Test Plan

# **Introduction**

## Test Plan Objectives

### **Objective:**

**The objective of the test plan is to ensure the quality and reliability of the local delivery system. The testing aims to validate the system's functionality, assess performance, verify capacity limitations, test system stability, evaluate accuracy in decision-making, and ensure user satisfaction.**

### **Expected Results:**

**The expected results include a validated and reliable system that optimizes package allocation, stable performance and provides a satisfactory user experience. Any identified issues or defects will be addressed to enhance the system's quality and functionality.**

# **Scope**

## What will be tested

The test plan encompasses the following areas for testing:

1. Functionality: Thorough testing will validate the system's functionality. The objective is to ensure the accurate assignment of packages to trucks based on available space and proximity to the destination. Emphasis will be placed on testing the system's ability to calculate the shortest path, considering obstacles such as buildings.
2. Performance: Performance testing will evaluate the system's responsiveness and efficiency. Various workloads and scenarios will be simulated to measure the system's capability in processing incoming shipments and allocating them to trucks within acceptable timeframes.
3. Capacity: The capacity of individual trucks and the system will be rigorously tested. The testing will assess the system's capability to handle diverse package sizes and weights while adhering to the predefined weight and volume limitations.
4. System Stability: The system's stability and robustness will be examined to ensure its resilience in dealing with unexpected situations and errors. Comprehensive testing will evaluate error-handling mechanisms, exception-handling processes, and the system's ability to recover from failures or disruptions.
5. Accuracy: Testing will be carried out to verify the accuracy of the system's decision-making processes. Specifically, the focus will be on ensuring the system can accurately determine the closest truck to the destination by considering the shortest path and available space.
6. User Satisfaction: User interface testing will be employed to assess the system's usability, visual appeal, and ease of use. Feedback from users will be gathered to identify areas for improvement and enhance overall user satisfaction.

## What will NOT be tested

Certain aspects fall outside the scope of this test plan and will not be explicitly tested:

1. Hardware: Testing of the physical hardware components, such as the trucks and their specific capabilities, is beyond the scope of this test plan.
2. Network Infrastructure: The test plan does not encompass testing the underlying network infrastructure the delivery system utilizes, as the focus remains on software functionality and performance.
3. External Dependencies: While integration with third-party APIs or services may be evaluated within the scope of functionality and performance testing, detailed testing of these external dependencies is not explicitly covered.
4. Security and Penetration Testing: The test plan does not include comprehensive security testing, such as penetration testing or vulnerability assessments. However, security considerations will be considered during the testing process.
5. Compatibility: The test plan does not explicitly cover compatibility testing with specific hardware or software configurations. However, primary compatibility considerations will be addressed within the functionality and performance testing scope.

It is essential to acknowledge that even though certain aspects are not explicitly included in the test plan, they may still impact the overall performance and functionality of the system. Therefore, their potential influence should be noticed during the evaluation process.

# **Test Strategy**

## **Test Execution and Methodology**

**This section will outline the methodology for performing the tests, including the different types of tests that will be conducted. While not all projects will require each type of test listed below, this provides a comprehensive overview of the possible testing approaches. Additionally, we will discuss the source of test data, the levels of testing involved, test deliverables, responsible roles, and an estimated timeframe for completing the testing process.**

1. **System Test: The system test will be conducted to validate the local delivery system's overall functionality, including verifying package assignment, route optimization, and accurate calculation of the shortest path. Test data will be generated to simulate various scenarios and edge cases to ensure robustness. The QA team will be responsible for this test.**
2. **Performance Test: Performance testing evaluates the system's responsiveness and efficiency under different workloads. This test will simulate high volumes of incoming shipments to measure the system's capability to process them within acceptable timeframes. Actual test data will be used, reflecting the expected workload. The QA team will be responsible for this test.**
3. **Security Test: Security testing will assess the system's vulnerability to potential threats and risks, including testing for vulnerabilities, authentication mechanisms, and data protection. Test data will be created to simulate various attack scenarios. The QA team will be responsible for this test.**
4. **Automated Test: Automated testing will simplify the testing process and improve efficiency, creating automated test scripts and utilizing tools for regression testing and continuous integration. Test data will be derived from the test scenarios. The QA team will be responsible for this test.**
5. **Stress and Volume Test: Stress and volume testing will evaluate the system's performance under high-stress conditions and at maximum capacity. Test data will be generated to simulate extreme scenarios, pushing the system to its limits. The QA team will be responsible for this test.**
6. **Recovery Test: Recovery testing will assess the system's ability to recover from failures or disruptions. It will involve intentionally causing failures and assessing the system's recovery mechanisms. The QA team will be created to simulate failure scenarios.**
7. **Documentation Test: Documentation testing will ensure the accuracy and completeness of the system's documentation, including user manuals, installation guides, and technical documentation. The QA team will be responsible for this test.**
8. **Beta Test: Beta testing will involve releasing the system to a limited number of external users to gather feedback and identify any issues or usability concerns. Real users will provide test data based on their usage of the system. The QA team and team leader will be responsible for executing beta tests.**
9. **User Acceptance Test: The end-users or their representatives will conduct the user acceptance test to validate whether the system meets their requirements and expectations. Objective test data reflecting actual usage scenarios will be utilized. The team leader and the QA team will collaborate in executing these tests.**

## **Test Design Process**

**The test design process involves several steps to ensure adequate testing of software.**

1. **Understanding Requirements: Start by thoroughly understanding the system's requirements which includes studying the functional and non-functional aspects of the software to know what needs to be tested.**
2. **Building a Traceability Matrix: Create a matrix that links the requirements to the corresponding test cases, which helps ensure that all requirements are covered and helps identify any gaps or inconsistencies.**
3. **Preparing Test Cases: Develop test cases based on the requirements and traceability matrix. Test cases describe specific inputs, expected outcomes, and steps for executing the tests. They cover various scenarios to ensure the software functions as intended.**
4. **Review by Quality Assurance Team: Have another quality assurance team member review the test cases, which helps ensure their clarity, accuracy, and completeness. The reviewer provides feedback and verifies that the test cases align with the requirements.**

**By following these steps, the testing team ensures that the software is thoroughly tested, all requirements are addressed, and potential issues are identified early on.**

# **Environment Requirements**

**The test environment for the package distribution project requires specific hardware, software, and data to facilitate effective testing. Here is an overview of the test environment:**

1. **Hardware:**

**Test computers/workstations: Adequate hardware configurations to support the testing activities.**

**Network connectivity: Stable internet connection for any network-related testing.**

1. **Software:**

**Operating System: Compatible operating systems for running the testing tools and applications.**

**Development Environment: Software development tools required for building and executing the project code.**

**Testing Tools: Specialized tools for automated testing, performance testing, and other specific testing needs.**

**Simulation Tools: Software or tools that simulate the city map, routes, trucks, and package data for testing purposes.**

1. **Test Data:**

**City Map Data: Predefined or generated city map data with buildings, roads, and grid information.**

**Package Data: Sample package information, including weight, box size, and destination coordinates, for test case scenarios.**

**Test Datasets: Various test datasets representing scenarios, such as total truck capacity, multiple package assignments, and diversions.**

1. **Network and Connectivity:**

**Internet Access: For accessing external resources or services, if applicable.**

**Local Network: For communication between the test computers/workstations and any server components involved in the project.**

1. **Test Environment Setup:**

**Installation Guide: Detailed instructions on setting up the necessary software and tools for test computers/workstations.**

**Configuration Guide: Guidelines for configuring the testing tools, simulation tools, and other required components.**

**Data Setup: Instructions for setting up the test data, including the city map, routes, and package information.**

**Test Environment Validation: Verification steps to ensure the test environment is correctly configured and ready for testing.**

**It is crucial to ensure that the test environment closely resembles the production environment to provide accurate testing results. The test environment setup should be documented, allowing the testing team to recreate the environment whenever necessary.**

**By providing the appropriate hardware, software, test data, and network connectivity, the test environment enables the execution of comprehensive tests, validating the efficiency and accuracy of the package distribution algorithm.**

# Execution Strategy

## Entry and Exit Criteria for Testing in the Package Distribution Project:

### Entry Criteria:

* + 1. Requirement Understanding: Understand the functional and non-functional requirements for package distribution and truck routing.
    2. Test Environment Setup: The test environment, including hardware, software, and data, is properly configured and ready for testing.
    3. Test Plan and Test Cases: A test plan is prepared, and test cases are developed based on the requirements, covering various scenarios.
    4. Test Data Availability: Sufficient test data is available, including package details, truck routes, and city map data.
    5. Test Resources: Adequate resources, including testers, test tools, and necessary documentation, are available for testing.
    6. Defect Management System: A defect tracking system or process is in place to report and manage any identified defects during testing.
    7. Test Schedule: Defined timeline and schedule for executing the testing activities.

### Exit Criteria:

* + 1. Test Completion: As per the test plan, all identified test cases have been executed.
    2. Test Coverage: Adequate test coverage is achieved, ensuring that critical functionalities and scenarios are thoroughly tested.
    3. Defect Closure: All critical and high-priority defects have been addressed, retested, and closed.
    4. Performance Goals: Performance testing objectives, such as response time and throughput, are met per the defined criteria.
    5. Stability and Reliability: The system demonstrates stability and reliability by running without major crashes or failures during testing.
    6. Regression Testing: Regression tests are conducted to ensure that changes made during testing have not introduced new issues or affected existing functionalities.
    7. Documentation: Test artefacts, including test plans, cases, and results, are appropriately documented, and stored for future reference.
    8. Stakeholder Approval: Relevant stakeholders, such as project managers or team leaders, have reviewed and approved the test results and overall testing process.
    9. Test Summary Report: A comprehensive test summary report is prepared, providing an overview of the testing activities, outcomes, and any outstanding issues or recommendations.

By adhering to this entry and exit criteria, the testing process in the package distribution project ensures that all necessary prerequisites are met before commencing testing and that the testing activities are executed effectively. The exit criteria define the conditions that must be fulfilled to consider testing complete and successful, allowing for the accurate assessment of the system's readiness for deployment. You can describe the severity of defects in this section and break them down into severity levels.

## Severity Levels of Defects

In the package distribution project, defects can be classified into different severity levels based on their impact:

* + 1. Critical: These defects cause the system to crash or produce incorrect results. They are serious and need immediate attention to prevent significant issues.
    2. High: Defects of high severity result in the lack of essential system functionality. Although workarounds may exist, there are better solutions that affect the system's usability.
    3. Medium: Medium severity defects degrade the system's quality but usually have workarounds to achieve the desired functionality. They impact specific features or performance, requiring attention.
    4. Low: Low-severity defects have minimal impact on system functionality. They include minor errors like unclear error messages or spelling mistakes, which don't significantly affect usability.
    5. Cosmetic: Cosmetic defects are superficial issues that make the user interface less optimal in appearance. While they don't affect functionality, addressing them improves the overall user experience.

By categorizing defects into severity levels, the development team can prioritize their resolution, focusing on critical and high-severity issues first. Medium, low, and cosmetic defects can be addressed in subsequent iterations or updates to enhance the system's quality and user satisfaction.

## Test Reporting

As part of the testing process in the package distribution project, various reports should be produced to provide valuable insights and updates on the testing progress and results. These reports serve as a means of communication and documentation for stakeholders involved in the project. The frequency of report generation and the recipients may vary depending on the project's timeline and organizational requirements.

### Test Execution Reports:

1. Generated after each round of test execution or at defined milestones.
2. Sent to the project manager, development team, and quality assurance team.
3. Contents may include:
   * + 1. Summary of test cases executed, passed, failed, and pending.
       2. Defects identified and their severity levels.
       3. Overview of test coverage and areas tested.
       4. Status of the test environment and any issues encountered.
       5. Recommendations for further testing or improvements.

### Defect Reports:

1. Generated whenever defects are identified during testing.
2. Sent to the development team, project manager, and quality assurance team.
3. Contents may include:
   1. Description of each defect, including steps to reproduce.
4. Severity level and impact on system functionality.
5. Assigned defect status (open, in progress, resolved, closed).
6. Defect resolution timeline and progress updates.
7. Any additional notes or observations related to the defects.

### Test Progress Reports:

1. Generated periodically to track overall testing progress.
2. Sent to project stakeholders, including management and relevant teams.
3. Contents may include:
4. Overview of test activities performed and remaining.
5. Test coverage metrics and progress towards completion.
6. Summary of test results and any significant findings.
7. Risk assessment and mitigation strategies.
8. Timeline updates and deviations from the original plan.
9. Resource allocation and utilization status.

The frequency of report generation will depend on the project's timeline and the stakeholders' needs. Typically, test execution reports and defect reports are produced after each test cycle, while test progress reports are generated weekly or at regular intervals. However, the exact timing and frequency should be determined in collaboration with project stakeholders to ensure effective communication and timely decision-making. Reports should be shared with relevant project stakeholders, including project managers, development teams, quality assurance teams, and other critical decision-makers. These reports provide valuable insights into the testing process, highlight critical issues, and facilitate informed decision-making regarding system improvements, defect resolution, and project timelines. This section could also have details of how the testers are going to feed information back to the project managers so that they can assign developers to fix the bugs. This section can detail the communication to occur between management, the development team, and the quality assurance team.

# **Test Schedule**

**Testing Schedule and Estimated Duration (6-week timeline):**

|  |  |
| --- | --- |
| **Week** | **Activities** |
| **1** | * **Review project requirements and testing objectives.** * **Identify test scenarios, test cases, and test data requirements.** * **Define test environment and tools setup.** * **Create the test plan and obtain necessary approvals.** |
| **2** | * **Design and document test cases based on identified scenarios.** * **Map test cases to requirements for traceability.** * **Review and refine test cases for accuracy and completeness.** * **Prepare necessary test data and test environment setup.** * **Creating Blackbox testing** |
| **3 - 4** | * **Execute test cases based on the defined test plan.** * **Record test results and capture any defects or issues.** * **Verify system functionality against expected outcomes.** * **Conduct regression testing as needed.** * **Iteratively execute test cases and retest resolved defects.** |
| **5** | * **Log defects and issues identified during test execution.** * **Assign severity levels and prioritize defect resolution.** * **Collaborate with the development team to investigate and fix defects.** * **Retest resolved defects and update defect status.** |
| **6** | * **Finalize test execution and ensure all planned tests are completed.** * **Compile test results and generate test summary reports.** * **Conduct a final review of the testing process and outcomes.** * **Share testing reports with stakeholders for review and feedback.** |

**Estimated Testing Duration: 6 weeks doing along with development team**

# **Control Procedures**

1. Reviews: Reviews involve examining project documents to ensure they are accurate and complete. Team members or stakeholders not directly involved in development or testing conduct these reviews to identify errors or areas for improvement.
2. Bug Review Meetings: Bug review meetings gather relevant stakeholders to discuss, and address identified defects. The purpose is to prioritize bug fixes based on severity and impact, ensuring timely resolution.
3. Change Request: Change requests manage modifications to the project. When a change is needed, a formal request is submitted, reviewed, and approved based on feasibility, risks, and project objectives. Approved changes follow a standardized process.
4. Defect Reporting: Defect reporting ensures the identification, documentation, and tracking of defects found during testing. Defect reports describe the defect and its severity, allowing responsible parties to investigate and resolve them.

The Package Distribution Project can maintain quality, facilitate collaboration, and effectively manage changes and defects by implementing these control procedures.

# **Functions To Be Tested**

1. int getNumRows(const struct Map\* map);
2. int getNumCols(const struct Map\* map);
3. void printMap(const struct Map\* map, const int base1, const int alphaCols);
4. struct Map addRoute(const struct Map\* map, const struct Route\* route);
5. void addPtToRoute(struct Route\* route, struct Point pt);
6. void addPointToRouteIfNot(struct Route\* route, const int row, const int col, const struct Point notThis);
7. void addPointToRoute(struct Route\* route, const int row, const int col);
8. struct Route getBlueRoute();
9. struct Route getGreenRoute();
10. struct Route getYellowRoute();
11. double distance(const struct Point\* p1, const struct Point\* p2);
12. struct Route shortestPath(const struct Map\* map, const struct Point start, const struct Point dest);
13. struct Route getPossibleMoves(const struct Map\* map, const struct Point p1, const struct Point backpath);
14. int getClosestPoint(const struct Route\* route, const struct Point pt);
15. int eqPt(const struct Point p1, const struct Point p2);

# **Resources and Responsibilities**

Resources

Resources include details such as hardware, software, testing tools, testing environments, and any other resources necessary for the testing process.

## Responsibilities

* 1. Project Manager is responsible for overseeing the entire project and coordinating with the testing team to ensure testing aligns with project objectives and timelines.
  2. Technical Lead is responsible for overseeing the technical aspects of a project and providing technical guidance and expertise.
  3. Developers is responsible for fixing defects identified during testing and collaborating with the testing team to resolve any issues.
  4. Test Manager is responsible for overseeing the entire testing process, managing the testing team, coordinating with stakeholders, and ensuring the overall quality of the testing effort.

The QA teams will execute the tests and produce the test deliverables. The team leader will provide feedback and participate in user acceptance testing.

# **Deliverables**

The test deliverables will include test plans, test cases, test scripts, test data, test reports, and any necessary documentation related to test results and defects. The QA team will produce these deliverables throughout the testing process.

# **Exit Criteria**

Testing is considered finished when the execution of all test data has been completed and a minimum of 95% of the tests have been successful. Additionally, there should be no remaining defects classified as critical, high, medium, or low priority.

# **Resumption Criteria**

The testing process will be resumed when the fixes for reported defects have been

implemented and verified, the software build is stable, the test environment is ready, required test data is available, and the defined resumption criteria have been fulfilled.

# **Dependencies**

## Personnel Dependencies

1. Project Manager: Siripa Purinruk
2. Test Manager: Bussarin Apichitchon
3. Technical Leader: Seyed Iman Modarres Sadeghi
4. Developers: Kishan Dewasi
5. Developers: Dhrumit Ketan Parekh
6. Developers: Jaskaran Singh
7. Developers: Farbod Maoyari
8. Developers: Varshilkumar Ileshkumar Parikh

## Software Dependencies

Software dependencies refer to the external software components or libraries the project relies on to function correctly. In the case of the Package Distribution Project written in C language, specific software dependencies must be considered. These can include:

* + 1. Compiler and Development Environment: The project might depend on a specific C compiler, such as GCC or Clang, along with the associated development environment (IDE or text editor) used for coding and building the program.
    2. Standard C Libraries: The project may rely on standard C libraries the programming language provides, such as stdio.h, stdlib.h, string.h, math.h, etc., for various functionalities like input/output operations, memory management, string manipulation, and mathematical calculations.
    3. External Libraries: Depending on the Package Distribution Project's specific requirements, external libraries might need to be used to facilitate certain functionalities. These libraries can be related to file handling, data structures, networking, or any other tasks the project requires.
    4. Operating System APIs: The project might utilize operating system-specific APIs (Application Programming Interfaces) to interact with the underlying system, access system resources, perform file operations, or handle other platform-dependent functionalities.

## Hardware Dependencies

Hardware dependencies refer to the specific hardware components or devices that the project relies on for proper execution. In the case of the Package Distribution Project written in C language, there might be certain hardware dependencies to consider. These can include:

* + 1. Computer System: The project requires a computer system to run the compiled executable program. This includes a processor, memory (RAM), and storage (hard drive or SSD) to host the operating system and execute the program.
    2. Input and Output Devices: The project may interact with various input and output devices, such as a keyboard, mouse, monitor, or printer. These devices allow users to input data, view program output, and interact with the application.
    3. Network Connectivity: If the Package Distribution Project involves communication with external systems or network operations, it might rely on network connectivity. This includes network interface cards, routers, and internet connectivity for data exchange with remote servers or devices.
    4. Storage Devices: The project might require access to specific storage devices, such as hard drives or solid-state drives (SSDs), to read or write data related to package distribution. This can include storing and retrieving information about packages, routes, or other relevant data.

## Test Data & Database

In the context of the Package Distribution Project developed in C language, Test Data and Database play an essential role in validating the functionality and performance of the program. Here's an overview of Test Data and Database considerations:

### Test Data:

Test data refers to the specific data inputs used to validate the program's functionality. It includes a range of scenarios and input values that cover different test cases. For the Package Distribution Project, test data might include:

1. Various package weights and sizes.
2. Row numbers and column letters represent different destinations.
3. Test cases to cover edge cases, such as maximum weight or size limits.
4. Invalid or erroneous inputs to test error handling and validation.

The test data should be designed to exercise different parts of the program and cover both expected and unexpected scenarios. It helps ensure the program performs as intended and correctly handles different inputs.

### Database:

1. The Package Distribution Project may involve storing and retrieving data related to packages, routes, and other relevant information. In such cases, a database can manage and organize this data. As the project is written in C language, a common approach is to use a lightweight and embedded database system, such as SQLite.
2. The database will store package details, truck routes, and delivery statuses. It enables efficient data retrieval and manipulation, ensuring accurate package distribution.
3. During testing, ensure that the database functions appropriately, data is stored correctly, and the program interacts with the database seamlessly, which may involve creating test scenarios that validate database operations, including data insertion, retrieval, and updating.

# **Risks**

## Schedule Risk

1. Delays in the development and testing phases may occur, leading to a potential risk of failing to meet the project deadlines.
2. Factors such as unforeseen complexities, resource constraints, or changes in requirements can impact the project schedule.

## Technical Risk

1. Using C language for the project may pose technical challenges, such as managing memory efficiently, ensuring proper error handling, or handling complex algorithms.
2. Integration issues with external systems or dependencies may arise, impacting the overall functionality and performance of the program.

## Management Risk

1. Inadequate project planning, ineffective communication, or poor coordination among team members and stakeholders can result in project management challenges.
2. Lack of clear project goals, scope creep, or insufficient allocation of resources may hinder the project's success.

## Personnel Risk

* + 1. Availability of skilled developers and testers proficient in C language may pose a risk to the project's execution.
    2. Team dynamics, including conflicts, turnover, or insufficient collaboration, may impact the overall productivity and quality of the project.

## Requirements Risk

* + 1. Unclear or ambiguous requirements may lead to misunderstandings and misalignment between stakeholders and the development team.
    2. Changes in requirements during the project lifecycle may require additional effort and impact the project schedule and deliverables.

# **Tools**

1. IDE (Integrated Development Environment) such as
2. Eclipse CDT
3. Code::Blocks
4. Dev-C++
5. VS
6. XCode
7. Version Control System (VCS) can track changes and collaborate with team members such as
8. Git
9. SVN (Subversion)
10. Mercurial
11. Testing Frameworks can be used to create and run automated tests to ensure the quality of code sch as
12. CUnit
13. Unity
14. Check
15. Code Analysis Tools can identify your C program's coding errors and memory issues such as
16. Cppcheck
17. Valgrind
18. Clang Analyzer
19. Documentation Tools can generate documentation from code comments to explain the functionality of program such as
20. Doxygen
21. Natural Docs

# **Documentation**

The team is responsible for producing and maintaining several documents  
throughout the project. These documents include:

1. Test Plan: The current document that outlines the initial plan for the project and serves as a reference throughout the testing process.
2. Blackbox Unit Testing Matrix: A detailed document listing the test data specifically for Milestone 3, focusing on blackbox unit testing.
3. Blackbox Unit Test Code: The code for the blackbox unit tests written in C using the Native Unit Testing Module.
4. Whitebox Unit Testing Matrix: A document containing additional test data specifically for Milestone 4, focusing on whitebox unit testing.
5. Unit Testing Matrix: A combined document where both blackbox and whitebox test data is executed and recorded.
6. Bug Fixes: A document that captures and maintains a record of any bugs identified and fixed during the development process.
7. Integration Testing Matrix: A document containing the test data and execution details for the integration tests.
8. Acceptance Testing Matrix: A document containing the test data and execution details for the acceptance tests.
9. Final Test Report: A comprehensive report summarizing the testing activities conducted throughout the project, including the tests executed, bugs fixed, and the final testing results.

# **Approvals**

Test approval will be granted through a team meeting where all members gather to thoroughly discuss and validate the testing results. This session aims to ensure the accuracy of the testing process.