SENG3011
Testing Documentation

Group Number: 02

Version	1.0
Print Date	16/05/2013 2:11am
Release Date	16/05/2013
Release State	Final
Approval State	Pending
Approved by	Group02
Prepared by	Group02
Reviewed by	Group02
Confidentiality Category	Confidential

Document Revision Control

Version	Date	Authors	Summary of Changes
v1.0	16/05/2013	Group02	Added in Introduction
v1.1	16/05/2013	Group02	Added in Architectural
v1.2	16/05/2013	Group02	Added in Testing Environment
v1.3	16/05/2013	Group02	Added in Overview of Test Data
v1.3	16/05/2013	Group02	Added in Illustration of Testing

Contents

1 Introduction **1**

2 Architecture **1**

3 Testing Environment **1**

4 Overview of Test Data **2**

5 Illustration of Testing **3**

1 Introduction

This report is designed to discuss the testing which we developed and used alongside our system.

Most tests were designed while we created our system. Other tests were created when we completed specific milestones.

2 Architecture

This section discusses our current Architecture of the system and which parts of it are being tested. We have also included whether or not the tests were functional or non functional.

The Trade Signal generator has been tested in a number of ways. Functional testing of the trade signal generator was ran with input files and checked if the output was the one we expected from our calculations. A number of a unit testing occurred on the trade signal generator which all passed. There was also non-functional testing performed on the trade signal generator attempting to speed the generator up.

The Trade Engine functional testing occurred in which the input was given and the output was compared to what we expected it to provide. In this the output matched we expected to occurred. The Engine was also unit tested which passed all the unit testing.

The trade Strategy Evaluator was also functionally tested in which we gave the evaluator input and the output was compared to what we expected the output to be, where it matched expected results. The strategy evaluator was also unit tested in which the test results matched with what we expected.

Console UI is a non-functional part of our system and required manual testing. We would pass the Console UI to a team member who didnt code the Console UI to see if he could follow it. He was able operate the GUI easily hence the user interface was designed well enough for someone who did not see the code to understand.

The Graphical UI was tested by showing a non-team member our UI and seeing if he could understand our UI. He was able to use the UI and see what to do where and how. This is how we tested that our graphical UI was easy to follow for a non-team member, thus would be easy for anyone to use it.

- Trade Signal Generator: Functional testing, unit testing, non-functional testing (attempting to speed up)
- Trade Engine: Functional testing, unit testing.
- Trade Strategy Evaluator: Functional testing, unit testing.
- Console UI: Non-functional testing, manual testing.
- Graphical UI: Non-functional testing, manual testing.

3 Testing Environment

This section contains the description of our testing environment. We have used JUnit testing and user testing .

- JUnit test suite:

We use the built in JUnit testing suite found inside eclipse to test individual parts of our

code. We wrote these test cases as we developed our system to make sure our system was performing as we expected it to.

- User testing:

  User testing was required during the creation of our GUI and our system, which occurred during the addition of new parts. User testing also helps verify with live test data that our system is adhering to the requirements provided.

- We do not test external libraries (CSV Parser):

  This has not been tested, however we have seen that it correctly parses in the information during the initial creation of our program and during the development process. We also do not explicitly test external libraries as we trust that it is a complete and well-developed product.

- We do not programmatically test GUI or UI:

  We only test our GUI and UI via user testing and in no such way do we test it programmatically. This is a limitation in our testing environment where we cannot test this.

4 Overview of Test Data

We provide an overview of our test data in this section to provide some clarity to the users of our system.

- We have test cases for each part of our program:
  - Trade strategy evaluator:
    - Tests if the evaluator reads in a trade list correctly.
    - Tests if profit breaks even.
    - Tests if profit calculation is correct with 1 buy and 2 sells.
    - Tests if profit calculation is correct with 2 buy and 1 sells.
  - Order book implementation:
    - Tests the Random Strategy by checking if the volume for the generated Bids and Asks orders are the same. It also checks whether list provided after running the strategy is not empty.
    - Tests the Momentum Strategy for a positive trend. It also checks whether list provided
      after running the strategy is not empty. We also check to make sure that a Bid has been created to match the positive trend.
    - Tests the Momentum Strategy for a Negative trend. It also checks whether list provided after running the strategy is not empty. We also check to make sure that an Ask has been created to match the Negative trend.
    - Tests the Mean Revision Strategy for a Positive trend. It also checks whether list provided after running the strategy is not empty. We also check to make sure that an Ask has been created to match the positive trend.
    - Tests the Mean Revision Strategy for a Negative trend. It also checks whether list provided after running the strategy is not empty. We also check to make sure that a Bid has been created to match the negative trend.
  - Algorithmic trades:
    - Test the algorithmic trader with a lower ask volume than buy volume and see if it still works.
    - Test the algorithmic trader with a lower buy volume than ask volume and see if it still works.
    - Test the algorithmic trader with the exact same buy and ask volumes and see if it still works.
  - UI and GUI Testing:
    - Run with the initial SIRCA input provided.

- Run with Random Strategy, check one bid and one ask is generated.
- Run with Momentum Strategy, check bid and ask orders are generated.
- Run with Mean Reversion Strategy, check bid and ask orders are generated.
- Attempt to quit via button.
- Check performance reports are correct.
- Attempt to load new SIRCA data.
- Repeat.

5 Illustration of Testing

Whenever a new feature is added to the system, unit tests are written soon after to make sure the feature operates correctly according to our requirements documents. Following writing unit tests for the feature that is just written, all tests must be run on top of that to ensure no other features are broken. We use JUnit for Eclipse to make this process much faster, a simple run button and a graphical display instantly tells us which tests pass and fail.

After all programmatic testing is passed, at least one of the two available interfaces must be run to ensure that the feature works functionally. We run it with live data (actual SIRCA data) and run whichever feature and check if it outputs what we expect given specific inputs. This form of manual testing takes a lot more time compared to automated unit tests. To ensure functional sanity, we get someone else (anyone but the person who wrote the code) to confirm that the feature recently written is both descriptive, accurate and informative for the user.