**TEST REPORT**

## **TESTING PROCESS (OVERVIEW)**

The testing process for this project was meticulously structured to ensure comprehensive validation of the software's functionality, compatibility, and efficiency. The testing activities were carried out in various stages, aligning with the program's development progress. The primary focus was on verifying the program's behaviour on different operating systems, assessing individual function performance through modular/unit testing, and validating multiple conditions using automated tests. The main goal was to ensure accurate package allocation to trucks and the optimal selection of routes for each truck.

### Test Plan Objectives:

The objectives of the test plan were as follows:

* **Compatibility Testing:** Validate program execution on both Windows and macOS operating systems.
* **Function Performance Testing:** Thoroughly test individual functions' performance through modular and unit testing.
* **Automated Tests:** Employ automated tests to evaluate diverse conditions and confirm adherence to requirements, simultaneously easing the testing process.
* **Diversion Calculation Testing:** Validate that the program accurately assigns packages to trucks and selects the most efficient routes.

### Testing Stages:

The test plan was executed across multiple distinct stages:

* **Unit Testing (Blackbox/Whitebox):** Individual functions were subjected to rigorous unit testing, encompassing both black box and white box test data. The focus was on validating expected behaviours and uncovering potential bugs or logical errors.
* **Integration Testing:** Different functions were assessed for compatibility to ensure cohesive functionality. Rigorous testing was conducted to verify that the output aligned with specified business requirements.
* **End-to-End Testing:** The complete program was evaluated to ensure seamless execution from start to finish. This stage focused on validating the program's overall behaviour, including data flow and interactions between different modules.
* **Acceptance Testing:** The final stage involved validating that all specified business requirements were satisfactorily met. This step ensured that the program fulfilled its intended purposes and met user expectations.

### Test Design Process:

The design of each test followed a systematic process:

* **Function Specifications:** Detailed descriptions of functions, including purpose, unusual conditions, and parameters, were outlined.
* **Traceability Matrix:** A traceability matrix was established to link requirements to corresponding test cases, ensuring comprehensive coverage.
* **Test Case Preparation:** Blackbox testing formed the very first step in the development of test cases and followed the route mentioned in the Testing Stages section above. These cases encompassed various test data inputs and conditions.
* **Peer Review:** Test cases were subjected to a thorough review by quality assurance team members to validate accuracy and comprehensiveness.

The integration of these rigorous testing stages and processes aimed to validate the program's functionality, performance, and adherence to business requirements.

### Scope of Testing:

The scope of testing for the functions was strategically defined to ensure a comprehensive assessment while optimizing testing efforts. The black box test data was thoughtfully selected to encompass a wide array of scenarios, including varying conditions and potential unusual cases. The goal was to thoroughly test the functions using real-world scenarios to ensure reliability and robustness.

During the development process, functions were designed to address potential unusual conditions, ensuring that certain scenarios, that would typically trigger these conditions would never arise. This strategic approach allowed the development teams to pre-emptively handle potential anomalies, thus minimizing the need for extensive testing of those unusual conditions in subsequent milestones.

Given the thoughtful design, the scope of testing focused primarily on the core functionality of the functions. This approach allowed testers to concentrate their efforts on the main operational aspects of the software, ensuring that it met the central business requirements and objectives. This approach facilitated a thorough examination of the functions' primary tasks, enabling the testing team to provide higher assurance regarding the software's expected performance.

## **TESTING COVERAGE:**

Maximizing testing coverage of the program has been an important aspect of our quality assurance efforts. Throughout the testing procedure, we strived to thoroughly assess the code through a wide range of situations. Our approach to testing coverage involved a systematic strategy to ensure that critical code paths, functionalities, and edge cases were accurately examined. We employed a combination of unit, integration, and end-to-end tests to ensure maximum coverage of the software codebase.

As mentioned previously, the automated tests helped us ease the overall testing procedure, allowing us to concentrate on developing test cases and enabling us to systematically examine numerous scenarios. However, it's important to note that while our testing efforts were extensive, achieving 100% coverage may be difficult due to the complexity of the software. Nonetheless, our comprehensive testing methodology contributed significantly to validating the program's functionality, performance, and adherence to requirements.

# BUGS ENCOUNTERED/FIXES APPLIED

### BUG DETAILS AGAINST TEST RUNS:

The table below provides a summary of test runs conducted for various functions, along with the bugs we encountered during the testing process. Both black box (BB) and white box (WB) tests were executed, revealing instances where bugs were identified. The results helped in the identification of areas for potential improvement and refinement.

|  |  |  |  |
| --- | --- | --- | --- |
| **FUNCTION** | **TESTS RUN** | **BUGS ENCOUNTERED**  **(IF APPLICABLE)** | **PASS** |
| **validate** | BB\_validate | **YES** | **YES** |
| WB\_validate | **YES** | **YES** |
| **getSpaceRemaining** | BB\_getSpaceRemaining | **N/A** | **YES** |
| WB\_getSpaceRemaining | **N/A** | **YES** |
| **getTruckByReference** | BB\_getTruckByReference | **N/A** | **YES** |
| WB\_getTruckByReference | **N/A** | **YES** |
| **getTruckDistances2** | BB\_getTruckDistances2 | **N/A** | **YES** |
| WB\_getTruckDistances2 | **N/A** | **YES** |
| **sortByLimitingFactor** | BB\_sortByLimitingFactor | **N/A** | **YES** |
| WB\_sortByLimitingFactor | **N/A** | **YES** |
| **findTruckAndDiversion** | BB\_findTruckAndDiversion | **YES** | **YES** |
| WB\_findTruckAndDiversion | **YES** | **YES** |
| **integrateValidateAndGetTruckDistances2** | ValidateAndGetTruckDistances2 | **N/A** | **YES** |
| **integrateGetTruckByRefereceAndGetSpace** | GetTruckByRefereceAndGetSpace | **N/A** | **YES** |
| **integrateGetDistancesAndSortByLimitingFactor** | GetDistancesAndSort | **N/A** | **YES** |
| **integrateAllFunctions** | MOCK\_RUN\_ENTIRE\_PROGRAM | **N/A** | **YES** |

### DETAILS OF BUGS:

The following table outlines the specifics of bugs identified during testing and their subsequent resolution status.

|  |  |  |
| --- | --- | --- |
| **FUNCTION** | **BUG DESCRIPTION** | **FIXED?** |
| validate() | Validate function didn't filter out invalid destinations.  (i.e., non-building grids or unserviceable destinations) | **YES** |
| findTruckAndDiversion() | * The initial version of the function failed to correctly identify the diversion route for the cases within the project document. * The subsequent version got closer to the expected outcome but had additional unnecessary moves in the final route. | **YES** |

### DETAILS OF FIXES APPLIED:

Provided below are the specific fixes and enhancements applied to address the identified bugs and improve the software's functionality.

|  |  |
| --- | --- |
| **FUNCTION** | **FIX/ENHANCEMENT** |
| validate() | * The validate() function was modified to include stricter checks to ensure a valid destination was entered. * The fix ensured that the destination entered is a valid building. * To prevent further issues in the program an additional check was put in place that prevents the users from entering non-edge buildings (unserviceable locations, because of the path-finding logic). |
| findTruckAndDiversion() | * The main path-finding logic (contained in helper functions) was modified to match the expected outcome in the project document. * The final version of the developed function provides a better route than the expected diversion route for the final case mentioned in the project document, by allowing diagonal moves (everywhere along the route except for the point before the order delivery destination) |

# PROJECT EXPENDITURE

1. **Personnel Costs:**

|  |  |  |  |
| --- | --- | --- | --- |
| **MEMBER** | **ROLE(S)** | **COMPENSATION (CAD/HR)** | **WORK HOURS/WEEK** |
| Cesca Dela Cruz | Tester | 20 | 24 |
| Gulpreet Kaur | Developer | 20 | 24 |
| In Tae Chung | Tester | 20 | 24 |
| Irish Banga | Developer/Tester | 30 | 36 |

Total costs (7 weeks): **CAD 17,640**

1. **Hardware Costs:**

Total hardware equipment cost: **CAD 6,000**

1. **Software Costs:**
   * VMware Fusion 13 Pro Subscription: CAD $323/user \* 2 users = CAD 646 (one-time purchase)
   * Visual Studio Enterprise Subscription: CAD $303/user/month \* 4 users = CAD 1,212/month (billed monthly)
   * GitHub Enterprise Subscription: CAD $28.24/user/month \* 4 users = CAD 112.96/month (billed monthly)
   * Jira Premium Subscription: CAD $20.52/user/month \* 4 users = CAD 82.08/month (billed monthly)

Total costs over 7 weeks:

* + VMware Fusion: CAD 646
  + Visual Studio: CAD 1,212 \* 2= CAD 2,424
  + GitHub: CAD 112.96 \* 2 = CAD 225.92
  + Jira: CAD 82.08 \* 2 = CAD 164.16

Total software costs: CAD 646 + CAD 2,424 + CAD 225.92 + CAD 164.16= **CAD 3,460.08**

**TOTAL PROJECT COST: CAD 27,100.08**