**Brief on Quality Assurance Team 1**

**Acceptance Tests**

During this sprint our team hoped to improve upon our performance from the last sprint by tackling issues brought up during that sprint, one of which was the issue of acceptance testing. By using acceptance tests our team was able to split up user stories into easy to check tasks which when brought together would reflect each aspect of the user story. Previously we had been slightly vague with our acceptance tests and had only met with the client once during confirmation of these tests. During our sprint review with the client for the previous sprint some fine details were discovered to be not what the client has envisioned, specifically the font, UI layout, button size etc.

We resolved to ensure that the highest quality of our product was delivered this time, and thus we set about resolving the two major issues with the previous sprint’s acceptance tests. This time after gathering initial acceptance tests in discussion with the customer the team went back to refine and broaden these tests into fully fledged, specific and testable acceptance tests. After this process of refinement and brainstorming was completed we returned to the client with the new acceptance tests to talk them through the exact features for each story to be developed. This second meeting helped clarify points even more, and as both parties had more time to think on the project, more features pertaining to error handling were discovered. Again these new acceptance tests were further refined and broadened by the team, which at this point felt that the features of the user story were being exhausted.

These practices built upon lessons learned at the end of sprint one. By preforming acceptance test discussion and development in this fashion we strongly believe that this sprint’s deliverables are of a much higher quality and truly reflect the client’s vision.

(The acceptance tests and their respective user stories are contained in the User stories and Testing Examples section).

**Refactoring**

Another new feature of this sprint was the extensive use of refactoring in order to adapt our code structure more fluidly and quickly to any changing needs of the client. After looking through materials given out through the lecture it was decided that one person take on the role of refactoring all the code in order to maintain a single style of refactoring. That style was determined by looking through online Open Source code which by its very nature has to be extremely friendly and intuitive to all parties involved. Refactoring was essential for the unit testing which occurred in our code, it helped provide a solid understanding of where other team members had gotten to and there process for getting there, incidentally this incremental refactoring effort identified relic code before unit testing for code coverage, which was swiftly removed. By providing a clearer foundation into the code base through changing method and variable names, removing dead code, and commenting in useful places like at the end of any statement, the team was able to proceed much swifter with fewer code duplicates for tasks being written due to the team work nature of the project. In these respects the code given to the client for their app will definitely have fewer bugs and provide a solid intellectual foundation for the next sprint.

It is important to note that during this refactoring process working code was left untouched, refactoring through our understanding should never change the outcome of the unit tests assigned to code.

An example of refactoring in a class:

/\*ShareSetsModel - Creates a new Array of ShareSet objects, populates their entries with hard coded information ie

\* current number of stocks in a company, the company's share ticker, and company name.Uses the information of the ShareSet objects to calculate and display the relevant information back to the user.

\* Methods Used:

3.0 calculateCurrentPortfolio - Determines the total value of the current portfolio.

3.1 calculateFridayPortfolio - Retrieves the previous stock close value of last friday @ 5pm.

3.2 calculateLossGain - Determines the total lost/gained over the entire portfolio by comparing current value against stock close of previous week's value.

3.3 calculateShareTotals - Calculates the total value of each share and inserts this new value into a list.

3.4 detectPlummetRocket - detects is a paticular stock is on a run or fall which is defined as a share increasing in 10% value over a week. Or if a stock value has decreased by 5% over a week.

3.5 roundDouble - rounds the value gathered from the Yahoo API to 2 decimal places.

\*/

(This has been reformatted to look appropriate in word)

These brief class descriptions and methods used within the class along with their rationale and descriptions were placed in every class. This helps us to ensure that there is only a single responsibility for each class, helping to increase class cohesion.

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Start of calculateCurrentPortfolio 3.0\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*End of calculateCurrentPortfolio 3.0\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

Before and after each method within the class these headers and footers were used to clarify when methods and sub-methods used start and end. There is nothing worse than looking through code trying to find a method in a large class, each method is given a number as shown above and using a search for that number will bring those who are reviewing the code to that particular class, increasing the ability to rapidly maintain/update the method.

/\*

\* calculateShareTotals - Calculates the total value of each share and inserts this new value into a list.

\* @param - none.

\* @return - the value for each share in the list.

Each method was given a direct explanation of its role within the class, helping to ensure high class cohesion. These include the returning parameter and parameter’s taken by the method to clarify how each class relates to each other (This was shown through metrics as well, but this manual method allows those developing the code to have instant knowledge of the required parameters for that code to function as intended.)

Not all refactoring included adding comments to the code, other refactoring processes included standardising the style of variable names which was not universal between the team before the sprint. All variable names were changed so the second word was capitalised ie. **thisWord**. Also by determining the unique individual role of each class we were able to identify much relic code. This code having no function any longer posed a serious risk of causing errors to the program should it be activated somehow, within our code a lot of obsolete prototype code from learning within the eclipse environment was removed.

The overall design was changed by this process as well; initially the Yahoo API fetch for the data was contained within the activity classes in our code. But this meant classes shared a common responsibility which was called by each activity, by refactoring this code out into its own class we were able to increase these activity classes cohesion through the sole responsibility rule.

As for examples to the progressive usage of refactoring for removing relic code; if you go back 5 commits on gitHub there will be prototype code in the Yahoo API class which pertained to testing during sprint 1. Much of the revised commenting for the code was done post- development of that code, however the moving and changing of names can be seen as early as November 5th on gitHub with Calum Murray removing relic code, the change to a standardised format for field names occurred during sprint 1 and should be visible in change between code from before the 5th of October 2012.

**Unit Testing**

Through unit testing using djunit the objective of any project is to identify relic code for removal, determine errors which may occur in hypothetical circumstances which may indeed become reality in the future. Unit testing provides a basis for testing acceptable data passed to variables as well as providing a simulation of how your code will respond should invalid entries be passed to data which may not be logical ie. If a stock price is returned as a negative number, how will the code respond?

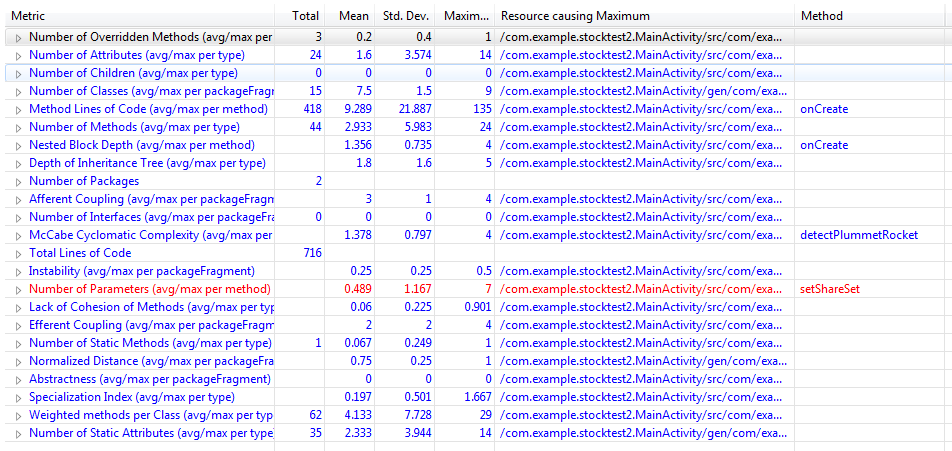
Our team used both black box testing and white box testing approaches during this sprint, white box when we already had working code and wanted to ensure that it was as error free as possible, and black box testing when developing new code to be implemented ie. We didn’t know what code to test upon and so the tests came first.

The software being used for unit testing did leave a lot of gaps which were uncovered during the team discussions on suitable testing; no android elements could be tested using the junits tests eg. What would happen if the client swiped a button instead of pressing it, would it still function? These had to be done manually, with the most common case being that an error message would be displayed should this case occur.

Through unit testing we were able to ensure that the client would be able to operate their application in the event that there would be no internet connection, no stock data, invalid stock data or incomplete stock data. Because these situations are taken care of the user experience is enhanced and thus the quality of our work could be assured.

**Metrics**

After preforming unit testing our group turned to the issue of using metrics to decrease the complexity of our code and attempt to increase code coverage by removing any (very few at this point) relic code. The code coverage percentage agreed by the group was 90% with our actual code coverage was 92% which was above reasonable expectation from the group. The primary metrics used and agreed on by our team was to increase the maintainability Index of our code and decrease as much as possible the cyclomatic complexity (number of branches within the code) as well as depth of inheritance, and lack of cohesion methods. By ensuring that our code was of low complexity and ensuring that each class has a unique responsibility we increased the re-usability and ability to grow our code over time.



(Overview of metrics using JUnit)

Blue indicates that a metric used is within acceptable bounds, while red indicates a metric which may cause issues. The single red entry points to our setShareSet class, this class has the responsibility for holding each of the parameters a set of shares may have i.e. Current value, historical data, current trading quantity etc. Having the number of parameters in the red due to this class is therefore acceptable as the class as defined by the team has the sole responsibility to hold all specifics of a share set. Of note here is the cyclomatic complexity at 1.378 branches on average per class which being below 5 is a very reasonable number, reducing the overall number of avenues our code can take when being run. Some more metrics to which the team paid special attention to are as follows:

The depth of inheritance indicates how many classes a class is inheriting from and thus depending on, while in the blue for this project it does have a maximum number of 5. This is because we have over-written the behaviour of Activity, by doing so we have had to extend it. By doing this we have required the use of several libraries by the class, which it inherits from. So from the perspective of the team this is acceptable.

Lack of cohesion Methods (LCOM) is a measure of how many of the instantiated fields of a class are used by each method. A perfect score would only occur if each field is used by all methods within a class, which is an unreasonable assumption in the reality of coding; a value between 0 and 1 is considered to be acceptable bounds of which we achieved a score of 0.06 with a maximum value of 0.901.

Weighted Methods per Class indicates the complexity of each method in a class which contributes to the overall complexity of the class in general. This was reduced in our code by splitting the Yahoo API into its own class, a class with a large number in this indicates it could probably be split up into two or more classes.

**Sprint Velocity**

From our last sprint we decided that our sprint velocity, the units of effort, that the team was capable of performing during this sprint being 9 effort units, however upon reflection and group discussion this sprint we decide to decrease this velocity to 7. This was due to the fact that we were unable to successfully complete one user story last spring, proving that 9 effort units was far too much for the team to confidently complete in one sprint. The reduction to 7 was decided because as a team we were able to complete the major story of last sprint and was near to completing the other user story, therefore only a small reduction in the sprint velocity was determined to be needed.

**User Stories and Testing Examples**

**Share Set Worth**

As a user,

I want to be able to see how much each set of shares is worth.

Rationale:

To manage my finances better and to get a clearer picture of how much money I have in shares of a particular company and whether they are worth keeping.

Priority: 87

Effort: 3

**Share set worth**

* The share set’s worth shall be accessed by pressing a button, on the main menu, of approximately 3 cm diameter.
* This value shown shall be live (with a 15minute delay from yahoo finance) for each share set displayed.
* The value displayed for each share shall be in Pounds and Pence to 2 decimal places.
* The value displayed for each share set shall be taken from the Last Trade Price on yahoo finance.
* The format of this display to the customer shall be in a list/table in the format:
  + “*Company name* *Shares Owned* *Price(£.pp)”*
* Each share set shall have its own list/table entry for the current worth of that set. They shall descend in alphabetical order.
* A “No feed is currently available” message shall be posted to the screen if no internet connection is available.
* If an individual set is unavailable from the yahoo API the following message shall be displayed for the price: “*N/A*”
* There shall be a 10 second upper time constraint on the time between pressing the button on the main menu and the display of the share set worth.

**Total portfolio Worth**

I want to be able to see how much all my shares is worth in total.

Rationale:

To manage my finances better and to have a clearer picture of how much money I have in shares.

Priority: 100

Effort: 2

**Total Worth**

* Record the total sum value of all portfolio shares, in Pounds Sterling, of the previous week’s Friday close values.
* Access shall be provided to this display through the main menu, which will have its own button.
* This total value shall be presented in sentence form eg.
  + “Your total portfolio is worth £1000 as of *DD/MM/YY*”
* The font size of the display message shall be size 20.
* Provide a check that the current price displayed is correct for verification. This will be achieved by showing the current stock price of shares on a laptop during a meeting.
* The font to be used on the display shall be a “sans serif” typeface.
* A “No feed is currently available” message shall be posted to the screen if no internet connection is available.
* If there is no data available for any individual share this error shall be presented to the screen in the format “There is not enough data available to compute portfolio value”.
* There shall be a 10 second upper time constraint on the time between pressing the button on the main menu and the display of the total portfolio worth.

**Portfolio Value Lost/Gained**

As a user,

I want to know how much money I have lost/gained in a week.

Rationale:

To manage my finances better and to have a clearer picture of how much money I have invested in shares and whether they investment was profitable or not.

Priority: 90

Effort: 2

Acceptance Test for this story:

**Money Lost and Gained per Week**

* The option to view the Total portfolio’s gain/loss per week shall be accessible as its own button from the main menu interface.
* The total value lost/gained of the portfolio shall be determined by comparing the value of all share sets with the value of the previous week at trading close on Friday.
* This value shall be presented in pounds only with no pence.
* The loss/gain for the portfolio shall be presented in green text for a gain and red text for a loss.
* A textual sentence statement of the format: “For the week of *DD/MM/YY* you have gained *£ AMOUNT GAINED*” shall be displayed to the customer.
* For a loss the textual statement shall read “For the week of *DD/MM/YY* you have lost *£ AMOUNT LOST*”.
* The *DD/MM/YY* of the textual statement shall be the date of the previous Friday (The current trading week being compared against).
* A “No feed is currently available” message shall be posted to the screen if no internet connection is available.
* If there is no data available for any individual share this error shall be presented to the screen in the format “There is not enough data available to compute how much has been lost/gained”.
* The font to be used displaying messages shall be aerial.
* There shall be a 10 second upper time constraint on the time between pressing the button on the main menu and the display of the money lost and gained in a week.

**Unit Testing For Sprint Two**

Test Portfolio Class Tests:

I have split these into their own files on word which has been submitted with the assignment. Also I have provided a paper copy of the code, as well as the original code used for our testing in their .JAVA format.

Screenshot of tests passing on next page. Due to the low visibility in the word document I have provided this image separately which you will be able to zoom in and out of on windows preview.

