**Test plan**

1. **Introduction: -**

**Test plan objectives: -**

The project includes the generation of a package-delivery application for a

a local delivery company that operates three distinct trucks that are assigned to three separate routes. When a shipment arrives, the program will assess which truck has sufficient capacity to accommodate the package and can also deliver it as close as possible to the intended destination. Available data from the customer is the **weight** of the shipment (kg), the **size** of the box (cubic meters), and the **destination** (row number, column letter ex. 3T).

The test plan's goal is to ensure the dependability and quality of the local delivery company system by confirming its operation, determining performance and capacity constraints, testing stability, gauging the reliability of decision-making, and assuring user satisfaction. A tested and dependable system that efficiently distributes packages, maintains consistent performance, and provides a positive user experience, is what is anticipated as a result. Identified problems or flaws will be fixed to raise the functioning and general quality of the system.

1. **SCOPE:-  
     
   what to test:-**

**Performance Evaluation**: The performance of the system will be thoroughly evaluated to assess its 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.

**Functionality Validation:** Extensive testing will be conducted to validate the system's functionality, specifically focusing on the accurate assignment of packages to trucks based on available space and proximity to the destination. Special attention will be given to testing the system's ability to calculate the shortest path, accounting for obstacles such as buildings.

**Capacity Testing:** Rigorous testing will be performed to determine the capacity of individual trucks and the system as a whole. This testing will assess the system's ability to handle packages of diverse sizes and weights while adhering to predefined weight and volume limitations.

* Truck Limitations**:** 
  + Max weight: 1000kg
  + Max room: 36 cubic meters
* Package Limitations:
  + Boxes in cubic meters must be = .25, .5, 1.
  + All boxes are squares so they have the same dimensions on all sides.

**System Stability Assessment:** The stability and robustness of the system will be examined to ensure its resilience in handling unexpected situations and errors. Comprehensive testing will evaluate the system's error-handling mechanisms, exception-handling processes, and its ability to recover from failures or disruptions.

**What not to test:-**

**External Dependencies:** While integration with third-party APIs or services may be considered within the scope of functionality and performance testing, detailed testing of these external dependencies is not explicitly addressed in this test plan.

**Security and Penetration Testing:** Comprehensive security testing, such as penetration testing or vulnerability assessments, is not specifically covered by this test plan. However, security considerations will be taken into account during the testing process.

**Hardware Assessment:** The scope of this test plan does not extend to the evaluation of physical hardware components, including trucks and their specific capabilities. The focus remains primarily on the software functionality of the local delivery system.

**Network Infrastructure**: Testing the underlying network infrastructure, which supports the delivery system, is not included in this test plan. The main emphasis is placed on assessing the software's functionality and performance.

1. **TEST STRATEGY:-   
   how to carry out the work:-**

The test execution and methodology for the local delivery system will follow a comprehensive approach to ensure the system's quality and reliability. The system test will be conducted to validate the overall functionality, including the accurate assignment of packages to trucks, route optimization, and the calculation of the shortest path. To achieve this, various test data will be generated to simulate different scenarios and edge cases, ensuring the system's robustness. The responsibility for conducting the system test lies with the QA team, who will closely monitor and assess the system's behavior.

Performance testing will play a crucial role in evaluating the system's responsiveness and efficiency under various workloads. By simulating high volumes of incoming shipments, the performance test will measure the system's capability to process and allocate packages within acceptable timeframes. Actual test data reflecting the expected workload will be used, allowing the QA team to assess how the system handles peak periods and heavy traffic. By identifying potential bottlenecks and optimizing performance, the delivery system can deliver a smooth and efficient user experience.

Furthermore, security testing will be conducted to assess the system's vulnerability to potential threats and risks. This will involve testing for vulnerabilities, examining authentication mechanisms, and evaluating data protection measures. Test data will be created to simulate different attack scenarios, allowing the QA team to identify and address any security weaknesses. By conducting rigorous security testing, the local delivery system can ensure the confidentiality, integrity, and availability of data, protecting it from unauthorized access and potential breaches.

Automated testing will be leveraged to streamline the testing process and enhance efficiency. Through the creation of automated test scripts and the utilization of tools for regression testing and continuous integration, the QA team can perform repetitive tests and validate system functionality with speed and accuracy. Test data derived from various test scenarios will be used to ensure the effectiveness of the automated testing approach. By implementing automated testing, the local delivery system can reduce manual effort, increase test coverage, and detect defects more efficiently.

Stress and volume testing will be employed to evaluate the system's performance under high-stress conditions and at maximum capacity. By generating test data that simulates extreme scenarios, the QA team will push the system to its limits to assess its stability, responsiveness, and scalability. This type of testing allows for identifying any potential performance issues and ensuring the system's robustness and reliability, even under demanding circumstances.

Recovery testing will focus on assessing the system's ability to recover from failures or disruptions. The QA team will intentionally cause failures and assess the system's recovery mechanisms, including error handling and fault tolerance. By conducting recovery testing, the local delivery system can demonstrate its resilience and the effectiveness of its recovery procedures, minimizing downtime and ensuring smooth operation even in the face of unexpected events.

Documentation testing will be carried out to verify the accuracy and completeness of the system's documentation, including user manuals, installation guides, and technical documentation. The QA team will review and validate the documentation against the actual system functionalities to ensure consistency and clarity. By conducting thorough documentation testing, the local delivery system can provide users with reliable and up-to-date information, facilitating their understanding and effective utilization of the system.

**Test design process:-**

The test design process plays a critical role in ensuring thorough software testing for the local delivery system. It involves several important steps to effectively plan and execute the tests.

The first step is to thoroughly understand the system requirements. This includes studying both the functional and non-functional aspects of the software to gain a clear understanding of what needs to be tested. By comprehensively understanding the requirements, the testing team can ensure that all critical areas of the system are covered during the testing process.

The next step is to build a traceability matrix. This matrix serves as a valuable tool to link the system requirements to the corresponding test cases. By creating this matrix, the testing team can ensure that each requirement is properly addressed and that all test cases are aligned with the intended functionalities of the software. It also helps identify any gaps or inconsistencies in the test coverage.

Once the traceability matrix is in place, the testing team proceeds with preparing test cases. Test cases are designed to cover various scenarios and test the software against different inputs and expected outcomes. These test cases provide detailed steps for executing the tests and serve as a guide to validate the functionalities of the local delivery system. During this phase, it is crucial to consider edge cases and boundary conditions to ensure comprehensive test coverage.

To maintain the quality of the test design, it is important to have a review process by involving another quality assurance team member. This review helps ensure the clarity, accuracy, and completeness of the test cases. The reviewer provides valuable feedback, identifies any potential improvements or gaps, and verifies that the test cases effectively align with the system requirements.

By following these steps, the testing team can ensure that the local delivery system undergoes thorough testing. This process ensures that the software is tested against all relevant requirements, minimizing the risk of undetected defects and improving the overall quality of the system. By identifying potential issues early on, the testing team can work closely with the development team to address them promptly, resulting in a more reliable and robust local delivery system.

1. **ENVIRONMENT REQUIREMENTS:-**

**Test Environment Setup Process:**

Detailed installation and configuration instructions should be available to set up the necessary software and tools on the test computers/workstations. Additionally, guidelines for data setup and validation steps to ensure the test environment's accuracy and readiness are crucial.

**Hardware Configuration:** For network-related testing, the test environment has to include properly setup test computers and workstations as well as consistent internet connectivity.

**Software requirements:** Compatible operating systems, development environments, and specialized testing tools should be present in the test environment. Additionally, city maps, routes, trucks, and package data can all be replicated using simulation tools.

The test environment should have predefined or created city map data, sample package data, and numerous test datasets representing different scenarios in order to conduct thorough testing.

**Network and Connectivity Setup:** The test environment should have an internet connection so that users of external resources can communicate with one another and with essential server components.

1. **EXECUTION STRATEGY**

**Entry criteria:-**

1. To ensure a systematic and effective testing process in the package distribution project, specific entry and exit criteria have been established. The entry criteria serve as prerequisites that must be met before commencing testing. These include a clear understanding of the project's functional and non-functional requirements, the proper setup of the test environment encompassing hardware, software, and data, the development of a comprehensive test plan and corresponding test cases, availability of sufficient test data, allocation of necessary test resources, implementation of a defect management system, and the definition of a test schedule.

**Exit criteria:-**

The exit criteria define the conditions that must be fulfilled to consider testing complete and successful. These criteria include the execution of all identified test cases, adequate coverage of critical functionalities and scenarios, resolution and verification of all critical and high-priority defects, achievement of performance goals as defined by response time and throughput, demonstration of system stability and reliability with minimal crashes or failures, conducting regression testing to ensure that changes made during testing do not introduce new issues or impact existing functionalities, proper documentation of test artifacts for future reference, approval of test results and the overall testing process by relevant stakeholders, and the creation of a comprehensive test summary report summarizing the testing activities, outcomes, and any outstanding issues or recommendations.

By adhering to these entry and exit criteria, the testing process ensures that the necessary prerequisites are met before testing begins and provides a clear set of conditions for assessing the completion and success of the testing phase in the package distribution project.

**b) Security levels of defect**

Categorizing defects into severity levels allows the development team to effectively manage and prioritize their efforts in addressing the most critical issues that can significantly impact system functionality and user experience. By promptly resolving critical and high-severity defects, the team can ensure the system's stability and prevent major disruptions.

Medium-severity defects, while not as severe as critical or high-severity ones, still require attention to maintain the system's overall quality and performance. These defects may affect specific features or performance aspects, and their resolution can contribute to improving the user experience and system functionality.

Low severity and cosmetic defects, although less impactful, should not be overlooked. While they may not directly affect system functionality, addressing these issues demonstrates attention to detail and a commitment to providing an optimal user interface and experience.

By addressing defects across different severity levels, the development team can incrementally enhance the system's quality, ensuring a reliable and user-friendly package distribution solution.

**c) Test reporting**

In the package distribution project, various reports are essential to communicate and document the testing progress and results. Test execution reports are generated after each test round, providing a summary of executed test cases, identified defects, test coverage, and recommendations for further testing. Defect reports describe each identified defect, including its description, severity, status, resolution timeline, and any additional notes. Test progress reports track overall testing progress, including test activities, coverage metrics, results, risk assessment, and timeline updates. The frequency of report generation and the recipients depend on the project's timeline and stakeholder requirements. Reports should be shared with project stakeholders, including project managers, development teams, and quality assurance teams, to facilitate effective communication and decision-making.

1. **Test schedule**

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

|  |  |
| --- | --- |
| **Milestone** | **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.** |

1. **Control procedures:-**

In the package distribution project, various reports are essential to communicate and document the testing progress and results. Test execution reports are generated after each test round, providing a summary of executed test cases, identified defects, test coverage, and recommendations for further testing. Defect reports describe each identified defect, including its description, severity, status, resolution timeline, and any additional notes. Test progress reports track overall testing progress, including test activities, coverage metrics, results, risk assessment, and timeline updates. The frequency of report generation and the recipients depend on the project's timeline and stakeholder requirements. Reports should be shared with project stakeholders, including project managers, development teams, and quality assurance teams, to facilitate effective communication and decision-making.

1. **Functions to be tested:-**
2. int getNumRows(const struct Map\* map);
3. int getNumCols(const struct Map\* map);
4. void printMap(const struct Map\* map, const int base1, const int alphaCols);
5. struct Map addRoute(const struct Map\* map, const struct Route\* route);
6. (void addPtToRoute(struct Route\* route, struct Point pt);
7. void addPointToRouteIfNot(struct Route\* route, const int row, const int col, const struct Point notThis);
8. void addPointToRoute(struct Route\* route, const int row, const int col);
9. struct Route getBlueRoute();
10. struct Route getGreenRoute();
11. struct Route getYellowRoute();
12. int canCarryWeight(const Truck\* truck, double shipmentWeight)
13. int canCarryVolume(const Truck\* truck, double shipmentVolume)
14. int addShipmentToTruck(Truck\* truck, double shipmentWeight, double shipmentVolume)
15. int findBestTruckForShipment(const Truck\* trucks, int numTrucks, const Shipment\* shipment, const Map\* map)
16. struct Route addDeliveryDiversion(Truck\* truck, const Map\* map, const Point)
17. struct Route addDeliveryDiversion(struct Truck\* truck, const struct Map\* map, const struct Point destination);
18. **Deliverables:-**

Depending on the requirements of each milestone which need to be submitted on the git-hub by the team lead and documents to be mailed to the professor by the project manager.

1. **) Dependencies**

a. Software Dependencies:

Software dependencies in the Package Distribution Project are the external software components or libraries that the project relies on for proper functionality. In the context of the C language used in this project, specific software dependencies must be considered. These dependencies include the compiler and development environment, which may require a specific C compiler such as GCC or Clang, along with an associated integrated development environment (IDE) or text editor for coding and building the program. The project also relies on standard C libraries, such as stdio.h, stdlib.h, string.h, math.h, for various functionalities like input/output operations, memory management, string manipulation, and mathematical calculations. Additionally, external libraries might be necessary to facilitate specific functionalities based on the project's requirements, such as libraries related to file handling, data structures, networking, or other required tasks. Furthermore, the project might utilize operating system-specific APIs to interact with the underlying system, access system resources, perform file operations, or handle other platform-dependent functionalities.

b. Hardware Dependencies:

Hardware dependencies in the Package Distribution Project refer to the specific hardware components or devices necessary for proper execution. In the context of the C language used in this project, there are several hardware dependencies to consider. The project requires a computer system with a processor, memory (RAM), and storage (hard drive or SSD) to host the operating system and execute the compiled program. Input and output devices, such as a keyboard, mouse, monitor, or printer, enable user interaction, data input, and program output viewing. If the project involves communication with external systems or network operations, it relies on network connectivity, which includes network interface cards, routers, and internet connectivity for data exchange with remote servers or devices. Storage devices, such as hard drives or solid-state drives (SSDs), are necessary for reading from and writing to data related to package distribution. This can include storing and retrieving information about packages, routes, or other relevant data. By considering these hardware dependencies, the Package Distribution Project can ensure that the necessary hardware components are available to support its execution and functionality.

1. **Tools:-**

a. IDE (Integrated Development Environment) such as

* 1. Eclipse CDT
  2. Code::Blocks
  3. Dev-C++

b. Testing Frameworks can be used to create and run automated tests to ensure the quality of code such as

* 1. CUnit
  2. Unity

c. Version Control System (VCS) can track changes and collaborate with team members such as

i. Git

ii. SVN (Subversion)

1. **Risks: -**

The Package Distribution Project entails several risks that need to be carefully managed throughout its execution. Schedule risk arises from **potential delays** in development and testing, influenced by unforeseen complexities, resource constraints, or changes in requirements. **Technical risk** exists due to challenges associated with using the C language, such as memory management, error handling, and integration with external systems. **Management risk** involves project planning, communication, and coordination issues, which can hinder success. **Personnel risk** stems from the availability of skilled C language developers and testers, as well as potential conflicts or turnover affecting team dynamics. Lastly, requirements risk encompasses the potential for unclear requirements and changes that may impact project alignment and add complexity. By proactively addressing these risks, the Package Distribution Project can increase the likelihood of a successful outcome.