Test Plan for Cargo Delivery Program

1. **Objective**

The Cargo **Delivery** Program (“Program”) is used to determine the package delivery plan, which will decide which truck the package will be placed in, print out where the truck will deliver the package, and print the path to divert from the main routes whenever necessary.

The **Program** will recommend a deliver plan that meets the following business purposes:

* 1. the **assigned** trucks can hold all the packages without exceeding their maximum capacity, either in terms of weight or volume;
  2. the assigned **trucks** will divert the least distances from the main routes to deliver the packages;
  3. when the **distance** to divert is the same for multiple trucks for a destination, the truck which is less full will be selected to deliver the package; and
  4. identify **packages** that cannot be taken by all trucks.

The testing for the Program will verify whether the Program being developed meets the above business **requirements**, identify and fix bugs that prevent the Program from performing its functions, and provide necessary documentations that record these activities.

1. **Scope**

**The testing for the Program will test all the functions required to meet the following business requirements, including:**

1. **to print a map with buildings and main routes marked;**
2. **to measure the diversion distance from a main route to a destination;**
3. **to identify which truck has the least diversion distance;**
4. **to calculate the truck’s remaining capacity;**
5. **to choose which truck should hold a given package with reference to its remaining capacity; and**
6. **to identify** **packages** that cannot be taken by all trucks and mark them to be shipped the next day.

**The testing will not test functions that are not required to meet the business requirements, such as calculating the costs of delivery, estimating time of delivery, calculating profit/loss from the delivery, scheduling the time of delivery during the day, etc.**

1. **Test Strategy**

**The test design process is outlined as follows:**

1. **Understanding the business requirements – all documents provided by the client are reviewed to understand their business processes and identify business requirements relevant to the design and development of the Program;**
2. **Building the traceability matrix – map the system functions that are relevant to achieving the specific business requirements in traceability matrix table;**
3. **Preparing test cases – for each function, design blackbox test cases before thoroughly studying the full logic of the functions. Then, design whitebox test cases after thoroughly studying the full logic of the functions, in order to ensure the functions still work for edge case data.**
4. **Reviewing by quality assurance team – test cases should be reviewed by quality assurance team before performing testing, and test reports should be reviewed by quality assurance team after performing testing. Bug fixes should be implemented after the approval from the quality assurance team.**
5. **System Test**

**The Program will be tested to ensure it can run on a Windows or Linux platform.**

**Estimated duration: continuously tested throughout the whole development cycle**

1. **Performance Test**

**The Program will be tested to ensure it completes all computations within 5 seconds on a regular notebook computer.**

**Estimated duration: continuously tested throughout the whole development cycle**

1. **Exploratory Test**

**The Program will undergo exploratory test first to ensure all critical defects are removed before subsequent tests. The exploratory test will include:**

1. **to create a blank map with at least 25 x 25 dimensions;**
2. **to return the number of rows and columns of the map;**
3. **to print a map with positions of all buildings;**
4. **to print a map with a single route;**
5. **to print a map with multiple routes with overlapping;**
6. **to indicate a package delivery destination on a map;**
7. **to set the capacity of a truck in terms of weight;**
8. **to set the capacity of a truck in terms of dimensions;**
9. **to set the loading of a package in terms of weight; and**
10. **to set the loading of a package in terms of dimensions.**

**Estimated duration: 1 week**

1. **Automated Test and Functional Test**

**Automated tests on the Program will be performed when all implementation of the Program is completed for the first time. This will include:**

1. **black box testing;**
2. **white box testing;**
3. **integration testing; and**
4. **acceptance testing,**

**The functions listed in “Section 8 – Functions To Be Tested” will be tested in an automated manner.**

**Estimated duration: 2 weeks**

1. **Stress and Volume Test**

**The Program will be tested to verify it can handle at least:**

1. **a map with 25 x 25 square grid;**
2. **3 main routes for trucks; and**
3. **625 (i.e. 25 x 25) package delivery destinations.**

**Estimated duration: continuously tested throughout the whole development cycle**

1. **User Acceptance Test**

**User Acceptance Test will be performed as a final verification the Program meet each of the state business requirements (R001 – R006).**

**Estimated duration: 1 week**

1. **Environment Requirements**

The Program should work on both Environment 1 and Environment 2.

Environment 1

* OS: Windows 10 or above
* Compiler: Visual Studio 2022

Environment 2

* OS: Linux - Matrix
* Compiler: GCC

1. **Execution Strategy**

Overall passing criteria:

* The software can demonstrate all the functionalities as specified in business requirement.
* No defect with severity "Critical", "High", "Medium" remain unsolved.
* 95% of defects with severity "Low" are solved.
* Solving of defect with severity "Feature Request" is optional.

Severity explanation:

* Critical: Blockage of system running including start-up failure, crashing in to midway.
* High: System is not crashed but fails to demonstrate functionality specified in the business requirement.
* Medium: Erroneous results are shown even if functionality can be demonstrated.
* Low: No error, but warning is shown.
* Feature Request: Improvement of user interface to facilitate human-machine interaction.

Test Reporting:

Each test case should include the following:

Test outline:

* Testing objectives
* Module tested
* Date of testing

Test data (each row in table):

* ID: Unique identifier to identify the case
* Description: Target functionality, blackbox / whitebox test, positive / negative case, normal / edge case
* Steps: Procedures of testing
* Test Data: Input data used
* Expected Results
* Actual Results
* Pass / Fail

If all the results in test report are passed, there is no further action needed for developer and the test report can be documented and filed for future reference. Otherwise, the test report will be a feedback to developers for further modification of the system. Tests will be performed again until the passing criteria are met.

Bug Reporting:

To facilitate the communication between quality assurance team and developer team, it is better to use JIRA as a bug reporting tools. After generating test report, new task of "Bug" type on JIRA can be created as ticket upon any "Fail" case reported. Development team can take the ticket to working on bug fixing and close all the tickets created. To create a ticket on JIRA, fill in the required fields:

* + Issue type: Bug
  + Summary: Brief outline of the bug
  + Test ID: Refer to test data in test report
  + Platform: Machine, OS, compiler
  + Assignee: development team member to be assigned
  + Severity: Critical, High, Medium, Low, Feature Request
  + Description: Detailed explanation of bug with input, expected result, actual result, steps to reproduce the bug as stated in test report (better with screenshot attached)
  + Labels: Which milestone (MS1, MS2, ...)

1. **Test Schedule**

|  |  |  |  |
| --- | --- | --- | --- |
| Test Activity | Estimated Duration | Start Date | Completion Date |
| MS4: Blackbox and Whitebox Testing | 5 days | July 18th, 2023 | July 23rd, 2023 |
| MS5: Integration Testing | 5 days | July 25th, 2023 | July 30th, 2023 |
| MS6: Acceptance Testing | 5 days | August 1st, 2023 | August 6th, 2023 |

1. **Control Procedures**

**Reviews:**

All team members, including developers, testers and quality assurance team, will meet at least once a week to review the test plans, cases and scripts.

**Test Environment:**

All team members, including developers, testers and quality assurance team, will discuss how to set up and manage the environment (hardware, software, network, etc.) required to perform the test: configuration of the test environment, configuration procedures, and maintenance and management of the environment.

**Change Request:**

All team members, including developers, testers and quality assurance team, will handle the receipt, approval and scheduling of change requests.

**Test Execution Schedule:**

All team members, including developers, testers and quality assurance team, will make the methods and procedures for creating a schedule plan for conducting tests.

**Risk Management:**

All team members, including developers, testers and quality assurance team, will consider procedures and methods for identifying and managing potential risks during testing.

1. **Functions To Be Tested**
2. struct Map populateMap();

* Create a map with the position of all buildings in it
* @returns a map with the position of all buildings added to it

1. int getNumRows(const struct Map\* map);

* Get the number of rows in a map.
* @param map - the map to query
* @returns the number of rows in the map.

1. int getNumCols(const struct Map\* map);

* Get the number of columns in a map.
* @param map - the map to query
* @returns - the number of columns in the map.

1. void printMap(const struct Map\* map, const int base1, const int alphaCols);

* Print the map usign the symbols:
* space = open space
* X = building
* B = blue route
* G = green route
* Y = yellow route
* . = B & G routes overlap
* - = B & Y routes overlap
* \* = G & Y routes overlap
* + = B & G & Y routes overlap
* P = a shortest path or route from one point to another
* @param map - map to print
* @param base1 - if true print row indices from 1 up otherwise 0 up
* @param alphaCols - if true print col header as letters, otherwise numbers

1. struct Map addRoute(const struct Map\* map, const struct Route\* route);

* Add a route to a map using the indicated symbol.
* @param map - map to add route to
* @param route - the route to add to the map
* @returns a copy of the original map with the route added to it

1. void addPtToRoute(struct Route\* route, struct Point pt);

* Add a point to a route
* @param route - the route to which the point should be added
* @param point - the point to add to the route.

1. void addPointToRouteIfNot(struct Route\* route, const int row, const int col, const struct Point notThis);

* Add a point to a route if it is not equal to another point.
* @param route - the route to which the point should be added
* @param row - the row of the point to add to the route.
* @param col - the column of the point to add to the route.
* @param notThis - the point will be added to the route ONLY if it is not equal to this point.

1. void addPointToRoute(struct Route\* route, const int row, const int col);

* Add a point to a route
* @param route - the route to which the point should be added
* @param point - the point to add to the route.

1. struct Route getBlueRoute();

* Build and return the route for the blue trucks.
* @returns - the route for the blue trucks.

1. struct Route getGreenRoute();

* Build and return the route for the green trucks.
* @returns - the route for the green trucks.

1. struct Route getYellowRoute();

* Build and return the route for the yellow trucks.
* @returns - the route for the yellow trucks.

1. double distance(const struct Point\* p1, const struct Point\* p2);

* Calculate the Euclidian distance between two points.
* @param p1 - the first point
* @param p2 - the second point
* @returns - the distance between p1 and p2.

1. struct Route shortestPath(const struct Map\* map, const struct Point start, const struct Point dest);

* Calculate the shortest path between two points so that the path does not pass through buildings.
* @param map - the map showing the location of buildings.
* @param start - the point to start from
* @param dest - the point to go to
* @returns - the shortest path from start to dest. If there is no path, then a Route of zero length is returned. If start and dest are the same point, it also returns a Route of zero length.

1. struct Route getPossibleMoves(const struct Map\* map, const struct Point p1, const struct Point backpath);

* Calculate all adjacent squares to a given point so that the squares do not overlap a building and do not include the back path.
* @param map - the map showing the location of buildings.
* @param p1 - the point to calculate possible moves for
* @param backpath - the previous point we visited on the path we travelled so we will exclude travelling backwards.
* @returns - a collection of adjacent points we could potentially move to.

1. int getClosestPoint(const struct Route\* route, const struct Point pt);

* Calculates the Euclidian distance from every point in a route to a single point and returns the index of the point in the route which is closest to the point.
* @param route - the route to use to find the closest point
* @param pt - the point to to find the member of the route which is closest to this point
* @returns - the index of the closest point on the route to the point or -1 if the route is empty.

1. int eqPt(const struct Point p1, const struct Point p2);

* Compare two points for equality.
* @param p1 - the first point
* @param p2 - the second point
* @returns - true if p1 is equal to p2

1. A function to set the capacity of a truck **in terms of weight**
2. **A function to set the capacity of a truck in terms of dimensions**
3. **A function to identify whether the weight or dimensions is the limiting factor of the capacity of a truck**
4. **A function to return the capacity of a truck in terms of percentage**
5. **A function to set the loading of a package in terms of weight**
6. **A function to set the loading of a package in terms of dimensions**
7. **A function to determine if a package can be loaded to a truck in terms of weight and dimensions**
8. **A function to return whether the truck has the shortest distance to divert to deliver a package**
9. **A function to return whether the truck has the largest remaining capacity in terms of percentage to deliver a package**
10. **A function to determine which truck should be assigned to deliver a package**
11. **A function to print the package(s) that cannot be delivered on the program run date**
12. **A function to print where the truck will deliver the package**
13. **A function to print the path to divert to deliver the package**
14. **Resources and Responsibilities**

**Developer:**

The developer is responsible for designing and developing the software application. They are crucial in creating the software solution according to the project requirements.

Responsibilities:

1. **Collaborate with the testing team:** The developer works closely with the testing team to understand the testing requirements and provide necessary support and information during the testing process. They actively participate in communication and coordination with testers.
2. **Address** defects **and issues:**During the testing phase, the developer addresses any defects or issues identified by the testing team. They analyze the reported defects, troubleshoot the root causes, and implement appropriate fixes or enhancements to improve the software quality.

**Tester:**

The tester is responsible for conducting the testing activities to verify the software application's functionality, performance, and quality.

Responsibilities:

1. **Develop test cases and scenarios:** Testers create comprehensive test cases and scenarios based on the project requirements. They ensure that the test cases cover different functionalities and software use cases.
2. **Execute test cases:** Testers execute the test cases and record the test results accurately. They meticulously follow the testing procedures and document any issues or defects encountered during the testing process.
3. **Identify and report defects:** Testers actively identify defects or issues in the software and report them to **the** development team. They provide detailed information about the discovered defects, including steps to reproduce, expected and actual results, and other relevant details.
4. **Collaborate with the development team:** Testers collaborate with the development team to discuss and resolve identified defects. They work closely with developers to ensure effective communication and timely resolution of issues.

**Quality Assurance (QA) Team:**

The QA team is responsible for ensuring the overall quality of the software application. They focus on establishing and maintaining high-quality standards throughout the software development lifecycle.

Responsibilities:

1. **Develop** and maintain test plan and strategy: The QA team creates a comprehensive test plan outlining the testing approach, objectives, scope, and timelines. They ensure that the testing activities align with the project requirements.
2. **Review and provide feedback:** The QA team reviews and provides feedback on various test artifacts, including test cases, scripts, and reports. They ensure that the testing process adheres to established standards and best practices.
3. **Conduct risk analysis:** The QA team performs risk analysis to identify potential risks and uncertainties in the software application. They assess the impact and probability of each risk and implement appropriate risk mitigation strategies to minimize the potential impact on the project.
4. **Coordinate bug review meetings:** The QA team coordinates meetings to discuss and prioritize reported defects. They ensure that bug review meetings involve relevant stakeholders and that the progress of defect resolution is actively monitored.
5. **Deliverables**

Test Plan:

Description: A comprehensive document outlining the testing approach, strategies, and objectives for the project.  
Submission Date: July 9th, 2023

Traceability Matrix:

 Description: A matrix mapping project requirements to associated test cases to ensure comprehensive coverage.  
Submission Date: July 16th, 2023

Test Cases and Test Data:

 Description: Detailed test cases and associated test data to verify the functionality of the system.  
Submission Date: July 23th, 2023; July 30th, 2023; August 6th, 2023

Test Report:

 Description: A summary report documenting the test results, including any issues or defects found during testing.  
Submission Date: July 23th, 2023; July 30th, 2023; August 6th, 2023

Source Code:

Description: The final version of the project's source code, including any modifications made during testing.  
Submission Date: August 6th, 2023

1. **Suspension / Exit Criteria**

***Criteria for the testing to be temporarily suspended***

* **If a critical defect or bug is identified that severity of the defect and its potential impact on the system's performance are significantly affected the testing stability and functionality of the system, the testing activities may be temporarily suspended.**
* **In case of unforeseen resource constraints such as unavailability of necessary hardware, software, or testing environment, testing activities may be temporarily suspended.**
* **If there are obstacles that prevent the execution of test cases, such as unmet prerequisites, incomplete or incorrect test data, or dependencies on external systems that are unavailable, testing activities may be temporarily suspended until the blockers are resolved.**
* **when a comprehensive set of tests has been executed, and the system has demonstrated satisfactory performance, reliability, and adherence to requirements, further testing activities may be temporarily suspended.**

***Exit Criteria for the testing to be permanently concluded***

* **When all identified test cases and test scenarios have been executed, all critical features and system behaviors have been adequately verified. Testing activities may be permanently concluded.**
* **The system should be evaluated against performance and usability standards, if the system meets all the specified acceptance criteria agreed upon by stakeholders, the testing activities can be permanently concluded.**

1. **Resumption Criteria**

**Triggers for Testing Interruption and Resumption**

**If specific circumstances occurs and testing is temporarily interrupted, the team should find out the solution to resolve the problem(s). Make sure the system or testing environment is stable and functional enough to continue testing without significantly impacting the overall quality and stability. After that, the testing can be resumed.**

**If the hardware, software, or testing environment were previously unavailable, those required resources must be made available to the testing team. Necessary prerequisites, test data, or external dependencies should be obtained in advance, any missing of them may cause testing process interruption. After resolving the hardware, software or testing environment issues, the testing can be resumed.**

**Decision-Making Authority for Testing Suspension**

**The stakeholders and project management must be aligned on the decision to resume testing, considering project priorities, revised timelines, and the potential impact of the interruption on the overall project schedule.**

**The authority to make the decision to suspend or resume testing activities typically lies with the project manager or test manager, who is responsible for overseeing the testing process. They may consult with the project key stakeholders, or the testing team before deciding on the suspension or resumption of testing.**

**Escalation and Contingency Plan**

**In case the resumption criteria cannot be met within a specified timeframe,** **the testing team should promptly notify the project manager or test manager about the inability to meet the resumption criteria.**

**The project manager or test manager should escalate the issue to the appropriate stakeholders or project authority, highlighting the reasons for the delay and any potential risks associated with further postponing testing activities and make decisions regarding the resumption of testing, including any necessary adjustments to the project plan, resources, or timelines.**

1. **Dependencies**
   1. Personnel Dependencies
      1. Skilled software developers: Availability of skilled software developers to build and maintain the application, resolving any route calculation or mapping issues.
      2. QA testers: Collaboration with QA testers to ensure the application functions correctly and meets quality standards.
      3. Project managers: Access to project managers for planning, coordination, and decision-making.
      4. Data analysts: Coordination with data analysts to analyze delivery data and optimize routes.
      5. UI/UX designers: Collaboration with UI/UX designers to create an intuitive and user-friendly interface.
      6. Technical writers: Availability of technical writers for test documentation.
      7. Drivers or delivery personnel: Access to drivers or delivery personnel for gathering feedback on usability and effectiveness.
   2. Software Dependencies
      1. Availability of APIs or libraries to integrate with the application for mapping and routing functionality.
      2. Compatibility with different operating systems and devices used by drivers and customer.
      3. Integration with a database system to store and retrieve delivery information.
   3. Hardware Dependencies
      1. Availability of development and testing devices like computers, smartphones, or tablets.
      2. Compatibility with hardware used by drivers for navigation.
      3. Adequate processing power and memory for efficient routing calculations.
   4. Test Data & Database
      1. Sample delivery data to simulate different scenarios for testing.
      2. Compatibility with the database management system for seamlessly working and storing.
      3. Compliance with data privacy regulation when storing and processing delivery information.
2. **Risks**
   1. Schedule
      1. Potential delays in the project timeline due to unforeseen technical issues or dependencies on external factors.
      2. Insufficient time allocated for testing and bug fixing, leading to compromised quality.
      3. Inaccurate estimation of development efforts, resulting in project delays.
   2. Technical
      1. Compatibility issues with different mapping services or address databases.
      2. Complexity in integrating the application with existing systems or technologies.
      3. Inadequate performance or scalability of the application when dealing with a large volume of delivery data.
   3. Management
      1. Inadequate project management practices, leading to miscommunication, scope creep, or resource misalignment.
      2. Lack of clear prioritization and decision-making, affecting the project's progress and deliverables.
      3. Insufficient stakeholder involvement and engagement, resulting in misalignment of expectations and requirements.
   4. Personnel
      1. Skill gaps or resource constraints, impacting the development, testing, or deployment phases.
      2. Limited availability of skilled software developers or testers, leading to resource constraints and potential delays in development and testing activities.
      3. Staff turnover or unavailability of key team members, affecting project continuity and knowledge transfer.
   5. Requirements
      1. Ambiguous **or changing requirements, requiring frequent adjustments and potential rework.**
      2. Incomplete **or inadequate understanding of user needs, resulting in a suboptimal user experience.**
      3. Misalignment **between stakeholder expectations and the delivered functionality.**
3. **Tools**

The software tools and technologies that will be utilized to enhance the efficiency and effectiveness are as follows:

**Jira:** to be used for project management and issue-tracking. It will be used for managing test cases, tracking defects, and overall project management. With Jira, teams can collaborate seamlessly, monitor progress, and efficiently manage the testing process.

**GitHub:** to be used for version control. It will enable team members to collaborate on code, manage repositories, and track changes. GitHub will be utilized to store and manage test scripts, test data, and other testing-related artifacts, ensuring version control and facilitating collaboration among team members.

1. **Documentation**

Milestone 2

- Data Structure

- Test Plan

Milestone 3

- Function Specifications

- Blackbox Test Documents

- Test Implementation Code

- Function-Test Matrix

Milestone 4

- Test Report

Milestone 5

- Integration Test Cases

- Function-Test Matrix Update

- Acceptance Test Cases

1. **Approvals**

The testing is approved subject to the completion of following tasks:

1. All test cases are performed.
2. All bugs of critical and high priorities are fixed.
3. Group members have reviewed all test case results and ensured that they meet the expected criteria.
4. Documentation, including test plans, test scripts, and test reports, are prepared and reviewed.
5. All stakeholders have provided their feedback and approval on the test plan and test results.
6. The application has undergone thorough testing and has met the acceptance criteria.
7. Any additional requirements or changes identified during the testing process have been addressed and implemented.
8. The testing team has conducted a final review of the test plan and associated documentation to ensure completeness and accuracy.

After completing these tasks, the testing process is considered approved, and the necessary documentation is in place to maintain proper communication, traceability, and quality assurance.