behaviour of a system at those boundaries. The basic premise of boundary value testing is that defects are more likely to occur near the boundaries of input ranges rather than in the middle.

=> The technique involves selecting test cases that fall on or near the boundaries of the input domain, which includes both valid and invalid boundary values. The idea is to test the system's response to values at the lower and upper limits, as well as values just beyond those limits. By testing these critical boundary conditions, testers can identify potential issues related to data validation, calculations, decision-making, and system stability.

=> Boundary value testing aims to detect errors that may arise due to incorrect handling or processing of values near the boundaries. By focusing on the boundary conditions, the technique helps uncover defects that might not be apparent through testing within the middle range of input values.

**Q4. What is Equivalence partitioning testing?**

=> Equivalence partitioning testing is a software testing technique that aims to reduce the number of test cases while still achieving sufficient test coverage. It involves dividing the input data into groups or partitions, where each partition is expected to exhibit similar behaviour. By selecting representative test cases from each partition, testers can effectively validate the system's functionality and identify defects.

=> The principle behind equivalence partitioning is that if a system behaves correctly for one value within a partition, it should behave correctly for all other values within the same partition. Conversely, if a system fails for one value within a partition, it is likely to fail for all other values within that partition as well. Therefore, there is no need to test every individual value within a partition; instead, a single representative test case can suffice.

=> It's worth noting that equivalence partitioning is most effective when the input space is large and varied, and the system's behavior can be generalized based on certain input characteristics. It is commonly used in functional testing, where the focus is on validating inputs and their corresponding outputs.

=> Overall, equivalence partitioning testing is a valuable technique for optimizing test coverage while minimizing the number of test cases, enabling efficient and effective testing of software systems.

**Q5. What is Integration testing?**

=> Integration testing is a software testing technique that focuses on testing the interaction and collaboration between different software components or modules when they are integrated together. The purpose of integration testing is to identify defects, inconsistencies, and failures that may arise due to the interaction between these components.

=> In a software system, individual components or modules are developed and tested independently. Integration testing comes after unit testing, where individual components have already been tested in isolation. The goal of integration testing is to ensure that when these components are combined and integrated, they work together as expected and fulfil the system's requirements.

=> By performing integration testing, software development teams can verify that the components work together seamlessly, detect and resolve integration issues, and ensure the proper functioning of the integrated system.

**Q6. What determines the level of risk?**

=> The level of risk is determined by several factors that collectively influence the likelihood and impact of potential negative events or outcomes. While the specific factors may vary depending on the context, here are some common elements that contribute to assessing and determining the level of risk.

**Q7. What is Alpha testing?**

=> Alpha testing is a type of software testing performed by the software development organization or a select group of users at the developer's site. It is conducted before the software is released to the public or to a larger audience. The primary objective of alpha testing is to assess the software's overall functionality, performance, and usability in a controlled environment.

=> Alpha testing serves as an essential step in the software development lifecycle, allowing developers to gain insights into the software's performance and usability in a controlled environment. It helps identify and resolve critical issues before moving towards wider beta testing or a public release, leading to a more stable and reliable final product.

**Q8. What is beta testing?**

=> Beta testing is a type of software testing performed by a selected group of external users, often referred to as beta testers, who are not directly involved in the software development process. It is conducted after alpha testing and prior to the official release of the software to the general public. The purpose of beta testing is to gather user feedback, identify potential issues, and assess the software's performance in real-world scenarios.

=> Overall, beta testing plays a crucial role in the software development lifecycle by engaging real users and collecting feedback to improve the software's quality and user experience before its official release to the public.

**Q9. What is component testing?**

=> Component testing, also known as module testing or unit testing, is a software testing technique that focuses on testing individual components or modules of a software system in isolation. The purpose of component testing is to verify the functionality, behaviour, and correctness of each component independently to ensure that they work as intended.

=> Component testing is an integral part of the software testing process, complementing other testing techniques such as integration testing and system testing. It provides a solid foundation for building reliable and robust software systems by validating the functionality and correctness of individual components before they are combined.

**Q10. What is functional system testing?**

=> Functional system testing is a software testing technique that focuses on verifying the functional requirements and behaviour of a software system as a whole. It involves testing the system's features, functions, and interactions to ensure that it meets the specified functional requirements and operates as intended.

=> Functional system testing is an essential part of the software testing process, complementing other testing techniques such as unit testing, integration testing, and acceptance testing. It focuses on validating the system's functional requirements and behaviour to ensure that the software system performs as expected and meets user expectations.

**Q11. What is Non-Functional Testing?**

=> Non-functional testing is a type of software testing that focuses on evaluating the characteristics and attributes of a software system, rather than its specific functionalities. It aims to assess the system's performance, reliability, usability, scalability, security, and other non-functional aspects that influence the overall quality and user experience. Non-functional testing helps ensure that the software system meets the desired standards and performance expectations.

=> To address this issue, performance testing is carried out to check &  
fine tune system response times. The goal of performance testing is  
to reduce response time to an acceptable level.

=> Types of Non-functional testing are  
∙ Performance Testing  
∙ Load Testing  
∙ Volume Testing  
∙ Stress Testing  
∙ Security Testing  
∙ Installation Testing  
∙ Penetration Testing  
∙ Compatibility Testing  
∙ Migration Testing

**Q12. What is GUI Testing?**

=> GUI testing, also known as Graphical User Interface testing, is a software testing technique that focuses on evaluating the functionality, usability, and responsiveness of a software application's graphical user interface. The GUI refers to the visual elements, such as windows, buttons, menus, dialog boxes, and other interactive components, through which users interact with the software.

=> The goal of GUI testing is to ensure that the graphical user interface behaves as intended, providing a smooth and intuitive user experience. It involves checking various aspects of the interface, including layout, design, navigation, input fields, buttons, error handling, and overall visual appearance.

=> GUI testing can be performed manually, where testers interact with the software application and observe its behavior, or automated using specialized tools that simulate user actions and verify the expected outcomes. Both approaches have their advantages and are often used in combination to achieve thorough testing coverage.

=> Overall, GUI testing is crucial for ensuring that the software's graphical user interface meets user expectations, functions correctly, and provides a positive user experience.

**Q13. What is Adhoc testing?**

=> Adhoc testing, also known as exploratory testing is an informal and unscripted software testing approach that involves testing without any predefined test cases or plans. It is a technique where testers use their domain knowledge, experience, and intuition to perform on-the-spot testing based on their understanding of the system under test.

=> In adhoc testing, testers are not bound by any specific test scenarios or scripts. Instead, they interact with the software application in an unplanned and spontaneous manner, exploring various features, functionalities, and user interactions. The primary objective of adhoc testing is to uncover defects, bugs, or usability issues that might not be easily detected through formal test cases.

=> However, it's important to note that adhoc testing should not replace structured testing methods entirely. While adhoc testing is valuable in finding unexpected issues, it may lack the thoroughness and repeatability of formal test cases. Therefore, a balanced approach that combines both adhoc and structured testing techniques is often recommended for comprehensive software testing.

**Q14. What is load testing?**

=> Load testing is a software testing technique that focuses on evaluating the performance and behaviour of a system under expected and anticipated high load conditions. It involves simulating realistic user loads and measuring the system's response to determine its ability to handle the anticipated workload.

=> The primary goal of load testing is to identify performance bottlenecks, determine the system's capacity limits, and ensure that it can handle the expected number of concurrent users, transactions, or requests without degradation in performance. By subjecting the system to various load levels, testers can analyse its behaviour, response times, resource utilization, and overall stability.

=> Load testing is crucial for ensuring that a system can handle the expected user load without performance degradation or failures. By proactively identifying performance bottlenecks and optimizing the system, organizations can deliver a high-quality, reliable, and responsive software application to their users.

**Q15. What is stress Testing?**

=> Stress testing is a software testing technique that aims to evaluate the stability, robustness, and reliability of a system under extreme and unfavourable conditions. It involves subjecting the system to high loads, excessive data volumes, or resource limitations to identify its breaking point and assess its behaviour under stress.

=> The primary goal of stress testing is to determine how the system handles and recovers from adverse conditions, such as high user loads, data spikes, insufficient resources, or network failures. By pushing the system to its limits, testers can uncover weaknesses, bottlenecks, and potential failure points that may impact its performance, stability, or data integrity.

=> Stress testing helps organizations identify and address system weaknesses before them impact real users or critical business operations. By subjecting the system to extreme conditions, it ensures that the software application can handle unexpected stress factors and continues to perform reliably under adverse circumstances.

**Q16. What is white box testing and list the types of white box testing?**

=> White box testing, also known as clear box testing or structural testing, is a software testing technique that focuses on examining the internal structure, design, and implementation details of a software application. It involves testing the code, logic, and paths within the system to ensure that they function as intended and meet the specified requirements.

=> White box testing requires knowledge of the internal workings of the system and access to the source code. Testers typically have a detailed understanding of the system's architecture, algorithms, and programming languages to effectively design and execute white box tests.

=> List of white box testing are hear under:

1. Statement Coverage Testing

2. Decision coverage Testing

3. Condition Coverage Testing

**Q17. What is black box testing? What are the different black box testing techniques?**

=> Black box testing is a software testing technique that focuses on evaluating the functionality and behaviour of a software application without examining its internal structure or implementation details. Testers treat the system as a "black box" and test it based on its specified inputs, expected outputs, and the system's external behaviour, without knowledge of its internal workings.

=> Black box testing is primarily concerned with validating whether the software application meets the specified requirements and performs as expected from the end-user's perspective. Testers do not have access to the source code and rely solely on the system's inputs and outputs to design and execute tests.

=> There are four specification-based or black-box techniques:

* Equivalence partitioning
* Boundary value analysis
* Decision tables
* State transition testing
* Use-case Testing
* Other Black Box Testing
* Syntax or Pattern Testing

**Q18. Mention what are the categories of defects?**

There are 5 types of defects which are hear under:-

1. Data Quality/Database Defects: Deals with improper handling of data in the database.
2. Critical Functionality Defects: The occurrence of these bugs hampers the crucial functionality of the application.
3. Functionality Defects: These defects affect the functionality of the application.
4. Security Defects: Application security defects generally involve improper handling of data sent from the user to the application. These defects are the most severe and given highest priority for a fix.
5. User Interface Defects: As the name suggests, the bugs deal with problems related to UI are usually considered less severe.

**Q19. Mention what bigbang testing is?**

=> Big Bang testing is a software testing approach where all the individual components or modules of a software system are tested together as a whole. It is a type of integration testing method that involves combining all the modules at once and executing tests on the integrated system. This approach is in contrast to incremental testing, where modules are tested individually and then gradually integrated and tested in a step-by-step manner.

=> In the Big Bang testing approach, the focus is on verifying the interactions and dependencies among different modules when they are integrated. The purpose is to identify any issues or defects that may arise due to the integration of modules and to ensure that the system functions correctly as a whole.

=> Overall, Big Bang testing is considered a high-risk approach and is typically used in situations where the system's complexity and dependencies are relatively low, and time constraints are tight.

**Q20. What is the purpose of exit criteria?**

=> Exit criteria, also known as test completion criteria, are predefined conditions or metrics that must be met in order to determine when to stop testing activities for a particular phase, level, or project. The purpose of exit criteria is to provide a set of guidelines or benchmarks to evaluate whether testing has been performed adequately and whether the software system is ready to progress to the next phase or be released.

=> Here are the main purposes of exit criteria in software testing:

**1. Evaluation of Test Completion:** Exit criteria help in determining when testing activities for a specific phase or level are considered complete. By defining specific conditions that need to be met, such as achieving a certain level of test coverage or executing a specified number of test cases, exit criteria enable testers and stakeholders to assess the progress and completeness of testing efforts.

**2. Quality Assessment:** Exit criteria serve as quality gates for evaluating the overall quality of the software system. They help in measuring the level of defects, severity of issues, and other quality indicators. Meeting the exit criteria demonstrates that the software system meets the predefined quality standards and is ready for the next phase or release.

**3. Risk Management:** Exit criteria aid in managing risks associated with software testing. They help identify and assess the risks that need to be mitigated before proceeding further. Exit criteria may include requirements such as achieving a certain level of test coverage in critical areas or resolving high-priority defects that pose significant risks to the system.

**4. Decision Making:** Exit criteria provide a basis for making informed decisions about software release or progression to the next phase. By establishing clear and measurable criteria, stakeholders can make decisions on whether the system is stable enough, meets the required quality standards, and is ready for the next set of activities, such as user acceptance testing or deployment.

**5. Resource Allocation:** Exit criteria assist in optimizing resource allocation by ensuring that testing activities are not prolonged indefinitely. Once the exit criteria are met, testing resources can be efficiently redirected to other critical tasks, maximizing the efficiency of the testing process.

=> It is important to establish and document exit criteria at the beginning of a project or testing phase, with the involvement of key stakeholders. This ensures that there is a clear understanding of the goals and expectations, and provides a common reference point for evaluating the completion and readiness of the software system.

**Q21. When should "Regression Testing" be performed?**

1. Change in requirements and code is modified according to the requirement
2. New feature is added to the software
3. Defect fixing
4. Performance issue fix

**Q22. What are 7 key principles? Explain in detail?**

=> The 7 key principles refer to a set of guiding principles for software testing that are outlined in the ISTQB (International Software Testing Qualifications Board) Foundation Level syllabus. These principles provide a foundation for effective and efficient testing practices. Let's explore each principle in detail:

1. Testing Shows the Presence of Defects:

This principle highlights that testing is not aimed at proving the absence of defects but rather at demonstrating their existence. Testing helps in identifying defects, inconsistencies, and deviations from expected behaviour. By executing tests, defects are uncovered, enabling their identification, analysis, and subsequent correction.

2. Exhaustive Testing is Impossible:

It is practically impossible to test a software system under all possible conditions and scenarios. The principle acknowledges that complete or exhaustive testing, which covers every possible input combination and usage scenario, is unattainable. Instead, testing efforts should be focused on achieving optimal risk coverage within the available resources and time constraints.

3. Early Testing:

The principle emphasizes the importance of initiating testing activities as early as possible in the software development life cycle. Testing should not be limited to the final stages of development but should start early, ideally in parallel with requirements gathering, design, and coding. Early testing enables the early detection and resolution of defects, reducing the cost and impact of rework.

4. Defect Clustering:

This principle states that defects tend to cluster or concentrate in specific areas or components of a software system. Not all parts of the system are equally error-prone. By analysing defect patterns and trends, testers can focus their testing efforts on high-risk areas, modules, or functionalities, thereby optimizing test coverage and defect detection.

5. Pesticide Paradox:

The pesticide paradox principle suggests that if the same set of tests is repeatedly executed, over time, they become less effective at finding new defects. This is analogous to the idea that pests can become resistant to a particular pesticide with continuous exposure. To overcome this paradox, testing needs to evolve and adapt by periodically reviewing and refreshing test cases to ensure their effectiveness.

6. Testing is Context Dependent:

Testing effectiveness is influenced by various contextual factors, including the nature of the software, project constraints, stakeholder expectations, available resources, and time limitations. The principle recognizes that testing approaches, techniques, and priorities should be adapted and tailored based on the specific context of the project.

7. Absence-of-Errors Fallacy:

The principle cautions against assuming that the absence of detected defects implies the absence of all defects. Testing can never guarantee that a software system is completely defect-free. It is important to understand that testing helps in increasing confidence in the system's behaviour, identifying known defects, and reducing risks. However, it does not eliminate the possibility of unknown or unforeseen defects.

=> These principles serve as fundamental guidelines for effective testing and help shape testing strategies and approaches. They encourage testers to focus on defect identification, prioritize testing efforts, start testing early, adapt to changing needs, and maintain a realistic perspective on the limitations of testing. By applying these principles, testing can be more efficient, targeted, and aligned with the project's objectives.

**Q23. Difference between QA v/s QC v/s Tester**

|  |  |  |  |
| --- | --- | --- | --- |
| **SR** | **QA** | **QC** | **Tester** |
| **1.** | Activities which ensure the implementation of processes, procedures and standards in context to verification of developed software and intended requirements. | Activities which ensure the verification of developed software with respect to documented (or not in some cases) requirements. | Activities which ensure the identification of bugs/error/defects in the Software. |
| **2.** | Focuses on processes and procedures rather than conducting actual testing on the system. | Focuses on actual testing by executing Software with intend to identify bug/defect through implementation of procedures and process. | Focuses on actual testing. |
| **3.** | Process oriented activities. | Product oriented activities. | Product oriented activities. |
| **4.** | Preventive activities. | It is a corrective process. | It is a preventive process. |
| **5.** | It is a subset of Software Test Life Cycle (STLC). | QC can be considered as the subset of Quality Assurance. | Testing is the subset of Quality Control. |

**Q24. Difference between Smoke and Sanity?**

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

**Q25. Difference between verification and Validation.**

|  |  |  |
| --- | --- | --- |
| **Criteria** | **Verification** | **Validation** |
| **Definition** | The process of evaluating work-products (not the actual final product) of a development phase to determine whether they meet the specified requirements for that phase. | The process of evaluating software during or at the end of the development process to determine whether it satisfies specified business requirements. |
| **Objective** | To ensure that the product is being built according to the requirements and design specifications. In other words, to ensure that work products meet their specified requirements | To ensure that the product actually meets the user’s needs, and that the specifications were correct in the first place. In other words, to demonstrate that the product fulfils its intended use when placed in its intended environment. |
| **Question** | Are we building the product right? | Are we building the right product? |
| **Evaluation Items** | Plans, Requirement Specs, Design Specs, Code, Test Cases | The actual product/software. |
| **Activities** | Reviews Walkthroughs Inspections | Testing |

**Q26. Explain types of Performance testing.**

=> Performance testing is a type of software testing that focuses on evaluating the performance characteristics of a system under specific workloads or conditions. It helps identify performance bottlenecks, measure response times, throughput, and scalability, and ensure that the system meets the required performance criteria. There are several types of performance testing, each serving a specific purpose. Here are the main types:

1. Load Testing:

Load testing involves simulating realistic user loads on the system to measure its behaviour and performance under normal and peak usage conditions. It helps determine how the system handles concurrent user requests, assess response times, and identify performance degradation or bottlenecks. Load testing ensures that the system can handle the expected user load without significant performance issues.

2. Stress Testing:

Stress testing involves testing the system beyond its normal operational capacity to evaluate its behaviour and performance under extreme or unusual conditions. It aims to identify the system's breaking point and understand how it recovers from failure. Stress testing typically involves increasing the load, data volume, or transaction complexity to push the system's limits. It helps assess the system's stability, robustness, and resilience.

3. Soak Testing:

Soak testing, also known as endurance testing, involves subjecting the system to a sustained workload for an extended duration to evaluate its performance and stability over time. It aims to identify any performance degradation or issues that may occur due to long-duration usage, resource leaks, memory leaks, or other factors. Soak testing helps ensure that the system remains stable and performs well over extended periods, such as continuous usage or overnight processing.

4. Spike Testing:

Spike testing involves testing the system's performance when it experiences sudden and significant increases or spikes in user load or transaction volume. It helps assess how the system handles sudden surges and if it can scale up to meet the increased demand. Spike testing can reveal issues related to resource allocation, concurrency, and response times during high-traffic situations.

5. Scalability Testing:

Scalability testing is performed to assess the system's ability to handle increased workload or user load by adding more resources such as servers, CPUs, or memory. It helps determine how the system scales and maintains performance as the workload grows. Scalability testing is essential for systems that are expected to handle increasing user demand or for applications that need to support future growth.

6. Volume Testing:

Volume testing involves testing the system's performance with a large amount of data to assess its behaviour and response times under such conditions. It helps identify how the system handles data storage, retrieval, and processing. Volume testing can uncover performance issues related to database performance, disk space, caching, and data handling algorithms.

7. Configuration Testing:

Configuration testing focuses on testing the system's performance under different hardware or software configurations. It aims to determine how different configurations impact the system's performance and to identify any configuration-specific performance issues. Configuration testing helps ensure that the system can perform optimally across a range of hardware, operating systems, browsers, and network configurations.

=> It's important to note that these types of performance testing can be conducted individually or in combination, depending on the specific testing objectives and requirements of the system being tested. The selection of the appropriate type(s) of performance testing depends on the system's characteristics, expected usage patterns, performance goals, and the testing goals defined by the project.

**Q27. What is Error, Defect, Bug and failure?**

=> In software testing, the terms error, defect, bug, and failure are related to different aspects of software quality issues. Here's an explanation of each term:

1. Error:

An error, also known as a mistake or a fault, is a human action or a misconception that produces an incorrect or unintended result. Errors are made during the software development process and can occur at any stage, including requirements gathering, design, coding, or documentation. Errors can lead to defects in the software.

2. Defect:

A defect, also known as a fault or a bug, is a flaw or deviation in the software that causes it to behave in an unintended or incorrect way. Defects are the result of errors made during software development or maintenance. They can manifest as coding mistakes, logic errors, missing or inaccurate functionality, or any other deviation from the expected behaviour. Defects can exist in various forms, such as syntax errors, logic errors, usability issues, performance issues, or security vulnerabilities.

3. Bug:

The term "bug" is often used interchangeably with "defect." Historically, the term "bug" was first coined by Grace Hopper when she found an actual moth causing a malfunction in a computer system. Today, "bug" is a colloquial term used to refer to defects or flaws in software. In practice, there is no significant difference between a bug and a defect—they both represent flaws in the software's behaviour.

4. Failure:

A failure occurs when the software does not behave as expected or does not meet the specified requirements or user expectations. It is the manifestation of a defect during the execution of the software. Failures can result from one or multiple defects or a combination of system-related factors. Failures can include crashes, incorrect outputs, system hangs, performance issues, or any other deviation from the expected behaviour. Failures are typically observed by end-users or testers during the testing or operational phases of the software.

=> In summary, an error is a human action or misconception, a defect or bug is a flaw or deviation in the software caused by errors, and a failure is the manifestation of a defect during software execution. Defects or bugs are the root cause of failures, and testing activities aim to identify and report defects to prevent failures from occurring in the production environment.

**Q28. Difference between Priority and Severity**

|  |  |
| --- | --- |
| **Defect Severity** | **Defect Priority** |
| Severity is absolute and Customer-Focused. | Priority is Relative and Business-Focused. |
| It is the extent to which the defect can affect the software. | The priority status is set based on the customer requirements. |
| it defines the impact that a given defect has on the system. | Priority defines the order in which we should resolve a defect. |
| Severity types are critical, major, moderate, minor ets. | Priority types are low, medium, high, and critical. |

**Q29. What is Bug Life Cycle?**

=> Bug Life Cycle, also known as Defect Life Cycle, is the process that a software bug goes through from its identification to its resolution. It outlines the various stages that a bug undergoes in its lifecycle, from the moment it is discovered until it is fixed and verified. The Bug Life Cycle helps in tracking and managing the progress of bug resolution within a software development or testing process.

=> The typical stages of Bug Life Cycle are as follows:

1. New: This stage represents the initial state of a bug when it is reported or identified. At this stage, the bug is logged into a bug tracking system or an issue management tool.

2. Assigned: Once the bug is reported, it is assigned to a developer or a development team responsible for fixing it. The bug is now acknowledged and taken up for investigation.

3. Open: In this stage, the developer begins to analyse the bug, understand its cause, and reproduce it if necessary. They investigate the issue and plan for its resolution.

4. In Progress: Once the developer starts working on fixing the bug, its status is changed to "In Progress." The developer makes the necessary changes to the code or resolves the underlying issue causing the bug.

5. Fixed: When the developer completes the bug fixing process, the bug is marked as "Fixed." The necessary code changes or corrections have been made, addressing the reported issue.

6. Verified: In this stage, the fixed bug is verified by the testing team or the quality assurance team. They perform testing to ensure that the bug is indeed resolved and doesn't reoccur. If the fix is successful, the bug proceeds to the next stage.

7. Closed: Once the bug is verified and confirmed as fixed, it is marked as "Closed." The bug is considered resolved and no further action is required. It remains in the system for future reference.

8. Reopened: Sometimes, after being closed, a bug may resurface due to additional issues or not being completely resolved. In such cases, it is reopened, and the bug life cycle goes back to the "Assigned" or "Open" stage for further investigation and resolution.

=> Bug Life Cycle helps in maintaining a structured approach to bug tracking and ensures that bugs are properly addressed and resolved. It provides transparency and enables effective communication among developers, testers, and other stakeholders involved in the software development process.

**Q30.** **Explain the difference between Functional testing and Non-functional testing**

|  |  |
| --- | --- |
| **Functional Testing** | **Non-Functional Testing** |
| Functional testing is performed using the functional specification provided by the client and verifies the system against the functional requirements. | Non-Functional testing checks the Performance, reliability, scalability and other non-functional aspects of the software system. |
| Functional testing is executed first. | Non-functional testing should be performed after functional testing. |
| Manual testing or automation tools can be used for functional testing. | Using tools will be effective for this testing. |
| Business requirements are the inputs to functional testing. | Performance parameters like speed, scalability are inputs to non-functional testing. |
| Functional testing describes what the product dose. | Non-functional testing describes how good the product works. |
| Easy to do manual testing. | Tough to do manual testing. |
| Types of Functional testing are:- Unit Testing Smoke Testing Sanity Testing Integration Testing White box testing Black Box testing User Acceptance testing Regression Testing | Types of Non-functional testing are:- Performance Testing Load Testing Volume Testing Stress Testing Security Testing Installation Testing Penetration Testing Compatibility Testing Migration Testing |

**Q31. To create HLR & TestCase of**

**1) 1)(Instagram , Facebook) only first page**

[**https://github.com/TINKESH0508/TInkesh-\_Tops/blob/main/Assinment/Modual%20two/Instagram%20HLR%20And%20Tese%20case.xlsx**](https://github.com/TINKESH0508/TInkesh-_Tops/blob/main/Assinment/Modual%20two/Instagram%20HLR%20And%20Tese%20case.xlsx)

**2) Facebook Login Page :** [**https://www.facebook.com/**](https://www.facebook.com/)

[**https://github.com/TINKESH0508/TInkesh-\_Tops/blob/main/Assinment/Modual%20two/Facebook%20HLR%20And%20Test%20Case.xlsx**](https://github.com/TINKESH0508/TInkesh-_Tops/blob/main/Assinment/Modual%20two/Facebook%20HLR%20And%20Test%20Case.xlsx)

**Q32. What is the difference between the STLC (Software Testing Life Cycle) and SDLC (Software Development Life Cycle)?**

|  |  |
| --- | --- |
| **STLC** | **SDLC** |
| Stlc full form is software testing life cycle. | Sdlc full form is software development life life cycle |
| This article is about Software Testing Life Cycle (STLC), which involves a series of activities to ensure software quality goals are met and consists of six major phases: Requirement Analysis, Test Planning, Test case development, Test Environment setup, Test Execution and Test Cycle closure. | SDLC is a structure imposed on the development of a software product that defines the process for planning, implementation, testing, documentation, deployment, and ongoing maintenance and support. There are a number of different development models. |
| STLC Phases  1. Requirement Analysis  2. Test Planning  3. Test case development  4. Test Environment setup  5. Test Execution  6. Test Cycle closure | SDLC Phases  Requirements Collection/Gathering  Analysis  Design  Implementation  Testing  Maintenance |

**Q33. What is the difference between test scenarios, test cases, and test script?**

|  |  |  |
| --- | --- | --- |
| **Test scenarios** | **Test cases** | **Test script** |
| A Scenario is any functionality that can be tested. It is also called Test Condition, or Test Possibility. | Test cases involve the set of steps, conditions and inputs which can be used while performing the testing tasks. | A test script in software testing is a set of instructions that will be performed on the system under test to test that the system functions as expected. |
| Test Scenario is ‘What to be tested’ | Test Case is ‘How to be tested’ | One script is written to explain how to simulate each business scenario |
| Test scenario is nothing but test procedure. | Test case consists of set of input values, execution precondition, expected Results and executed post-condition developed to cover certain test Condition. | The Test Script can be manual or automated |
| The scenarios are derived from use cases. | Test cases are derived (or written) from test scenario. | The Test Procedures Specification specifies the sequence of actions for a test, i.e. one or more Test Cases. It is also known as a Test Script |

**Q34. Explain what Test Plan is? What is the information that should be covered?**

=> A test plan is a document that outlines the overall approach, objectives, scope, and schedule for testing a software application or system. It serves as a roadmap for the testing process and provides guidance to the testing team on how to proceed with their activities. The primary purpose of a test plan is to ensure that the software or system is thoroughly tested to meet the desired quality standards.

=> Here are some key pieces of information that should be covered in a test plan:

1. Introduction: This section provides an overview of the software or system being tested, including its purpose, objectives, and any relevant background information.

2. Test Objectives: It outlines the specific goals and objectives of the testing effort. This helps to align the testing activities with the overall project goals and ensure that the testing is focused and targeted.

3. Scope: The scope defines what aspects of the software or system will be tested and what will be excluded from the testing. It helps to set boundaries and provide a clear understanding of what is to be tested.

4. Test Strategy: The test strategy outlines the overall approach and methodologies to be employed during testing. It includes details on the types of tests to be conducted, such as functional testing, performance testing, security testing, etc., and the testing techniques to be used.

5. Test Schedule: This section provides a timeline for the testing activities, including start and end dates, milestones, and key deliverables. It helps to plan and manage the testing effort effectively.

6. Test Environment: It describes the hardware, software, and network configurations that will be used for testing. It includes details about the test environment setup, test data requirements, and any dependencies on external systems or resources.

7. Test Deliverables: It lists the documents and artifacts that will be produced during the testing process, such as test cases, test scripts, test reports, and any other relevant documentation.

8. Test Execution: This section covers the actual execution of tests, including the responsibilities of the testing team, the test schedule, and the criteria for determining when testing is complete.

9. Test Risks and Mitigation: It identifies potential risks and challenges that may affect the testing process or the quality of the software. It also outlines the measures to mitigate those risks and minimize their impact.

10. Test Sign-Off: This section specifies the criteria and process for obtaining the necessary approvals and sign-off from stakeholders once testing is completed. It ensures that the software is ready for release or further stages of development.

11. Test Resources: It outlines the resources required for testing, including personnel, tools, equipment, and training needs. It helps in identifying and allocating the necessary resources for the testing effort.

12. Change Management: This section describes how changes or defects identified during testing will be managed, including the process for reporting, tracking, and resolving issues.

=> Overall, a test plan provides a comprehensive overview of the testing process and helps ensure that testing activities are well-organized, efficient, and effective in achieving the desired quality goals.

**Q35. What is priority?**

=> Priority refers to the relative importance or urgency assigned to a task, requirement, defect, or any other item in a project. It helps determine the order in which items should be addressed, resolved, or completed based on their significance and impact on the project's goals and objectives.

=> In the context of software development and testing, priority is often used to prioritize and manage the resolution of defects or issues found during testing. When a defect is identified, it is assigned a priority level to indicate its criticality and the order in which it should be fixed.

=> Commonly used priority levels include:

1. High Priority: Issues that have a severe impact on the system's functionality, performance, or security and require immediate attention. These are critical issues that can significantly affect the user experience or prevent the software from functioning as intended.
2. Medium Priority: Issues that have a noticeable impact on the system's functionality or performance but do not require immediate attention. These issues may affect specific features or user scenarios that are important but not critical to the overall system functionality.
3. Low Priority: Issues that have a minor or negligible impact on the system's functionality or performance. These issues may include minor cosmetic defects, suggestions for improvements, or enhancements that are not critical for the software's basic functionality.

**Q36. What is severity?**

=> Severity refers to the degree of impact or seriousness of a defect or issue on the functionality, performance, or usability of a software application or system. It is a measure of how severe or critical the issue is and helps in prioritizing and allocating resources for its resolution.

=> When a defect is identified during testing or reported by users, it is assigned a severity level to indicate the impact it has on the software. The severity level is typically assigned based on the following scale or categories:

1. Critical Severity: Defects that cause complete failure or shutdown of the system, preventing it from performing its intended function. These issues result in the system being unusable or inoperable and require immediate attention.
2. High Severity: Defects that have a significant impact on the functionality or performance of the system, resulting in major errors or limitations. These issues affect critical features or workflows and may severely impact the user experience or prevent the software from meeting its intended purpose.
3. Medium Severity: Defects that have a moderate impact on the system's functionality or performance. These issues may result in errors or limitations in specific scenarios or features but do not cause complete failure or significantly hinder the system's core functionality.
4. Low Severity: Defects that have a minor or negligible impact on the system's functionality or performance. These issues may include cosmetic defects, minor inconveniences, or non-critical features that are not essential for the basic functioning of the software.

**Q37. Bug categories are…**

=> The specific bug categories may vary depending on the organization or project, but here are some common categories:

1. Functional Bugs: These bugs relate to issues in the functional behaviour of the software. They occur when the software does not perform its intended functions correctly or produces incorrect results. Examples include incorrect calculations, data validation failures, or incorrect handling of user inputs.
2. UI/UX Bugs: These bugs involve issues related to the user interface (UI) or user experience (UX) of the software. They may include problems with layout, formatting, responsiveness, usability, or inconsistencies in visual design. Examples include misaligned elements, overlapping text, non-responsive buttons, or confusing navigation.
3. Performance Bugs: Performance bugs refer to issues related to the performance of the software, such as slow response times, excessive memory usage, or high CPU utilization. These bugs can impact the efficiency and responsiveness of the system and may cause delays or unresponsiveness during normal usage.
4. Compatibility Bugs: Compatibility bugs occur when the software does not function properly or has issues when used in specific environments, configurations, or with certain hardware, software, or operating systems. These bugs may cause the software to crash, freeze, or produce incorrect results in specific setups.
5. Security Bugs: Security bugs involve vulnerabilities or weaknesses in the software that can be exploited by attackers to gain unauthorized access, manipulate data, or perform malicious actions. These bugs may include issues like cross-site scripting (XSS), SQL injection, authentication bypass, or insecure data storage.
6. Localization/Internationalization Bugs: These bugs are related to issues in adapting the software for different languages, cultures, or locales. They may involve problems with text encoding, translation errors, date/time format issues, or improper handling of international characters.
7. Documentation Bugs: Documentation bugs pertain to errors or omissions in the software's documentation or user guides. These bugs may include inaccurate instructions, out-dated information, missing explanations, or inconsistencies in the documentation.
8. Installation/Deployment Bugs: These bugs occur during the installation or deployment process of the software. They may involve issues like incomplete installations, incorrect dependencies, configuration problems, or compatibility issues with specific deployment environments.
9. Data Integrity Bugs: Data integrity bugs involve issues related to the accuracy, completeness, or consistency of data handled by the software. These bugs may result in data corruption, loss, or incorrect processing of information.

**Q38. Advantage of Bugzila.**

* Bugzilla is an open-source issue/bug tracking system that allows developers effectively to keep track of outstanding problems with their product. It is written in Perl and uses MYSQL database.
* Bugzilla is a defect tracking tool, however it can be used as a test management tool as such it can be easily linked with other test case management tools like Quality Centre, Test link etc.
* This open bug-tracker enables users to stay connected with their clients or employees, to communicate about problems effectively throughout the data management chain
* Key features of Bugzilla includes
* Advanced search capabilities
* E-mail Notifications
* Modify/file Bugs by e-mail
* Time tracking
* Strong security
* Customization

**Q39. Difference between priority and severity**

|  |  |
| --- | --- |
| **Priority** | **Severity** |
| Priority is Relative and Business-Focused. | Severity is absolute and Customer-Focused |
| Priority defines the order in which we should resolve a defect. Should we fix it now, or can it wait? This priority status is set by the tester to the developer mentioning the time frame to fix the defect. If high priority is mentioned then the developer has to fix it at the earliest. The priority status is set based on the customer requirements. | It is the extent to which the defect can affect the software. In other words it defines the impact that a given defect has on the system. |
| If the company name is misspelled in the home page of the website, then the priority is high and severity is low to fix it. | If an application or web page crashes when a remote link is clicked, in this case clicking the remote link by a user is rare but the impact of application crashing is severe. So the severity is high but priority is low. |

**Q40. What are the different Methodologies in Agile Development Model?**

=> The Agile methodology is a way to manage a project by breaking it up into several phases. It involves constant collaboration with stakeholders and continuous improvement at every stage. Once the work begins, teams cycle through a process of planning, executing, and evaluating.

=> The different Methodologies in Agile Development Model

* Scrum
* Kanban
* DSDM (Dynamic Software Development Method)
* FDD (Feature Driven Development)

**Q41. Explain the difference between Authorization and Authentication in Web testing. What are the common problems faced in Web testing?**

=> Authorization and authentication are two essential concepts in web testing that relate to user access and security. Here's an explanation of each term and the differences between them:

1. Authentication: Authentication is the process of verifying the identity of a user or system before granting access to specific resources or functionalities. It ensures that the user is who they claim to be. Common authentication mechanisms include usernames and passwords, biometric authentication (such as fingerprint or facial recognition), tokens, certificates, and single sign-on (SSO) solutions. In web testing, the focus is on verifying that the authentication process works correctly and securely, preventing unauthorized access.

2. Authorization: Authorization refers to the process of granting or denying access to specific resources or functionalities based on the authenticated user's permissions and privileges. It determines what actions a user can perform and what data they can access. Authorization is usually based on roles, permissions, and access control lists (ACLs). In web testing, the goal is to ensure that the authorization rules are correctly implemented, and users can only access the resources they are authorized to access.

=> To summarize, authentication is about verifying the user's identity, while authorization is about granting or denying access based on that identity.

=> Common problems faced in web testing include:

1. Broken Links: Links within a web application can become broken or dead over time, leading to a poor user experience. Web testing involves checking for broken links and ensuring that all links navigate correctly.

2. Cross-Browser Compatibility: Web applications need to be tested across different browsers and their versions to ensure consistent functionality and appearance. Incompatibilities may arise due to variations in rendering engines or browser-specific features.

3. Performance Issues: Web applications need to handle a large number of concurrent users efficiently. Testing for performance issues such as slow response times, bottlenecks, and scalability problems is crucial to ensure optimal user experience.

4. Security Vulnerabilities: Web applications are often targeted by malicious actors. Web testing involves identifying and addressing common security vulnerabilities, such as cross-site scripting (XSS), cross-site request forgery (CSRF), SQL injection, and insecure session management.

5. Usability Testing: Usability testing focuses on evaluating the user-friendliness of a web application. It involves assessing the application's intuitiveness, ease of navigation, clarity of instructions, and overall user satisfaction.

6. Data Validation: Input validation is critical to prevent security vulnerabilities and data integrity issues. Web testing involves checking for proper validation of user input to prevent malicious input or unintended errors.

7. Compatibility with Different Devices: With the wide variety of devices used to access the web, including smartphones, tablets, and desktops, web testing should include testing on various screen sizes, resolutions, and operating systems to ensure compatibility and responsiveness.

=> These are just a few common problems in web testing, and the actual challenges can vary depending on the specific web application and its requirements.

**Q42 To create HLR & TestCase of WebBased (WhatsApp web , Instagram)**

**1. WhatsApp Web :** [**https://web.whatsapp.com/**](https://web.whatsapp.com/)

[**https://github.com/TINKESH0508/TInkesh-\_Tops/blob/main/Assinment/Modual%20two/web%20whatsapp%20HLR%20And%20Test%20Case.xlsx**](https://github.com/TINKESH0508/TInkesh-_Tops/blob/main/Assinment/Modual%20two/web%20whatsapp%20HLR%20And%20Test%20Case.xlsx)

**2. Instagram web :**

[**https://github.com/TINKESH0508/TInkesh-\_Tops/blob/main/Assinment/Modual%20two/Instagram%20HLR%20And%20Tese%20case.xlsx**](https://github.com/TINKESH0508/TInkesh-_Tops/blob/main/Assinment/Modual%20two/Instagram%20HLR%20And%20Tese%20case.xlsx)

**Q43.  To create HLR and TestCase on this Link.** [**https://artoftesting.com/**](https://artoftesting.com/)

[**https://github.com/TINKESH0508/TInkesh-\_Tops/blob/main/Assinment/Modual%20two/artoftesting%20HLR%20And%20Test%20Case.xlsx**](https://github.com/TINKESH0508/TInkesh-_Tops/blob/main/Assinment/Modual%20two/artoftesting%20HLR%20And%20Test%20Case.xlsx)

**Q44. Write a scenario of only Whatsapp chat messages**

[**https://github.com/TINKESH0508/TInkesh-\_Tops/blob/main/Assinment/Modual%20two/cenario%20of%20only%20Whatsapp%20chat%20messages.xlsx**](https://github.com/TINKESH0508/TInkesh-_Tops/blob/main/Assinment/Modual%20two/cenario%20of%20only%20Whatsapp%20chat%20messages.xlsx)

**Q45. Write a Scenario of Pen.**

[**https://github.com/TINKESH0508/TInkesh-\_Tops/blob/main/Assinment/Modual%20two/Scenario%20of%20Pen.xlsx**](https://github.com/TINKESH0508/TInkesh-_Tops/blob/main/Assinment/Modual%20two/Scenario%20of%20Pen.xlsx)

**Q46. Write a Scenario of Pen Stand.**

[**https://github.com/TINKESH0508/TInkesh-\_Tops/blob/main/Assinment/Modual%20two/Scenario%20of%20Pen%20Stand.xlsx**](https://github.com/TINKESH0508/TInkesh-_Tops/blob/main/Assinment/Modual%20two/Scenario%20of%20Pen%20Stand.xlsx)

**Q47. Write a Scenario of Door.**

[**https://github.com/TINKESH0508/TInkesh-\_Tops/blob/main/Assinment/Modual%20two/Scenario%20of%20Door.xlsx**](https://github.com/TINKESH0508/TInkesh-_Tops/blob/main/Assinment/Modual%20two/Scenario%20of%20Door.xlsx)

**Q48. Write a Scenario of ATM.**

[**https://github.com/TINKESH0508/TInkesh-\_Tops/blob/main/Assinment/Modual%20two/Scenarion%20Of%20atm.xlsx**](https://github.com/TINKESH0508/TInkesh-_Tops/blob/main/Assinment/Modual%20two/Scenarion%20Of%20atm.xlsx)

**Q49. When to used Usability Testing?**

=> Usability testing is typically used during the design and development process of a product or service to evaluate its usability. It involves observing and gathering feedback from users as they interact with a prototype or a finished product to identify usability issues and make informed design decisions.

=> Here are some specific scenarios when usability testing is commonly used:

1. Early design stages: Usability testing can be conducted during the initial stages of the design process to gather user feedback on concept ideas, wireframes, or low-fidelity prototypes. This helps in validating design assumptions, identifying potential usability problems, and informing iterative design improvements.

2. Prototype evaluation: Usability testing is useful when testing interactive prototypes that simulate the functionality of a product before its full development. It allows designers and developers to observe how users navigate through the prototype, interact with its features, and gather insights on user expectations and preferences.

3. Iterative design: Usability testing is often conducted throughout the iterative design process to validate design changes and improvements. By testing and collecting feedback on successive versions of a product, designers can refine and optimize the user experience based on user insights and preferences.

4. Comparative evaluation: Usability testing can be used to compare different design options or variants. By testing multiple designs with users, it helps in identifying the strengths and weaknesses of each design option, making informed decisions about which design direction to pursue.

5. New feature assessment: When introducing new features or significant updates to an existing product, usability testing can gauge how well users understand and utilize the new functionality. This ensures that the new features align with user needs and expectations and do not introduce any usability problems.

6. Post-launch evaluation: Usability testing can be conducted after the product is launched to gather user feedback, identify areas of improvement, and prioritize future updates or enhancements. This helps in continuously enhancing the user experience and maintaining a user-centric approach.

=> Overall, usability testing is valuable whenever you want to evaluate the usability of a product, gather user feedback, and ensure that the design meets the needs and expectations of its intended users.

**Q50. What is the procedure for GUI Testing?**

=> GUI testing, also known as Graphical User Interface testing, focuses on evaluating the usability and functionality of the graphical elements of a software application. Here is a general procedure for conducting GUI testing:

1. Test Planning: Define the objectives, scope, and test strategy for GUI testing. Identify the target audience and their requirements. Determine the platforms, browsers, or devices on which the application needs to be tested.

2. Identify Test Cases: Identify the test scenarios and test cases specific to GUI testing. These test cases should cover various aspects such as layout, navigation, input fields, buttons, menus, forms, error handling, and visual elements.

3. Test Environment Setup: Set up the necessary test environment, including the hardware, software, operating systems, and other dependencies required for GUI testing.

4. Functional Testing: Conduct functional testing to ensure that the GUI elements are working as intended. Test the functionality of buttons, links, menus, forms, and other interactive elements. Verify that the application responds correctly to user actions and inputs.

5. Layout and Design Testing: Evaluate the layout and design of the graphical elements. Test for consistency in color schemes, fonts, alignment, spacing, and overall visual appeal. Verify that the GUI follows established design guidelines and standards.

6. Usability Testing: Assess the usability of the GUI by simulating real user interactions. Test the application's ease of use, intuitiveness, and user-friendliness. Identify any usability issues such as confusing navigation, unclear labels, or inefficient workflows.

7. Error Handling: Test how the application handles errors and displays error messages. Verify that error messages are clear, meaningful, and provide guidance to the user on how to resolve the issue.

8. Compatibility Testing: Ensure that the GUI is compatible with different browsers, screen resolutions, and devices. Test the application on various platforms and configurations to identify any rendering or functionality issues.

9. Accessibility Testing: Evaluate the GUI for accessibility compliance. Test for compatibility with assistive technologies, keyboard navigation, alternative text for images, proper color contrast, and other accessibility guidelines.

10. Localization Testing: If the application supports multiple languages or regions, conduct localization testing to verify that the GUI elements are correctly translated and culturally appropriate for each target audience.

11. Performance Testing: Assess the GUI's performance by testing its response time, loading speed, and overall responsiveness. Check for any delays or lags in the graphical rendering or user interactions.

12. Regression Testing: Perform regression testing to ensure that any changes or fixes made to the GUI do not introduce new issues or break existing functionality.

13. Documentation: Document the test results, including any issues, bugs, or suggestions for improvement. Create a comprehensive test report that summarizes the GUI testing process and outcomes.

=> Remember that the specific steps and techniques may vary depending on the project and the tools used for GUI testing. It's essential to adapt the procedure to suit the specific needs of your application and project requirements.

**Q51. Write a scenario of Microwave Owen.**

[**https://github.com/TINKESH0508/TInkesh-\_Tops/blob/main/Assinment/Modual%20two/Scenario%20of%20maicrowev%20owen.xlsx**](https://github.com/TINKESH0508/TInkesh-_Tops/blob/main/Assinment/Modual%20two/Scenario%20of%20maicrowev%20owen.xlsx)

**Q52. Write a scenario of Coffee vending Machine.**

[**https://github.com/TINKESH0508/TInkesh-\_Tops/blob/main/Assinment/Modual%20two/scenario%20of%20Coffee%20vending%20Machine.xlsx**](https://github.com/TINKESH0508/TInkesh-_Tops/blob/main/Assinment/Modual%20two/scenario%20of%20Coffee%20vending%20Machine.xlsx)

**Q53. Write a scenario of chair.**

[**https://github.com/TINKESH0508/TInkesh-\_Tops/blob/main/Assinment/Modual%20two/scenario%20of%20chair.xlsx**](https://github.com/TINKESH0508/TInkesh-_Tops/blob/main/Assinment/Modual%20two/scenario%20of%20chair.xlsx)