TEST PLAN: SPRINT GRID APP

ChangeLog

Version	**Change Date**	**By**	**Description**
version number	Date of Change	Name of the perso who made changes	Description of the changes made
v1.0.0	29-01-2023	Sneha K P	Added detailed test plan for Sprint grid app

01. INTRODUCTION

The "Sprint Grid" is an application that enables users to create tasks and assign them built-in statuses as "To Do", "In Progress", "In Testing", "Blocked", and "Done" for specific dates.

02. SCOPE & OBJECTIVES

SCOPE

The scope of the project is to write a test plan and automate end-to-end tests for the "sprint-grid" frontend application. The tests cover all the features of the application including creating tasks, assigning built-in statuses, creating new date columns, updating statuses, removing rows and columns, and validating input fields.

OBJECTIVES

- To write a **test plan** for the "sprint-grid" application based on the provided requirements.
- To **automate end-to-end tests** for the "sprint-grid" app using a JavaScript framework.
- To discover any **bugs** in the application and ensure that the appropriate tests fail
- To provide a **report** of any failed manual tests that were discovered during testing
- To ensure that the tests are able to **cover all the features** of the application and validate input fields as per the requirements.

03. TESTING METHODOLOGY

- **Functional Testing**: This type of testing is used to ensure that the application functions as expected and meets the requirements outlined in the test plan. This can include testing the ability to create tasks, assign statuses, and remove rows and columns.

- **Unit Testing**: This type of testing is used to test individual units or components of the application, such as individual functions or methods. This can include testing the validation of input fields, the functionality of the date picker or keyboard, and the functionality of the status field.

- **Integration Testing**: This type of testing is used to test how different components of the application work together. This can include testing the integration of the front and backend or testing the integration of different modules or features.

- **End-to-End Testing**: This type of testing is used to test the application as a whole, from start to finish. This can include testing the complete user flow, from creating a task to removing a column, and ensuring that all components of the application work together seamlessly.

- **Acceptance Testing**: This type of testing is used to ensure that the application meets the needs and expectations of the end-users. This can include testing the usability, performance, and overall user experience.

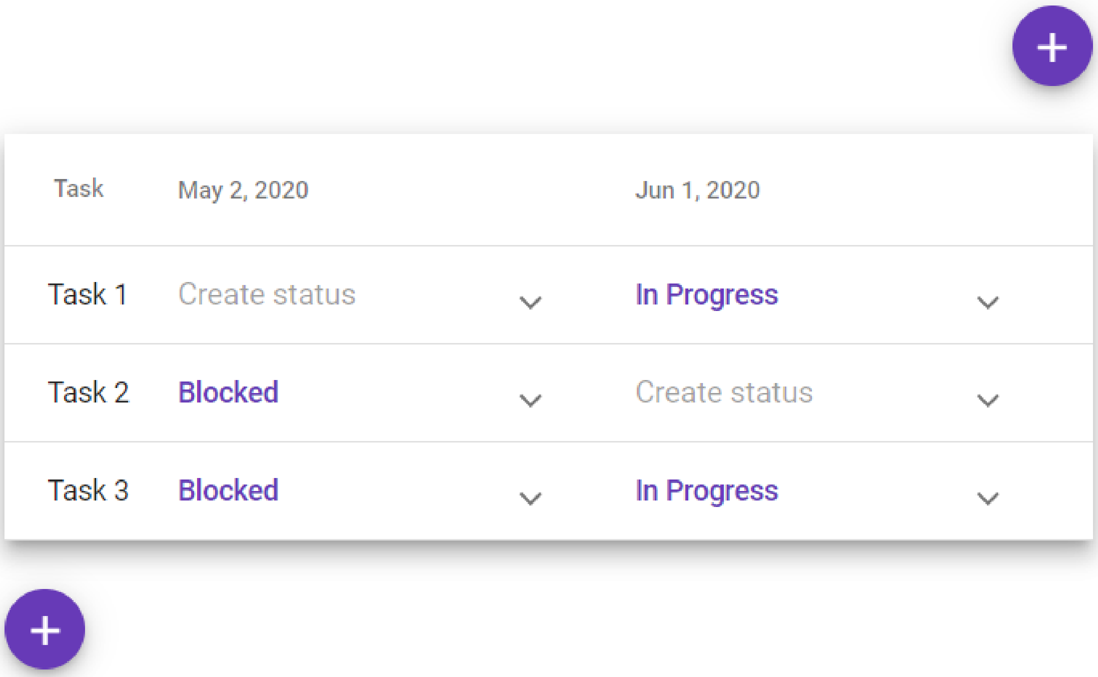
- **Performance Testing**: This type of testing is used to evaluate the performance of the application under different loads and conditions. This can include testing the response time, throughput, and scalability of the application.

- **Security Testing**: This type of testing is used to evaluate the security of the application. This can include testing the application against known vulnerabilities, testing the encryption of sensitive data, and testing the application's ability to handle malicious attacks.

04. TEST APPROACH

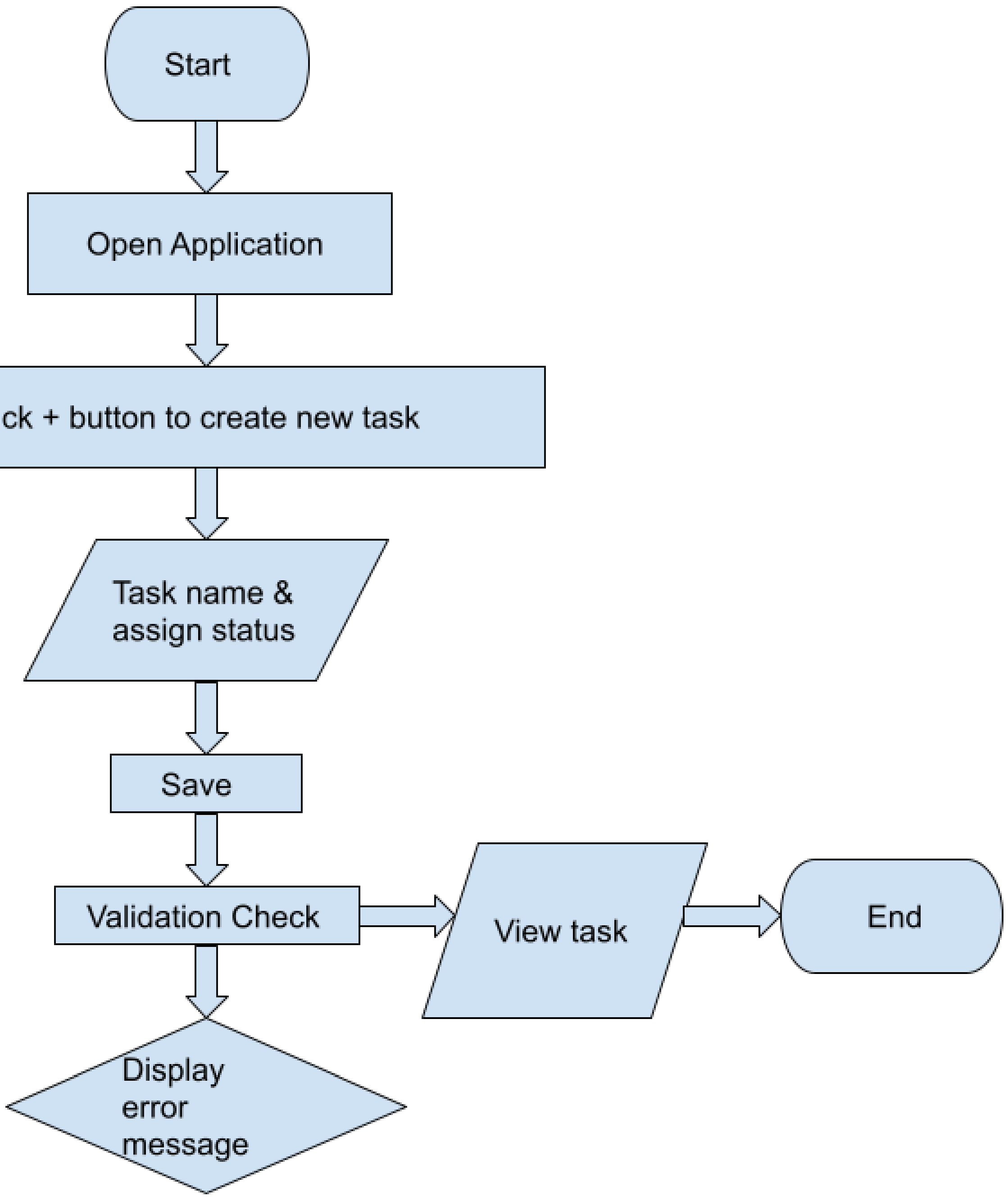
04.01. FLOW CHART

APPLICATION



Task	May 2, 2020	Jun 1, 2020
Task 1	Create status	In Progress
Task 2	Blocked	Create status
Task 3	Blocked	In Progress

FLOW CHART


Start
Open Application
Click + button to create new task
Task name & assign status
Save
Validation Check
View task
End
Display error message

04.02. TEST SCENARIOS

Verify the below scenarios:

- New task creation
- New date column creation
- Assign status to a particular task
- Remove a created task
- Remove date creation
- Errors

- Validate Task name field
- Validate Date field
- Validate Status field

04.03 TEST CASES

Test cases are attached in another document along with RTM.

05. ASSUMPTIONS

- The "sprint-grid" application is a web-based application(local) that can be accessed through a web browser.

- The application is built using JavaScript, and the front end is developed using a framework such as React or Angular.

- The application has a back-end server that handles the creation, update, and deletion of tasks and statuses.

- The application supports multiple browsers and the latest browser versions.

- The application supports multiple screen sizes and resolutions.

- The application is accessibility compliant.

- The application runs on a stable internet connection.

06. FUNCTIONALITY

06.01 Risks Identified & Mitigation Plan

- **Inability to create new tasks**: This risk can be mitigated by thoroughly testing the functionality that allows users to create new tasks, and by ensuring that the application has proper input validation and error handling in place.

- **Inability to update task statuses**: This risk can be mitigated by thoroughly testing the functionality that allows users to update task statuses, and by ensuring that the application has proper input validation and error handling in place.

- **Incorrect display of task statuses**: This risk can be mitigated by thoroughly testing the functionality that displays task statuses, and by ensuring that the application has proper input validation and error handling in place.

- **Inability to delete tasks**: This risk can be mitigated by thoroughly testing the functionality that allows users to delete tasks, and by ensuring that the application has proper input validation and error handling in place.

- **Inability to filter or search tasks**: This risk can be mitigated by thoroughly testing the functionality that allows users to filter or search tasks, and by ensuring that the application has proper input validation and error handling in place.

06.02 TEST STRATEGY

The functional test strategy is to ensure testing the ability to create tasks, update task statuses, create date columns, and ensure that duplicate dates cannot be created. Additionally, testing the ability to delete tasks and columns, and the ability to move tasks between columns. The test cases should cover both positive and negative scenarios and cover all the possible edge cases.

06.03 TEST AUTOMATION PLAN

- **Identifying the test cases that are suitable for automation**: This involves reviewing the functional requirements and use cases to identify the test scenarios that can be automated.

- **Selecting the automation tools**: Based on the type of application, the testing goals and the resources available, the appropriate automation tools need to be selected. For example,**Cypress** can be used for UI Testing using **Java script**, while Appium could be used for automating mobile app testing.

- **Building the test automation framework**: Once the automation tools have been selected, the next step is to build a test automation framework that provides a structure for creating, executing and maintaining test scripts. **Eg. Jest or Mocha with Cypress.**

- **Designing and implementing the test scripts**: Based on the test cases identified earlier, the test scripts need to be designed and implemented using the automation tools and framework.

- **Executing the test scripts**: The test scripts need to be executed on the target application to check for any defects.

- **Reporting and analyzing the test results**: The test results need to be analyzed and reported to the relevant stakeholders, including the development team and the business stakeholders.

- **Maintenance and updating the test scripts**: As the application changes over time, the test scripts need to be updated and maintained to ensure that they continue to be relevant and effective.

06.04 DELIVERABLES

- **Test Plan**: A document that outlines the testing approach, objectives, test environment, and resources needed for the functional testing process.

- **Test Cases**: A set of detailed and structured test cases that cover all the functional requirements of the system.

- **Test Scripts**: Automated or manual scripts that can be used to execute the test cases.

- **Test Data**: A set of data that will be used to test the system's functionality.

- **Test Results**: A report that documents the outcome of the testing process, including any defects or issues encountered.

- **Defect Reports**: A document that describes any defects or issues found during testing, along with their severity and priority.

- **Test Closure Report**: A document that summarizes the testing process, including the overall test results, defects found, and any recommendations for improvement.

- **Test Metrics**: A set of metrics that measure the effectiveness and efficiency of the testing process, such as test coverage, defects found, and test execution time.

07. SECURITY

07.01.01 RISK IDENTIFIED

- **SQL injection attacks**, where a malicious user could input code into a form field that would allow them to access or manipulate the database.

- **Cross-site scripting (XSS) attacks**, where a malicious user could input code into a form field that would execute on the user's browser, potentially allowing them to steal user data or control the user's browser.

- **Cross-site request forgery (CSRF) attacks**, where a malicious user could trick the user into making a request to the application that would allow them to perform actions on the user's behalf.

- **Inadequate user authentication and access controls**, where a malicious user could gain unauthorized access to sensitive data.

07.01.02 MITIGATION PLANNED

- Using prepared statements or **parameterized queries** to prevent SQL injection attacks.

- Implementing input **validation and sanitization** to prevent XSS attacks.

- Using **CSRF** tokens to prevent CSRF attacks.

- Implementing **strong user authentication** and access controls, such as multi-factor authentication and role-based access controls.

- Regularly **monitoring** and testing the system for security vulnerabilities.

- Keeping all software up to date with the latest **security patches.**

- Regularly **auditing and testing** the security of the system.

- Conducting **penetration** testing to identify and fix vulnerabilities.

- Train the team and developers on security **best practices.**

- Have a **clear incident management** and incident response plan in place.

07.02 AUTOMATION

The automation plan for security testing would involve the use of specialized security testing tools that can automate the process of identifying and mitigating security risks.

These tools can be used to perform a variety of security tests, such as vulnerability scanning, penetration testing, and security compliance testing.

The first step in the automation plan would be to identify the **specific security risks** that are relevant to the use case, such as those related to data confidentiality, integrity, and availability. Once the risks have been identified, the next step would be to select and configure the appropriate security testing tools to perform automated tests against these risks.
The automation plan would also involve **setting up test environments** and creating test scripts that can be run against the security testing tools. The test scripts should be designed to simulate real-world scenarios that are relevant to the use case, such as user interactions, data flows, and network interactions.

After the automated security tests have been completed, the results would be analyzed and any **vulnerabilities or security issues** that are identified would be reported and tracked. The next step would be to implement appropriate mitigation measures to address the identified issues. The automation plan should also include a process for regular testing and monitoring to ensure that the security controls remain effective over time.

The deliverables for security testing would include a comprehensive report that provides an overview of the security risks identified and the mitigation measures implemented, as well as detailed **test results and documentation** of any issues or vulnerabilities that were identified.

07.03 TEST STRATEGY AND DELIVERABLES

The test strategy for security testing would involve identifying potential security vulnerabilities and risks associated with the system and its functionality. This would involve conducting a thorough analysis of the system architecture, data flow, and access controls to **identify any potential vulnerabilities.**

Once potential vulnerabilities have been identified, a comprehensive set of tests should be developed to verify that the system is secure and that any identified vulnerabilities have been addressed. This may include testing for common vulnerabilities such as SQL injection, cross-site scripting, and cross-site request forgery, as well as testing for more advanced threats such as **denial of service attacks and privilege escalation.**

It is also important to consider the overall security of the system, including the security of the data stored within it, the **security of communications** between different components of the system, and the security of the system's interfaces and APIs.

A key part of the security testing strategy would be to use automated testing tools to scan the code and infrastructure for known vulnerabilities, and to perform penetration testing to simulate real-world attack scenarios.
Additionally, consider conducting regular security audits, threat modeling, and risk assessments to identify any **new security risks** as the system evolves over time.

The deliverables for security testing would include a detailed report of the vulnerabilities identified and the steps taken to mitigate them, along with any recommendations for further improvements to the system's security. It's also important to keep track of any new vulnerabilities discovered in the future and proactively address them.

08. PERFORMANCE

08.01 RISK IDENTIFIED & MITIGATION PLANS

Performance risks may include issues such as slow load times, difficulty handling a high volume of requests, and difficulty scaling the system to handle increased usage. Some potential mitigation plans for these risks could include:

- **Optimizing** the system's code and database queries to improve performance
- Utilizing caching mechanisms to **reduce the number of requests** to the server
- Implementing **load testing** to identify and address bottlenecks in the system
- Utilizing **cloud-based infrastructure** or autoscaling to handle increased usage
- Optimizing the system to handle a high number of **concurrent users**
- Implementing monitoring and **alerting** to quickly identify and address any performance issues as they arise

08.02. TEST STRATEGY

- **Define the performance requirements**: Identify the performance criteria that are important for the system, such as response time, throughput, and scalability.

- **Identify the test environment:** Identify the hardware and software configurations that will be used for testing.

- **Identify the test data:** Identify the data that will be used for testing, such as the number of users and transactions.

- **Identify the test cases:** Identify the test cases that will be used to test the system's performance, such as load testing, stress testing, and endurance testing.

- **Identify the tools:** Identify the tools that will be used for performance testing, such as **Apache JMeter, LoadRunner, and Gatling.**

- **Execute the tests:** Execute the performance tests and collect the data.

- **Analyze the results:** Analyze the results of the tests and identify any performance bottlenecks.

- **Report the findings:** Prepare a report that summarizes the findings of the performance testing and includes recommendations for improving the system's performance.

- **Monitor the performance:** Continuously monitor the system's performance in the production environment and make necessary adjustments.

08.03 AUTOMATION

- Identifying the performance testing objectives and **key performance indicators** (KPIs) for the system.

- Identifying and selecting the appropriate performance testing tools and **frameworks** to be used for the automation.

- Developing the **test scripts and test cases** that will be used to simulate different load and usage scenarios.

- Configuring and setting up the **test environment,** including any necessary test data and test users.

- Executing the **performance tests** in the test environment and analyzing the results to identify any performance bottlenecks or issues.

- **Iterating** on the test scripts and test cases as necessary to improve the performance of the system.

- **Documenting the results** of the performance tests and any recommendations for performance improvements.

- **Continuously monitoring** the system's performance in production and comparing it with the results from the performance tests to ensure that the system is meeting the performance requirements.

08.04 DELIVERABLES

- **Performance test plan**: This document outlines the approach, test cases, and test data to be used in performance testing, as well as any assumptions and constraints.

- **Performance test results**: This document contains the results of the performance tests, including any metrics collected, such as response times, throughput, and resource utilization.

- **Performance test report**: This document summarizes the performance test results and provides recommendations for any identified performance issues.

- **Performance test scripts**: These scripts, which may be automated, are used to run the performance tests and collect the necessary data.

- **Performance test data**: This data, which may be synthetic or real-world, is used to simulate the expected load on the system.

- **Performance test environment**: This includes the hardware and software set up used for performance testing, including any load generators or monitoring tools used.

- **Performance test metrics:** This include the metrics used to measure the performance of the system under test, such as response time, throughput, and resource utilization.

- **Performance test artifacts:** This include any additional documents or files related to the performance testing process, such as screenshots, log files, and configuration files.

09. TEST TEAM ORGANIZATION

- **Test Lead**: The test lead would be responsible for overseeing the entire testing process and ensuring that it is executed effectively and efficiently. They would also be responsible for coordinating with other teams and stakeholders to ensure that testing is aligned with project goals and timelines.

- **Test Analyst:** Test analysts would be responsible for creating and executing test cases, as well as identifying and reporting any defects or issues that are discovered during testing.

- **Test Automation Engineer**: The test automation engineer would be responsible for designing and implementing automated test scripts to support the testing process.

- **Performance Tester:** The performance tester would be responsible for designing and executing performance tests to ensure that the system can handle the expected load and perform well under stress.

- **Security Tester:** The security tester would be responsible for identifying and testing for any security vulnerabilities in the system and ensuring that the system is protected against potential threats.

- **Test Manager:** The test manager would be responsible for managing the test team, ensuring that testing is completed on time and within budget, and reporting on testing progress to stakeholders.

10. TEST SCHEDULE

- **Planning**: This phase would involve finalizing the test plan, test cases and test scenarios, and test data. It would also involve identifying the testing tools and environments required.

- **Design**: This phase would involve creating test cases and test scenarios, and documenting the test approach, test environment, and test data.

- **Execution**: This phase would involve executing the test cases, reporting defects, and tracking their resolution.

- **Closure**: This phase would involve documenting the testing results and metrics, and conducting a lessons-learned session.

- **Preparation**:
  - Define the scope and objectives of the project
  - Identify the tasks that need to be completed
  - Identify the testing methodologies to be used
  - Identify the testing tools and resources required
  - Identify the test environment
  - Identify the test data
  - Identify the test team members and their roles
  - Identify the entry and exit criteria
- **Planning**:
  - Create a test plan
  - Define the test levels (unit, integration, system, acceptance)
  - Define the test types (functional, non-functional)
  - Define the test techniques (black box, white box, grey box)
  - Define the test schedule
  - Define the test deliverables
  - Define the test risks and mitigation plans
- **Design**:
  - Create test cases and test scenarios
  - Create test scripts
  - Create test data
  - Create test environment

    - Create test harnesses and stubs
- **Execution**:
    - Execute the tests
    - Record the test results
    - Report defects
    - Retest the defects
    - Perform regression testing
- **Evaluation**:
    - Evaluate the test results
    - Evaluate the test coverage
    - Evaluate the test effectiveness
    - Evaluate the test efficiency
- **Closure**:
    - Close the test activities
    - Archive the test deliverables
    - Review the lessons learned
    - Update the test documentation
    - Communicate the test results to the stakeholders

- **Define the scope and objectives of the project:** Clearly define the goals and objectives of the project, including the specific features and functionality that will be tested.

- **Identify the testing methodology:** Determine the testing methodology that will be used for the project, such as Agile, Waterfall, or a hybrid approach.

- **Create a test plan:** Develop a detailed test plan that outlines the steps, resources, and timelines required to successfully test the project.

- **Develop test scenarios:** Create a set of test scenarios that cover all the functional and non-functional requirements of the project.

- **Identify functional risks and mitigation plans:** Identify any potential functional risks that could impact the success of the project and develop mitigation plans to address them.

- **Develop a functional test strategy:** Create a functional test strategy that outlines the approach and tools that will be used to test the functionality of the project.

- **Develop an automation plan:** Determine the automation tools and frameworks that will be used to automate the functional testing process.

- **Identify security risks and mitigation plans:** Identify any potential security risks that could impact the project and develop mitigation plans to address them.

- **Develop a security test strategy:** Create a security test strategy that outlines the approach and tools that will be used to test the security of the project.

- **Develop a performance test strategy:** Create a performance test strategy that outlines the approach and tools that will be used to test the performance of the project.

- **Develop a test team organization:** Define the roles and responsibilities of the test team members, including the lead tester, test engineers, and automation developers.

- **Develop a test schedule:** Create a detailed schedule that outlines the start and end dates of the different testing phases and milestones.

- **Monitor and track progress:** Monitor the progress of the testing process and track any issues or risks that arise.

- **Prepare deliverables:** Prepare all necessary deliverables such as test cases, test reports, and test results.

- **Conduct a final review:** Conduct a final review of the project before release to ensure that all testing has been completed and that the project meets all the requirements.

11. DEFECT CLASSIFICATION MECHANISM

11.01 DEFECT LOGGING & STATUS CHANGING MECHANISM

- **Identification**: The first step is to identify the defects during the testing process. This can be done by the testing team who will be conducting functional, security, and performance testing.

- **Logging**: The next step is to log the defects found into a defect tracking system. This system will store all the details of the defects such as the type of defect, the module in which the defect was found, the severity of the defect, and a description of the defect.

- **Assigning**: Once the defects are logged, they need to be assigned to the appropriate team member or developer for further analysis and resolution.
- Investigation: The developer or team member assigned to the defect will investigate the issue and try to replicate the problem. They will also analyze the cause of the problem and come up with a solution.

- **Resolution**: Once the problem is understood, the developer or team member will implement the solution and verify that the problem is resolved.

- **Testing**: After the resolution of the problem, the testing team will conduct regression testing to ensure that the fix does not break any other functionality.

- **Closing**: After the testing team confirms that the issue is resolved, the defect is closed and the status is updated in the defect tracking system.

- **Reporting**: A report will be generated from the defect tracking system on a regular basis, which will be reviewed by the project manager and the testing team lead. This report will include details on the number of defects found, the number of defects resolved, the number of defects pending, and the status of each defect.

11.02 TURN AROUND TIME FOR DEFECT FIXES

The turn around time for defect fixes will depend on the severity of the defect and the complexity of the fix. In general, the process for addressing defects would involve the following steps:

- Defects are **logged** by the testing team, along with detailed information about the steps to reproduce the issue and any relevant error messages or logs.

- The development team **triages the defects**, prioritizing them based on the severity and impact to the system.

- The development team works to **fix the defects,** with the goal of delivering a fix as quickly as possible.

- Once the fix has been implemented, the testing team **verifies** that the defect has been resolved and that the fix does not introduce any new issues.

- If the defect is deemed to be resolved, the status of the defect is changed to "**closed**".

12. CONFIGURATION MANAGEMENT

Configuration management would involve maintaining and controlling the changes made to the system's hardware, software, and documentation throughout the testing process. This includes maintaining a record of all the versions of the system's components, tracking changes made to the system, and ensuring that the system is in a known and stable state before testing begins.

- **Identify and document the configuration items:** This includes identifying all the hardware and software components that make up the system and documenting their versions and configurations.

- **Develop a configuration management plan:** This plan should include procedures for controlling changes to the system, as well as procedures for identifying, reporting, and resolving configuration management issues.

- **Implement the configuration management plan:** The plan should be put into action, and all changes made to the system should be tracked and controlled.

- **Perform regular configuration audits:** Audits should be conducted at regular intervals to ensure that the system is in the desired configuration and that the configuration management plan is being followed correctly.

- **Maintain configuration management records:** Records should be maintained of all the changes made to the system and all configuration management activities, including audits and issues that were identified and resolved.

- **Review and update the configuration management plan:** The configuration management plan should be reviewed and updated as needed to ensure that it remains relevant and effective.

13. RELEASE MANAGEMENT

- Identifying the **release version** for the use case.
- Reviewing and **consolidating** all the test cases, test results, and defects for the use case.
- Preparing the **release notes** for the use case, which includes the features that have been added, modified, or removed.
- **Reviewing** the product documentation and ensuring that it is up-to-date and accurate.
- Preparing the **test environment** for the release.
- Preparing and **executing** the test plan for the release.
- Identifying and **resolving** any issues found during testing.
- **Sign-off** on the release by the relevant stakeholders.
- **Deployment** of the release to the production environment.
- **Conducting post-release** testing to ensure that the release is stable and working as expected.
- **Communicating the release** to the end-users and providing support for any issues that may arise.
- **Continuously monitoring** the release to ensure that it is stable and addressing any issues that may arise.