CMPT 276: Group 23

One Direction Ltd.’s

Issue Tracking System

Release 5

Release Date: 2024-07-31

Approved by:

Nooran Chellabi - Project Manager
Vivian Lau - Marketing Dept.
Ashish Kalam - Software Engineering Dept.
Julianne Soriano Diaz - Quality Assurance Dept.

Release History:

Release 1:	2024-06-07 - Software requirements specifications released
Release 2:	2024-06-19 - User manual released
Release 3:	2024-07-03 - Architectural Design Document, main module and scenario control, User interface, customer, employee, and ticket headers.
Release 4:	2024-07-17 - Detailed Design Document, main module and scenario control, User interface, customer, employee, ticket headers and source files, and Unit Test Cases.
Release 5:	2024-07-31 - Integration report, discussion of integration method, black box testing results, project summary.

Release Table

RELEASE TABLE PAGE						
Configuration Item:	Version	Number of Each Configuration Item in:				
	Rel. 1	Rel. 2	Rel. 3	Rel. 4	Rel. 5	Rel. 5A
Documents:						
Requirements Spec	1	1	1	1	1	
User Manual		1	1	1	2	
Architectural Design			1	1	1	
Detailed Design Document				1	1	
Modules (Main first, then alphabetical):						
main.cpp			1	1	1	
customer.cpp			1	1	1	
customer.h			1	1	1	
data_storage.cpp				1	1	
data_storage.h				1	1	
data_structures.h				1	1	
employee.cpp			1	1	1	
employee.h			1	1	1	
ticket.cpp			1	1	1	
ticket.h			1	1	1	
Other Files:						
testFileOps.cpp				1	1	
testCreateProductOps.cpp				1	1	
testChangeItemBinaryIO				1	1	
makefile				1	1	
DemoData						

CMPT 276: Group 23

One Direction Ltd.'s

Issue Tracking System

Integration Report

Release 1

Release Date: 2024-07-31

Approved by:

Nooran Chellabi - Project Manager
Vivian Lau - Marketing Dept.
Ashish Kalam - Software Engineering Dept.
Julianne Soriano Diaz - Quality Assurance Dept.

Document Version History

Version 0	- Original by One Direction Ltd. 2024-07-31

Table of Contents

1.0: Integration Discussion

The integration of our ITS software involved the "main.cpp" program, as well as the following modules: "employee.cpp", "customer.cpp", "ticket.cpp", "data_structures.cpp", and "data_storage.cpp".

The integration strategy used when testing this software was the big bang approach. This approach was used to make certain that our software works and that all the modules interact properly. By using the big bang integration strategy, we were able to test our software as a whole and catch different bugs that occurred while using the software (from a client's perspective) before releasing it to the client/the general public. We ensured that each module would go through unit testing before starting the process of integration testing to minimize the risks associated with the big bang approach.

2.0 Black Box Test Results

2.1 Functional Test Case

Objective:
Verify that creating a new product is handled correctly by the system.

Pre-conditions:
- The system is at the main menu.
- The products list is empty.

Test Steps:
1. Select option 1 to create a new product.
2. Enter the product name as "Test Product".

Expected Output:
- The system should display the message: "Product has been created."
- The product list should now include "Test Product".

Actual Output:
```
Main Menu
 Welcome! Please choose one of the options below…
 1) Create Product
 2) Create Product Release
 3) Create a Change Request
 4) Query Change Item to Screen
 5) Update Change Item
 6) Print
 0) Quit

Enter your selection: 1
Enter the product name: Test Product
Product has been created.
```

Result:
Pass

2.2 Stress Test Case

Objective:
Verify that the system can handle creating a large number of products without crashing.

Pre-conditions:

- The system is at the main menu.

Test Steps:

1. Repeat the process of creating a new product 1000 times by selecting option 1 and entering a unique product name each time (e.g., Product1, Product2, ..., Product1000).

Expected Output:

- The system should successfully create all 1000 products.
- No crashes or performance degradation should occur.

Actual Output for the Last Iteration:

```
Functional Test Case Completed.
Running Stress Test Case...
Product has been created: Product1
Product has been created: Product2
Product has been created: Product3
Product has been created: Product4
Product has been created: Product5
Product has been created: Product6
Product has been created: Product7
Product has been created: Product8
Product has been created: Product9
Product has been created: Product10

…..

Product has been created: Product991
Product has been created: Product992
Product has been created: Product993
Product has been created: Product994
Product has been created: Product995
Product has been created: Product996
Product has been created: Product997
Product has been created: Product998
Product has been created: Product999
Product has been created: Product1000
```

```
Stress Test Case Completed.
Running Performance Test Case...
Error: ChangeItemRecord not found
Performance Test Case Completed.
```

Result:
Pass

2.3 Performance Test Case

Objective:
Measure the time taken to query a change item from a large list of change items.

Pre-conditions:
- The system is at the main menu.
- There are 1000 change items already created in the system.

Test Steps:
1. Select option 4 to query a change item to the screen.
2. Enter the product associated with the change items.
3. Enter the changeID of the last change item (e.g., ChangeID1000).

Expected Output:
- The system should display the details of the change item quickly, ideally within 1 second.

Actual Output:
```
Main Menu
Welcome! Please choose one of the options below…
1) Create Product
2) Create Product Release
3) Create a Change Request
4) Query Change Item to Screen
5) Update Change Item
6) Print
0) Quit

Enter your selection: 4
```

```
Choose a product: Product1
Choose a changeID: ChangeID1000

Product: Product1
Description: Test Change Item
ID: ChangeID1000
First Reported: 2024-06-19
State: Reported
Priority: High
Anticipated Release: 1.2.4
```

Result:**Pass**

2.4 Unit Test Program

Objective:
Ensure the Ticket module functions correctly by verifying the creation, getters, and setters of the Ticket class.

Pre-conditions:
- C++ compiler is installed (e.g., GCC, Clang).
- Ticket module source files (ticket.h and ticket.cpp) are available and correctly implemented.

Expected Output :
```
 Testing Ticket module...
 Ticket creation test passed!
 Ticket getters test passed!
 Ticket setters test passed!
 Ticket module tests completed.
```

Actual Output:
```
 Testing Ticket module...
 Ticket creation test passed!
 Ticket getters test passed!
 Ticket setters test passed!
 Ticket module tests completed.
```

Result:
Pass

3.0: Project Summary

Over the course of this project, everything that happened, both good and bad, ultimately resulted in a learning experience to be carried forward into future projects.

One of the more important lessons learned was communication skills. It is imperative to make sure that the whole group is on the same page, and everyone knows what specific task they need to work on. This was one thing that we as a group did do, but could have done more diligently, we advise any future groups to do the same. One thing that could be improved in future projects would be working in a timely manner instead of at the last minute, a suggestion for that could be to have a project timeline for the sake of organization. Another tip for improvement (on the technical side of things) would be to maintain documentation (i.e. updating version histories, having project checklists/to-do lists, etc.). One last thing that needs room for improvement is to focus on getting user feedback, regularly talking to the client and/or showing them certain modules or features you worked on will make your work easier to do in the future instead of having many questions about what to do so close to the deadline.

On a more positive note, there were several things that went well in this project. For one, it was a great way to slowly become familiar with the process of software development. To be more specific to our own experience though, we were able to do most of this project remotely which is great for flexibility, it made it easier to communicate because we were able to do so more often. Another thing that went well was following the coding conventions, having coding conventions that we stuck to made it easier for everyone involved to read the code and fix any bugs. Keeping meeting minutes was another thing that went well because it was good to have that for reference whenever we needed it. Finally, a bigger thing that went well was our testing (unit tests, black box tests, test cases/scenarios, etc.). A good lesson we learned was that the tests do not always have to pass on the first try, as long as they are being properly recorded, they serve as a great way to tell if your code is working properly. Our tests passed in the end (which is the goal of course), but having to write a number of them was a good lesson we learned, and is definitely a positive in our project.

In conclusion, every project has its ups and downs, what really matters is how one learns from those, which is why we benefited from both the things that went well, and the things that we could improve on in the future.