



Software Engineering Assignment

Student: Dean Beckerton (BEC12375372)

Lecturer: Dr J Murray

Module: CMP3111M

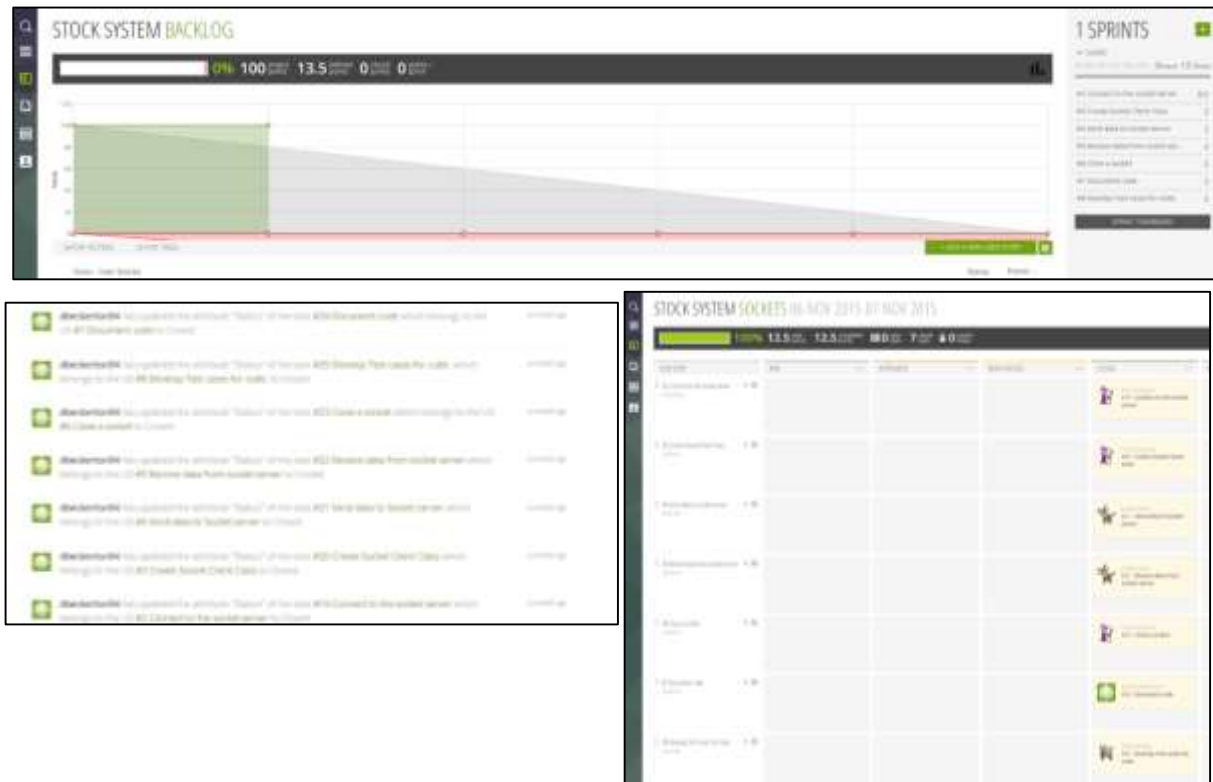
Contents Page

Title Page	1
Contents	2
Personal Sprint Log.....	(3 to 7)
Design Patterns	7
Critical Reflection.....	(7 to 11)
Alternative software development methodologies.....	(11 to 12)
Critical Evaluation.....	(12 to 13)
References.....	(13 to 14)
Appendix.....	(15 to 43)

1.0 Personal Sprint Log

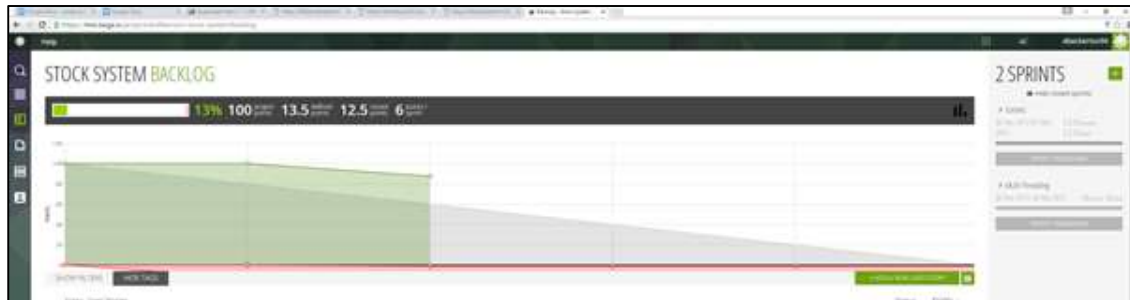
For this assignment an artefact was developed using the Scrum methodology, below are a number of Sprint logs created within the development of this artefact. The Sprint is a specific period of time where set tasks have to be completed under strict time conditions, these tasks are allocated to specific team members within the Scrum team. For this project sprints were created and contained within an online project management platform known as Taiga. Below is evidence of the Sprint logs, although further evidence can be found within Appendix 1.6.

Sprint One - Stock System Sockets (06 November – 07 November)



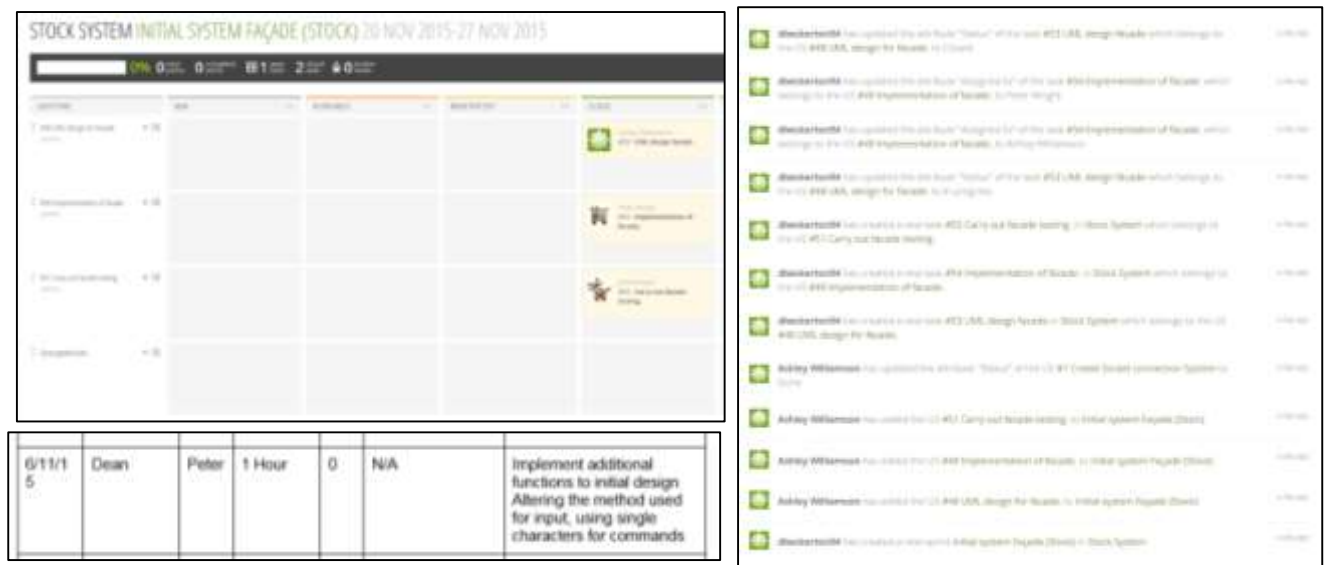
This first Sprint was started within the second week of workshops. Within this Sprint I was assigned the role of Scrum master, this role involved a number of different tasks. Firstly this involved keeping the group away from any external distractions, this was arguably quite easy to implement as this only applied through workshops were the team were already dedicated to completing this set work and not concentrating on other work or external factors that may affect their performance. Secondly being Scrum master also involved using the Taiga software to firstly create the Sprint followed by creating additional user stores from which team members were then allocated certain tasks to complete, this can be evidenced by the screenshots above using the Taiga history settings while also showing the created tasks in the screenshots above and the Sprint start. After this sprint had been completed the artefact now had the ability to connect to the Client server through the command window.

1.1. Sprint Two – Multithreading (6 November – 20 November)



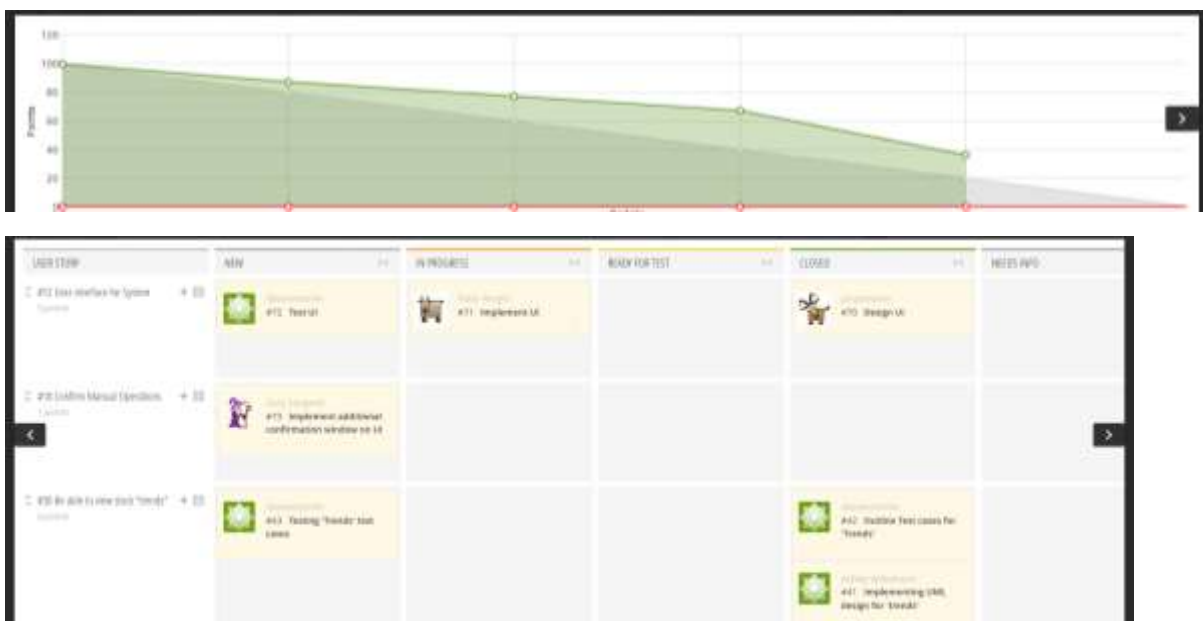
The second Sprint was also started within the second week of workshops but also carried on till third week. Within this Sprint I was assigned the role of testing. This role involved completing a number of different tasks. Firstly this involved outlining a number of different test cases that would be required to be implemented to allow for a full test of the artefacts functionality, these test cases were outlined and stored within the group Google drive (See Appendix 1.4) The test cases were then implemented to the existing code, these particular test cases involved implementing a number of different write lines throughout the code to test whether the program is functioning as the Client intended and as the design specified, for this case this involved establishing a connection by sending a message to the server and also retrieving a response (See Appendix 1.4). After this Sprint was completed the artefact now had the ability to connect to the Server while also having the ability to both send and receive messages to and from the Server.

1.3 Sprint Three - Initial system Façade (20 November – 27 November)

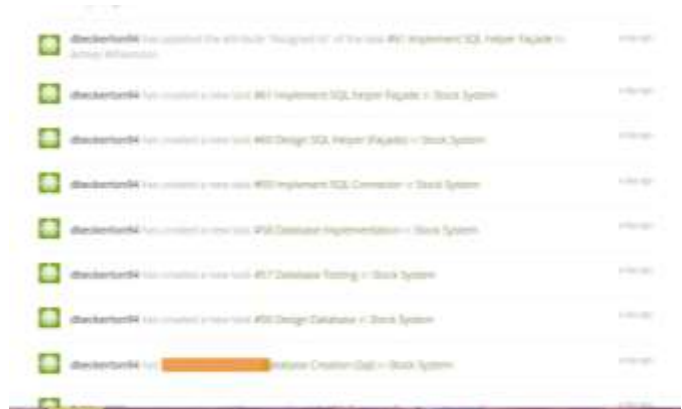


For the third Sprint I was assigned the position of programmer, this involved implementing the Façade alongside another team member this meant that we worked under the methodology of pair programming (See Appendix 1.1). For this Sprint task the programming involved creating an initial Façade, this meant that during the pair programming process we had to implement a method to combine all the functions of both sub components and systems into an outward facing, usable, class for the Client. Additionally within this Sprint I also created and managed the Taiga Sprint as seen in the screenshot above, this involved creating the initial Sprint and user stories while also assigning the tasks to each team member that was completing the set Sprint tasks. Once the Sprint was completed the artefact now had an outward facing, usable class for the client meaning the main elements left to developed for the artefact included a database and user interface.

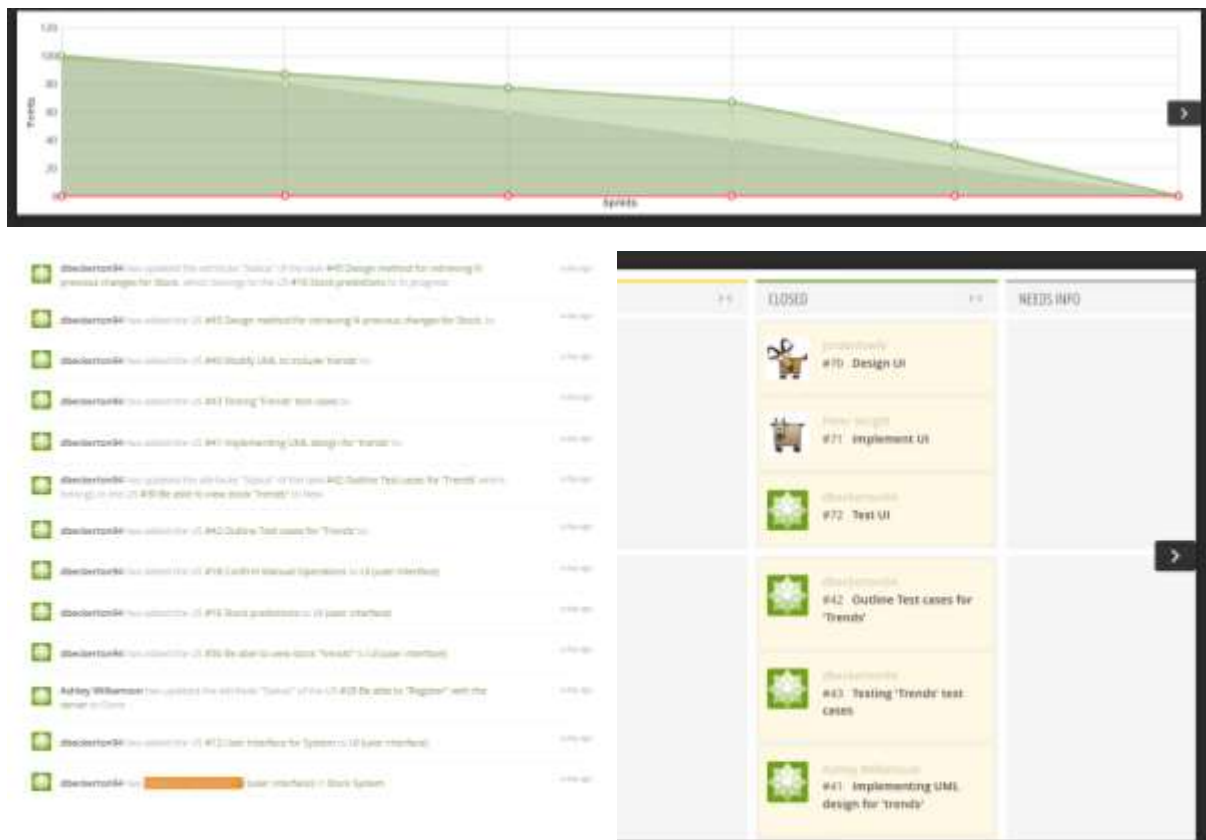
1.4 Sprint Four – Database creation (27 November – 04 December)



For the Fourth Sprint I was initially assigned to complete the testing of the Database, although I also was later also assigned the task of Designer to the Database (See appendix 1.6) as members initially assigned the task of designing the database did not attend. The first task I had to complete was to design the database, this involved using Unified modelling language (UML) diagrams to implement a database design that could then be looked at and implemented to the system (See Appendix 1.2) The next task I had to complete was the testing of the Database, this involved inputting variables into the database, this would then allow for any unexpected errors to be throwing back out the Console window if the Database was not functioning correctly for the Client (See Appendix 1.3). Lastly within this Sprint I also created and managed the Taiga Sprint as seen in the screenshot above this again involved creating the initial Sprint and user stories while also assigning the tasks to each team member. After this Sprint was completed the artefact was pretty much complete all the artefact now lacks is a UI, therefore the artefact now functions almost identically to the user requirements.



1.2. Sprint Five – UI (user interface) (04 December – 11 December)



For the fifth Sprint I was assigned the position of testing the main user interface of the artefact. The first task completed involved outlining the initial test cases for trends within the UI, this again was inputted and saved within the group's Google drive area (See Appendix 1.4). Secondly the test cases

were then implemented to the user interface, this involved testing basic aspects of the UI such as clicking buttons and completing tasks the user would be completing when using the artefact while also testing other aspects such as spam clicking areas or attempting to complete tasks the user should not be completing such as buying negative stocks. Lastly I was also responsible for Taiga within this sprint as seen in the screenshot above this involved creating the initial sprint and user stories while also assigning the tasks to each team member. After this Sprint was completed the artefact now includes a functional user interface and additionally all the Clients required functions to make this artefact a fully deliverable and usable piece of software.

2.0 Design patterns

Throughout the implementation of the artefact a number of different design patterns were used “a design pattern is a general repeatable solution to a commonly occurring problem in software design” (SourceMaking, 2015) these design patterns allowed for a number of different benefits once applied to the artefact.

2.1 Façade Pattern

One key design pattern implemented within the development of the artefact was the Façade pattern “The Facade defines a unified, higher level interface to a subsystem, that makes it easier to use” (AG Communication Systems, 1999) There are a number of benefits of implementing this design pattern. Firstly it is an easy pattern to implement. Secondly it allowed for the ability to combine all the complex methods together through one single Façade class and therefore provides for a much more structured environment for the artefact.

2.2 Singleton Pattern

Another key design implemented within the development of the artefact was the Singleton pattern “a singleton is a class which only allows a single instance of itself to be created, and usually gives simple access to that instance” (Yoda, 2006) In terms of the artefact the use of this pattern was implemented so that only one instance of the class can ever be created, in relation to this artefact the ‘one instance’ involves the use of a server connection.

2.3 Model View Controller (MVC)

The last main design pattern implemented within the development of the artefact was the MVC pattern “The model is the data, the view is the window on the screen, and the controller is the glue between the two taking the data and presenting that to the view.” (K Hong, 2015) In relation the implementation of this design pattern, it was used to combine all features of the artefact (console window, buttons, display) this method was implemented to provide for the best user interface possible for the user (See Appendix 1.2)

3.0 Critical Reflection

Within this assignment the development of this artefact involved the use of the Scrum methodology “Scrum defines a minimal set of process elements that make up a Sprint. Scrum defines three roles (roles), three activities (ceremonies) and three types of results (artifacts).” (Foegen, 2010) Scrum proved both useful and also obstructive within the process of the development of the artefact for a number of reasons. Scrum is broken down below into its main elements and explained on its impact to the artefact below.

3.1 Scrum Client

In terms of a critical reflection of the scrum methodology the first piece of reflection that can be discussed is the artefact in relevance to its product owner. Within this assignment the product owner (client) was very useful within the scrum methodology and provided a good line of communication, through both verbal and written delivery while also being regularly available to answer any required questions, this allowed for the artefacts development to be much more defined and precise allowing our team to know what was required within the artefact to meet the user's requirements (See Appendix 1.3)

Although as a reflection the point could be made that within some sprints the client proved obstructive to the Scrum methodology, this falls in respects to the team who when working on specific tasks the client sometimes upset this process through verbal communication (See Appendix 1.5) causing an impact upon the artefact in the form of a slower development.

3.2 Scrum master

Another point to discuss within the critical reflection of the scrum methodology is the use of the scrum master "the ScrumMaster is responsible for making sure a Scrum team lives by the values and practices of Scrum" (Mountain Goat, 2015) within this assignment the Scrum master role seemed beneficial towards the development of the artefact in a quite a few ways. The Scrum master was responsible for the online project management platform (Taiga) this had a beneficial impact upon the artefacts development due to the reduction of work load for the team members while additionally this also allowed for the organisation of Sprints and tasks the team needed to achieve, thus this meant the Scrum master helped remove obstructions that could prevent the team from achieving its sprint goals (See appendix 1.6).

Although the Scrum master role did sometimes prove obstructive to the development of the artefact, one reason being due to one less member completing tasks within Sprints. Furthermore the Scrum masters role sometimes could not be achieved due to the level of distributions being too hard to obstruct causing a particular impact upon the artifacts development (See Appendix 1.5) "interruptions are not related to the team's current work and need to be blocked by the ScrumMaster so that the team will be able to focus on its current goal: the Sprint and its Product Backlog Items." (Berteig, 2014) Thus this ultimately lead to the reduction of productivity for development of the artefact throughout small periods of the Scrum Sprints. As an overall reflection of the Scrum master role it may be more beneficial for the Scrum master to have dedicated more time to the completion of certain tasks alongside the team, as this could have provided for a quicker development process for the artefact under the Scrum methodology.

3.3 Sprint planning meeting

Another important part of Scrum methodology is the Sprint planning meetings "time-boxed event of 30 days, or less, that serves as a container for the other Scrum events and activities. Sprints are done consecutively, without intermediate gaps." (Scrum, 2015) Within this assignment a Sprint planning meeting method proved quite useful to the artefact as it lead to a number of different impacts. Firstly it allowed for a greater level of understanding and communication within the team, this allowed for the greater capacity to all work together during the Sprint and support each other in order to complete individual tasks, it also made sure all team members participated accordingly during the Sprint (See Appendix 1.5) , overall this lead to a development impact in the form of

quicker Sprint development while also again allowing for the production of an artefact of higher quality due to all team members being used as effectively and efficiently as possible.

Meanwhile on the opposing side of impacts it could also be argued that the Sprint planning meetings were a hindrance and also had a negative impact upon the development of the artefact, one reason for this was due to some Sprint meetings team members set themselves non-challenging goals (See Appendix 1.5) this impacted the artefact as some of the team members were not working to an efficient level meaning the artefacts overall development was slower within some Sprints compared to others. In terms of a reflective stand point for future sprints, users should set themselves realistic but also challenging goals to allow for a better level of productivity towards the artefact.

3.4 Product backlog

Another part of the Scrum methodology is the product backlog “The product backlog is a list of functional and non-functional requirements that, when turned into functionality, will deliver this vision, the product backlog is prioritised so that the items most likely to generate value are top priority” (Schwaber, 2014) Within the development of the artefact the product backlog had a number of different positive impacts to the development of the artefact and proved useful in many different aspects. Firstly it allowed for the better ability to develop an artefact that would meet as many of its user requirements as possible (See Appendix 1.3) this impacted the development of the artefact as it allowed for the team to implement the most important features of the artefact first and add then add any additional features if any time remained at the end of the project, overall leading to an artefact that would meet its client needs more effectively.

On the other hand the product backlog could also be argued to have a negative impact upon the development of the artefact. One reason that this could be argued was due to many tasks within sprints being prioritised as high priority (See Appendix 1.3) this meant that when it came to task allocation within the Sprints, some tasks were allocated in arguably wrong positions leading to tasks the client arguable considered to be less important completed before tasks the client argued to be more important, this lead to the artefacts development to be arguable less efficient that it could have been.

Additionally the product backlogs implementation from the Scrum methodology also had a reduced impact due to factors such as members of the team not always showing (See Appendix 1.5) additionally due to environmental impacts such as a lack of time to complete tasks from the backlog due to workshops only consisting a maximum of two hours which directly impacted the artefacts development process. In relation to a critical reflection standing point, it may have been more beneficial to gain a better understanding of the individual importance of each task within a Sprint to allow for the better allocation of task completion for future Sprints which may lead to a higher productivity level in terms of the artefacts development, while also attempting to be better handle team members non-attendance.

3.5 Scrum meeting

The Scrum meeting was another important piece of the scrum methodology that was implemented and had a direct impact within the development of the artefact “Scrum meeting by a Scrum team is a 15-minute long and each team member addresses three questions: what did I do yesterday, what will I do today and what impediments are in my way? Scrum produces three artefacts, namely: product backlogs, sprint backlogs and burn-down charts.” (Hossain, 2009, 176) Within the development of the artefact the Scrum meetings proved useful due to a number of reasons. Firstly the Scrum meetings allowed for team members to discuss any issues present within the current

sprint which allowed other members to help or resolve any issues present, this has the impact of the quicker resolution of errors or problems within the Sprints (See Appendix 1.5) meaning the development of the artefact was much more quicker than if Scrum meetings were not implemented.

On the other hand it can also be argued that Scrum meetings proved obstructive and had a negative impact upon the development of the artefact. One reason for this argument is some members had strong viewpoints within these Scrum meetings and attempted to influence the way some members completed their tasks with ultimately lead to a breakdown in communication and also lead to demoralisation of some members (See Appendix 1.5) this impacted the development of the artefact in the form of a slower development process and in some cases a lower standard of development quality. Additionally the impact of scrum meetings were also hindered by members not attending (See Appendix 1.6) this meant that the team sometimes had an unclear view on the current status of the artefact leading to the artefacts development being prolonged or reserved for the next Sprint, overall reducing the productivity of development. In terms of a critical reflection stand point, the Scrum meetings could be improved in the future by team members being more professional while also a better standard of communication throughout the Scrum process which may result in the better rely of information of absent team members meaning Scrum meetings would be more effective and overall have more of a positive impact upon the artefact.

3.6 Sprint

One of the most important elements of the scrum methodology is the Sprint “A sprint produces a visible, usable, deliverable product that implements one or more user interactions with the system. The key idea behind each sprint is to deliver valuable functionality” (Rising et al, 2000, 30) within the development of the artefact the Sprints proved useful for a number of reasons. Sprints allowed for the ability to efficiently pull tasks from the product backlog to be completed within set periods of time, this allowed for a more efficient process of development of the artefact across the period of Sprints. Sprints were also useful to the artefacts development due to being short time based, this meant that it was easy to assess performance and plan necessary adaptations if necessary making the ability to change and effect the development of the artefact much more effective while also allowing for the ability to adapt to changed user requirements (See Appendix 1.5) meaning the development of the artefact was much more efficient than using a different method.

On the other hand it can also be argued that Sprints also proved obstructive at times and sometimes had a negative impact upon the development of the artefact. One reason Sprints could be argued to be obstructive to the artefacts development was its to do with its poor application, in relevance to issues such as some Sprints went on for too long of a period of time (See Appendix 1.6), this meant that sometimes it was difficult to assess performance and plan necessary adaptations within longer sprints. As an overall reflection to the scrum methodology it may be more beneficial to shorten sprints by splitting longer sprints into two different sprints therefore reducing any negative impacts towards the development of the artefact.

3.7 Pair Programming

One type of agile development used within the development of this artefact was pair programming “two programmers working side-by-side at one computer, collaborating on the same design, algorithm, code or test” (Jeffries, 2000, 2) within the development of the artefact pair programming proved useful in terms of the artefacts development for a number of reasons. Firstly it allowed for arguably higher quality implementation to the development of the artefact as it allowed for more than one person to develop the same area of code which would likely be better than one person (See Appendix 1.1) Secondly pair programming was useful towards the development of the artefact

due to the ability for more than one person to solve issues or problems therefore increasing the level of efficient and impacting the development of the artefact in a positive manner.

Alternatively pair programming can also be reflected to be disruptive to the development of the artefact. One reason being that pair programming was sometimes arguable slower to implement than when team members programmed on their own, this was possibly due to communication issues while also some team members had different styles for which they preferred to implement code, this arguably resulted in a less efficient implementation of the artefact. In terms of a critical reflection stand point pair programming could have been more useful within the Scrum methodology if team members were more similar and had better levels of communication.

4.0 Alternative software development methodologies

XP (Extreme programming)

One alternative software development methodology that may have been more beneficial to the development of the artefact is the Extreme programming (XP) based methodology “Extreme Programming emphasizes teamwork. Managers, customers, and developers are all equal partners in a collaborative team. Extreme Programming implements a simple, yet effective environment enabling teams to become highly productive” (Wells, 2013) arguably this methodology could be more appropriate than that of Scrum to the development of the artefact. One reason being the ability of having increased freedom in relevance to task implementation meaning features can be swapped into the XP team’s iteration more easily than that of Scrum, this could have proven more beneficial to the artifacts development in reference to issues that occurred within the Scrum methodology of not adopting the right tasks due to many tasks being high priority (See Appendix 1.1), therefore the XP methodology arguably would be better adapted to solve this issue therefore leading to more of a positive impact than that of Scrum.

Another reason that XP may be more of an appropriate methodology to this artefact is the fact the features of the artefact are developed to the prioritisation of the customer, and the team is then required to work on the specific tasks in a specific linear order, this could prove beneficial over the Scrum methodology due to the fact many tasks were prioritised as important (See Appendix 1.5) meaning it was hard to choosing which tasks were more important to the client than others, additionally within this specific artefacts development the client had a professional and clear insight of what was required, therefore using this methodology it would eliminate this problem and make it the more appropriate choice for the development of the artefact.

5.0 Critical evaluation

In relation to the use of the Scrum methodology there are both a number of key advantages and disadvantages to the implementation of this methodology. One key advantage of the use of Scrum is the use of Sprints throughout the implementation of an artefact, this methodology shields itself in events for worst case scenarios as its Sprint processes allow for a working artefact after each Sprint “During each sprint we perform all activities necessary to create a working product increment (some of the product, not all of it” (Rubin, 2012, 34) This differs greatly from that of most other methodologies that follow step by step approaches to develop a working product at the end of the cycle, one example of an this is the waterfall methodology “Developing a system using the Waterfall Model can be a long, painstaking process that does not yield a working version of the system until

late in the process.” (Centre Technology in Government, 2003) Scrum allows for a working product after each sprint iteration, giving it a clear advantage over other such methodologies.

Meanwhile another key advantage of the Scrum methodology is the ability to review the artefact after each sprint before moving onto the next, therefore meaning testing is completed throughout the artefacts development process, this provides a clear advantage over other many different methodologies that only complete testing once the artefact has been fully built, for example the waterfall methodology “When in the testing stage, you have no option of changing the designing that was not well thought of. This is a heavy concern considering no single working product is produced until later stages of the cycle.” (Silver, 2014) Scrum allows for the ability to repeat testing through each sprint iteration meaning the end product will be functional, unlike the risk of other development methodologies such as Waterfall.

On the opposing side there are also a number of disadvantages of the Scrum methodology than can be discussed. Firstly one main disadvantage of the Scrum methodology is the reliance on team members “The team holds a Daily Scrum meeting every day. All team members (including the Scrum Master and Product Owner) must attend” (Highsmith, 2014) This could be argued to be a key weakness with the Scrum methodology as team members are assigned set tasks that cannot be changed and if issues with team members do arise it could have a serious impact on the development of an artefact, compared to that of other methodologies such as the spiral methodology which has risk approaches “The spiral model is similar to the incremental model, with more emphasis placed on risk analysis” (ISTQB EXAM CERTIFICATION ,2015) Scrum does not take into account any risk factors which could prove to be a weakness with choosing this software development method. Furthermore the Waterfall method could also be a good alternative “In the case of employee turnover, waterfall’s strong documentation allows for minimal project impact.” (Base36, 2012) Therefore making the Waterfall method another clear option over Scrum which lacks clear documentation of such things as implementation which means Scrum could be argued to be much more of a higher risk methodology than others available.

Furthermore another disadvantage that could be apparent with the Scrum methodology is the reliance upon the product owner “It requires the product owner to lead product discovery, to help identify and describe requirements, and to make sure that the product backlog is ready for the next sprint planning meeting. It also means that the product owner has to engage in product planning, visioning and product road mapping” (Pichler, 2010) Arguably this could be identified to be a weakness of the Scrum methodology compared to other methodologies available as the product owner could cause a large negative impacts upon the development of the artefact than that of other methodologies such as Waterfall as the method follows a sequential method where user requirements are first gathered from the client before any implementation occurs “In waterfall methodologies all the requirements gathering and design work is done before any coding takes place.” (Bowes, 2014) This provides the benefit of the user not impacting the development of the artefact during implementation stages unlike that of the Scrum methodology as the user provides much less of a role within the development of the artefact.

As an overall evaluative argument I would argue that although Scrum has a number of disadvantages that could occur with the implementation of this specific methodology, there are also a number of strengths that in my opinion outweigh the weaknesses of this methodology, although I would also comment it entirely depends on many other factors such as the artefact, team size and product owner that could also influence the choice of methodology that would be best appropriate to implement.

References

1. SourceMaking. (2015) Design Patterns. [Online] Available from: https://sourcemaking.com/design_patterns [Accessed 16/12/2015]
2. AG Communication Systems. (1999) Design Patterns - Elements of Reusable Object-Oriented Software. [Online] Available from: http://asi.insa-rouen.fr/enseignement/siteUV/genie_logiciel/supports/ressources/exemples_de_la_vie_reelle_pour_illustrer_pattern.pdf [Accessed 16/12/2015]
3. Yoda. (2006) Implementing the Singleton Pattern in C#. [Online] Available from: <http://www.yoda.arachsys.com/csharp/singleton.html> [Accessed 16/12/2015]
4. Hong, K. (2015) Design Patterns Model View Controller (MVC) Pattern. [Online] Available from: http://www.bogotobogo.com/DesignPatterns/mvc_model_view_controller_pattern.php [Accessed 16/12/2015]
5. T, Foegen. (2010) what is Scrum? . [Online] Available from: https://www.wibas.com/media/filer_public/2013/09/30/wibas_whatisscrum_en.pdf [Accessed 16/12/2015]
6. Mountain Goat (2015) Scrum Master. [Online] Available from: <https://www.mountaingoatsoftware.com/agile/scrum/scrummaster> [Accessed 16/12/2015]
7. Berteig, M. (2014) the rules of scrum: all people outside the team know that it is the scrum master's job to shield the team from interruptions. [Online] Available from: <http://www.agileadvice.com/2014/01/28/scrumxplean/the-rules-of-scrum-all-people-outside-the-team-know-that-it-is-the-scrummasters-job-to-shield-the-team-from-interruptions/> [Accessed 16/12/2015]
8. Scrum (2015) Glossary of Scrum Terms. [Online] Available from: <https://www.scrum.org/Resources/Scrum-Glossary> [Accessed 16/12/2015]
9. Schwaber, K. (2014) What Is Scrum? . [Online] Available from: http://www.volaroint.com/wp-content/uploads/dlm_uploads/2014/03/DC-VOLARO-Training-Scrum-What_Is_Scrum.pdf [Accessed 16/12/2015]
10. Hossain, E. and Babar, M. (2009) Using Scrum in Global Software Development: A Systematic Literature Review. Fourth IEEE International Conference on Global Software Engineering
11. Rising, L. and Norman, S. (2000) The Scrum Software Development Process for Small Teams. [Online] Available from: <http://faculty.salisbury.edu/~xswang/Research/Papers/SERelated/scrum/s4026.pdf> [Accessed 16/12/2015]
12. Jeffries, R. (2000) Strengthening the Case for Pair-Programming. [Online] Available from: <http://www.cs.utah.edu/~lwilliam/Papers/ieeeSoftware.PDF> [Accessed 16/12/2015]
13. Wells, D. (2013) Extreme Programming: A gentle introduction. [Online] Available from: <http://www.extremeprogramming.org/> [Accessed 16/12/2015]
14. Rubin, K. (2012) Essential Scrum: A Practical Guide to the Most Popular Agile Process. Upper Saddle River: Addison-Wesley.
15. Centre Technology in Government. (2003) The Waterfall model. [Online] Available from: http://www.ctg.albany.edu/publications/reports/survey_of_sysdev?chapter=5&PrintVersion=2 [Accessed 16/12/2015]
16. Silver, M. (2014) Waterfall Methodology Advantages and Disadvantages. [Online] Available from: <http://spectechular.walkme.com/waterfall-methodology-advantages-disadvantages/> [Accessed 16/12/2015]

17. Highsmith, J. (2014) Chapter 4. Scrum and Self-Organizing Teams. [Online] Available from: <https://www.safaribooksonline.com/library/view/learning-agile/9781449363819/ch04.html#CHP-4-FN-1> [Accessed 16/12/2015]
18. ISTQB EXAM CERTIFICATION. (2015) What is Spiral model- advantages, disadvantages and when to use it? . [Online] Available from: <http://istqbexamcertification.com/what-is-spiral-model-advantages-disadvantages-and-when-to-use-it/> [Accessed 16/12/2015]
19. Base36. (2012) Agile & Waterfall Methodologies – A Side-By-Side Comparison. [Online] Available from: <http://www.base36.com/2012/12/agile-waterfall-methodologies-a-side-by-side-comparison/> [Accessed 16/12/2015]
20. Pichler, R. (2010) Common Product Owner Traps. [Online] Available from: <https://www.scrumalliance.org/community/articles/2010/april/common-product-owner-traps> [Accessed 16/12/2015]
21. Bowes, J. (2014) Agile vs Waterfall: Comparing project management methods. [Online] Available from: <http://manifesto.co.uk/agile-vs-waterfall-comparing-project-management-methodologies/> [Accessed 16/12/2015]

Appendix

1.1 Pair Programming Log

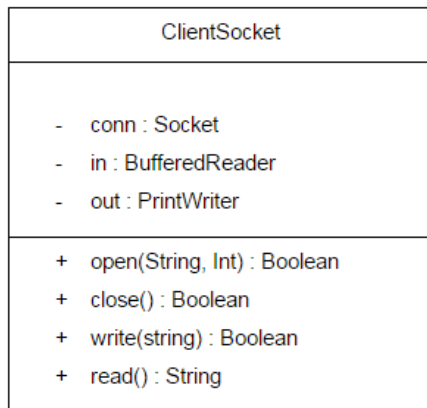
Date	Observer	Partner	Duration Driving	Errors	Types of error	Comments
30/10/15	Ashley	Gary	15m	0	N/A	Documentation added to Client Socket class.
30/10/15	Jordan	Gary	20mins	0	N/A	We found one issue in which the computer we using was not connected to the correct internet meaning we couldn't connect to the STSServer.
30/10/15	Gary	Jordan	20 minutes	1	Program hanged	Sending the wrong text command to the server resulted in the program hanging when trying to receive the response. The server only responded to one single text command.
6/11/15	Gary	Ashley	1 hour	5	Threading errors.	Unused Socket reference in sending and receiving classes. Issues implementing cross thread communication. Use of on-thread safe data types. Thread commands outside of synchronization block. Use of wait instead of sleep. Synchronization on object field rather than object.

6/11/15	Ashley	Gary	1 hour	Many	Issues with implementation of Cross-Thread communication	NullPointerExceptions on test Server - DISP has a null reference to stock market. "mySM" vs "mySM.getStockMarket()". Monitor exceptions when using Synchronization methods and blocks. Synchronization block within a Synchronization method. Readline preventing ability to interrupt reader thread due to it blocking. Input Stream use rather than BufferedReader for 'ready'. Extra newline on last server response.
6/11/15	Dean	Peter	1 Hour	0	N/A	Implement additional functions to initial design Altering the method used for input, using single characters for commands
30/11/15	Ashley	Gary	4 Hour	Many	SQL Errors, in both Database Schemata Implementation of multiple facades and implementers for SQL system and System facades to link. Constraints on data, and the way which updating happens.	Entire SQL Database designed and hosted. SQL Connector designed and written. SQL Facade implemented. System Facade design and implemented to bridge User layer with SQLHelper layer, Remote Stocks and Remote DB for logging. Account, Transaction, and Stock Classes defined for use in these systems.
4/12/15	Jordan	Peter	30mins	Multiple	Function names not resolved in library, Functions not having intended effects, NullPointerException	Attempted to create a console user interface using the Lanterna library (https://code.google.com/p/lanterna/). The library documentation did not match up with the version used in the program leading to confusion in trying to create interface elements. Little overall progress was made and lanterna was dropped in favour of Java Swing GUI
11/12/15	Peter	Gary	1hr	2	Null pointer, Database Connection Refused errors	The network in the computer labs prevented connections on the port used by the SQL, meaning

						the Interface was blocked. Functions were not implemented at the facade level to push updates to the interface so a monitor class was implemented to update and display stocks periodically.
11/12/15	Ashley	Peter	20 minutes	1	Server throwing errors on buy/sell	Each item in stock item from comboList wasn't uppcased, traced problem down to database. Client changed data schema, was using toLower, new data require toUpper.
11/12/15	Gary	Peter	20 minutes	1	Syncing issues	Client changed data update rate, causes issues trying to sync clocks.
12/12/15	Ashley	Gary	40 minutes	Many	UI Integration and refactoring	Found that some methods weren't included in system facades, added overloads for those.
12/12/15	Gary	Ashley	40 minutes	Many	Jarvis implementation	Certain exposed system facade functions weren't exactly optimal and required overloading. Modifying existing Monitor.

1.2 UML diagrams and class diagrams

Software Engineering - Stock Market System



conn : Socket - This is used to store the socket once created.

in: BufferedReader - This is used to send a request to the server, reading from stdin.

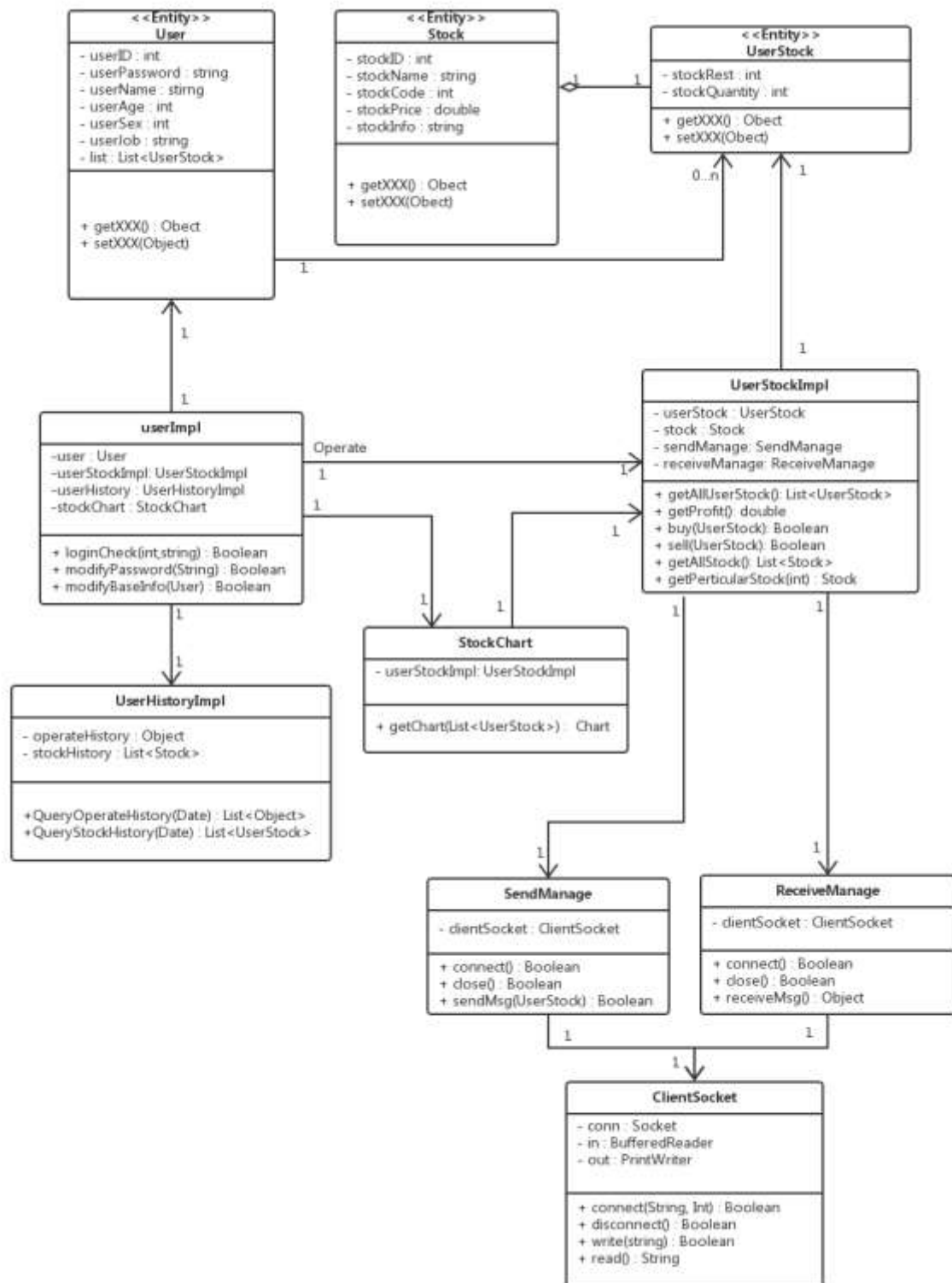
out: PrintWriter - This is used to read a String response from the server.

Open: is used to make the initial connection the server. Connect would take two parameters - The IP and Port.

Close: Close is used to terminate the connection to the server.

Write: Write is a method in which we send data to the server.

Read: Read is the method in which we read from the server and return any input from the server.



Class diagram:

- 1) **User** : is a entity,containing records of the user's profile, including name, ID, Password, gender,age and shares owned.
- 2) **Stock** : is a superclass, containing records of all stock.
- 3) **UserStock** : inherits from **Stock** class, adding new attributes of stock and quantity.

All three of the above classes are entities,it is implemented as a function with corresponding methods to get and set the attribute values.

- 4) **UserImpl** ,are contained with these main methods:
 loginCheck () is a method of login information authentication.
 modifyPassword () is a method in which users could set a new password.
 modifyBaseInfo () revises the fundamental user information.

This class will invoke methods from **UserStockImpl** class,used to realize querying stock information , buying, selling, viewing details, etc

Invoking getChart()method form **StockChart**,used to realize viewing 'trend' of stock.
 Invoking method form **UserHistoryImpl**,used to query the history of user operation record and user shares holding.

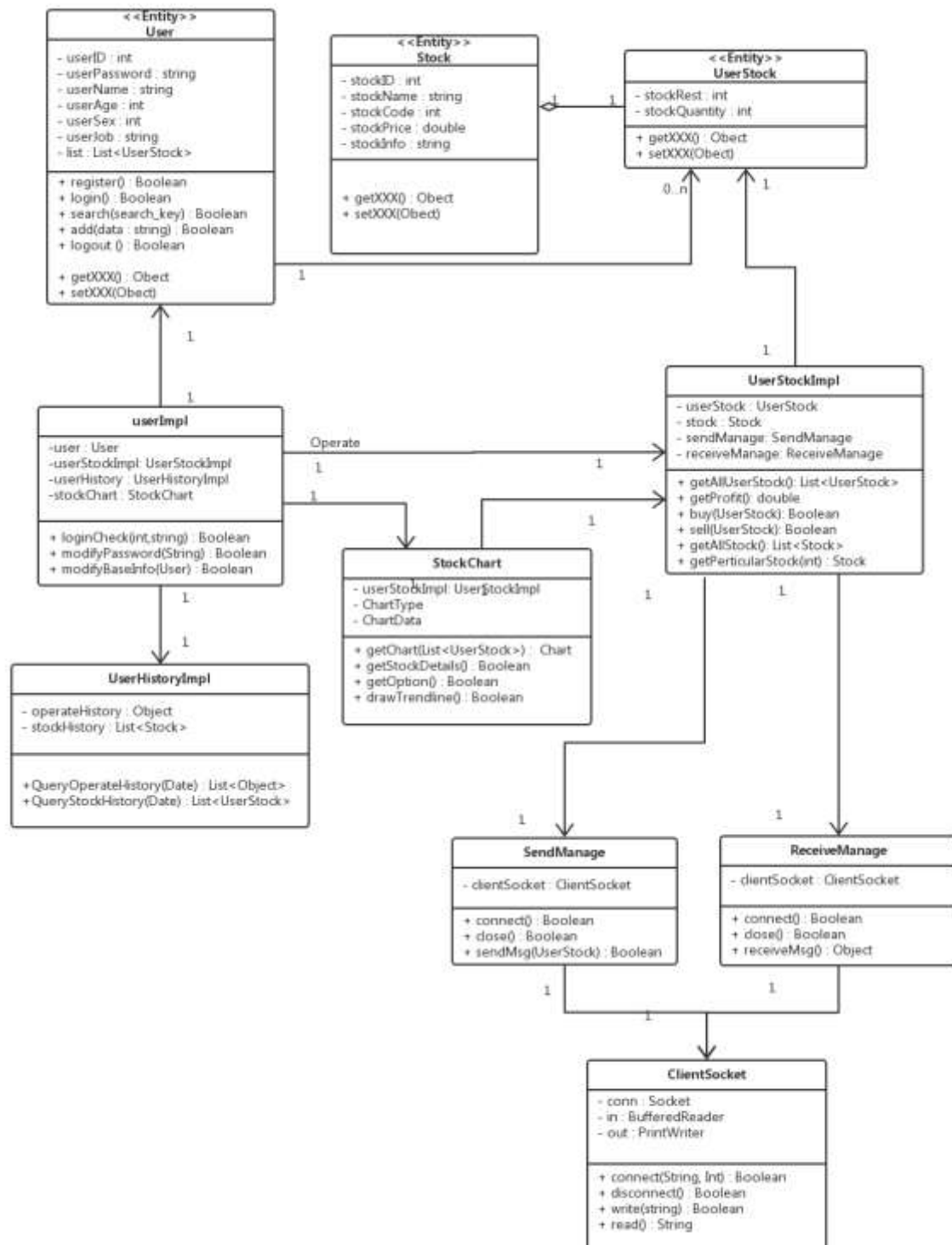
- 5) **UserStockImpl** ,are contained with these main methods :
 getAllUserStock(): get all stock of the current user
 getProfit(): is used to figure out profits of the current stock.
 buy(UserStock): purchasing shares
 sell(UserStock): selling shares
 getAllStock(): all of the information of stock is listed on the form
 getParticularStock(int) : user passes in a stock symbol; the provider returns the current price and the current day's high, low, and volume for the indicated stock.

- 6) **SendManage ReceiveManage** :both of them used to manage socket,which can be used to invoke by **UserStockImpl**
 are contained with these main methods :
 connect() : opening connection
 close() : closing connection
 sendMsg(UserStock) : through socket sends stock message which purchased
 (SendManage owned)
 receiveMsg() : through socket gets the stock information(ReceiveManage owned)

- 7) **ClientSocket** connected clients
 are contained with these main methods :
 Connect () : is used to make the initial connection the server. Connect would take two parameters - The IP and Port.
 Disconnect () : Close is used to terminate the connection to the server.
 Write () : Write is a method in which we send data to the server.
 Read () : Read is the method in which we read from the server and return any input from the server

- 8) **UserHistoryImpl** ,are contained with these main methods :
 QueryOperateHistory (Date) : according to the time to query the history of user operation record.
 QueryStockHistory (Date) : according to the time to query the history of user shares holding.

9) StockChart ,are contained with these main methods : :
 getChart ()used to get information for making 'trend' of stock.



Class diagram:

- 1) **User** : is a entity,containing records of the user's profile, including name, ID, Password, gender,age and shares owned.
- 2) **Stock** : is a superclass, containing records of all stock.
- 3) **UserStock** : inherits from **Stock** class, adding new attributes of stock and quantity.

All three of the above classes are entities,it is implemented as a function with corresponding methods to get and set the attribute values.

- 4) **UserImpl** ,are contained with these main methods:
 loginCheck () is a method of login information authentication.
 modifyPassword () is a method in which users could set a new password.
 modifyBaseInfo () revises the fundamental user information.

This class will invoke methods from **UserStockImpl** class,used to realize querying stock information , buying, selling, viewing details, etc

Invoking getChart()method form **StockChart**,used to realize viewing 'trend' of stock.

Invoking method form **UserHistoryImpl**,used to query the history of user operation record and user shares holding.

- 5) **UserStockImpl** ,are contained with these main methods :
 getAllUserStock(): get all stock of the current user
 getProfit(): is used to figure out profits of the current stock.
 buy(UserStock): purchasing shares
 sell(UserStock): selling shares
 getAllStock(): all of the information of stock is listed on the form
 getPerticularStock(int) : user passes in a stock symbol; the provider returns the current price and the current day's high, low, and volume for the indicated stock.

- 6) **SendManage,ReceiveManage** :both of them used to manage socket,which can be used to invoke by **UserStockImpl**
 are contained with these main methods :
 connect() : opening connection
 close() : closing connection
 sendMsg(UserStock) : through socket sends stock message which purchased
 (SendManage owned)
 receiveMsg() : through socket gets the stock information(ReceiveManage owned)

- 7) **ClientSocket** connected clients
 are contained with these main methods :
 Connect () : is used to make the initial connection the server. Connect would take two parameters - The IP and Port.
 Disconnect () : Close is used to terminate the connection to the server.
 Write () : Write is a method in which we send data to the server.
 Read () : Read is the method in which we read from the server and return any input from the server

- 8) **UserHistoryImpl** ,are contained with these main methods :

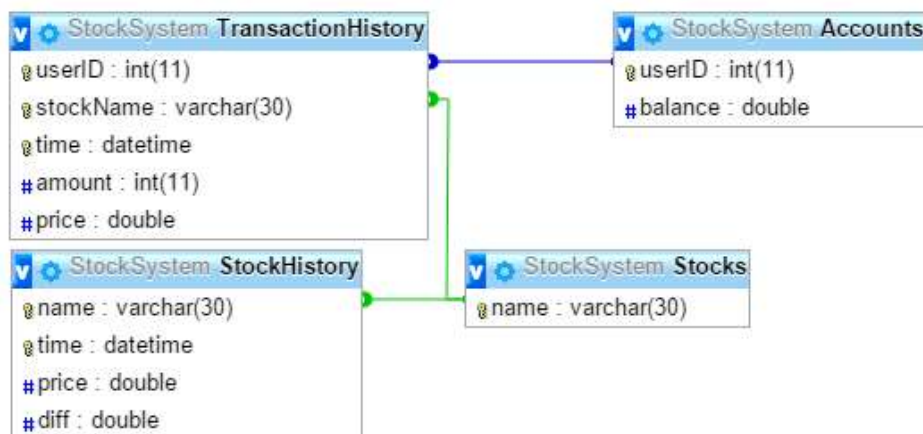
QueryOperateHistory (Date) : according to the time to query the history of user operation record.

QueryStockHistory (Date) : according to the time to query the history of user shares holding.

9) **StockChart** ,are contained with these main methods :

getChart () used to get information for making 'trend' of stock.

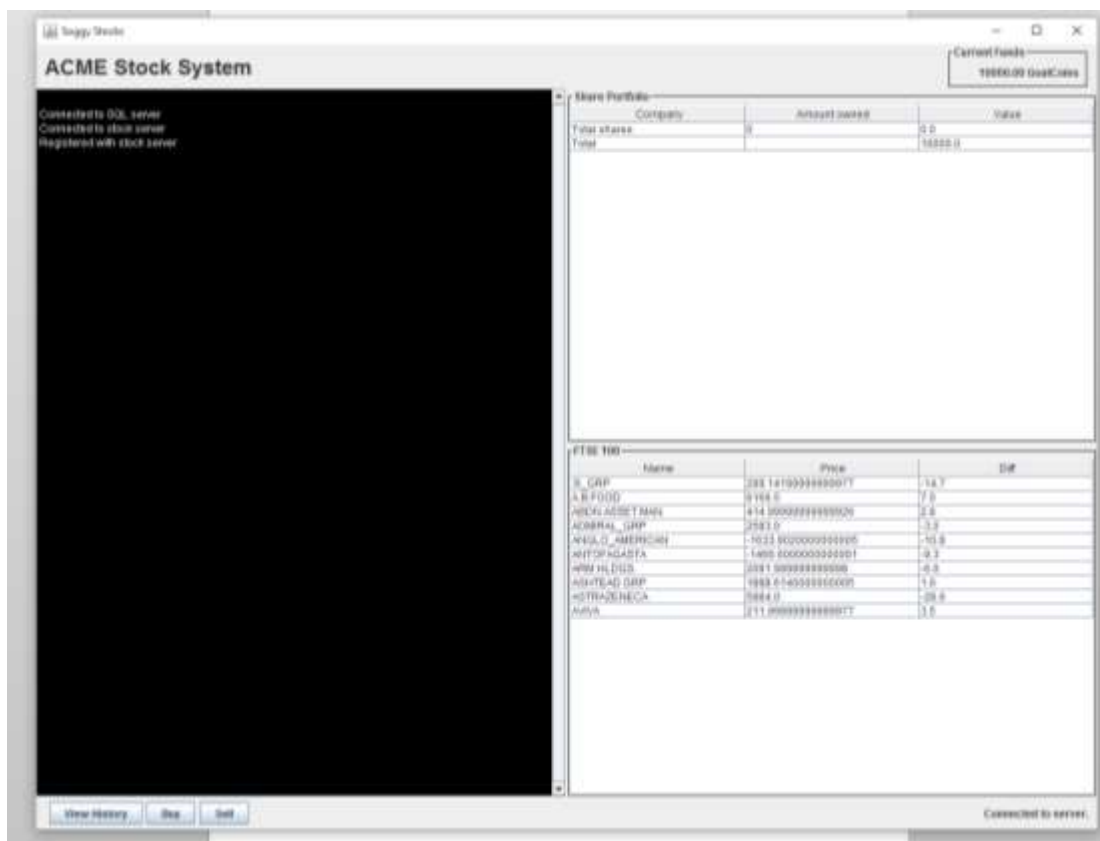
Database Diagram



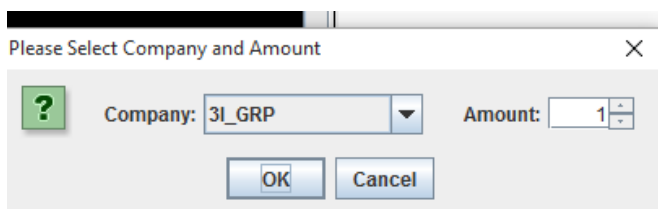
Software Engineering - Stock Market System

Facade
<ul style="list-style-type: none"> - ClientSocket clientSocket; - <u>SendMesssages</u> sender; - ReadMessages receiver;
<ul style="list-style-type: none"> + Facade(ip, port); + buy(); + sell(); + getStocks(); + close();

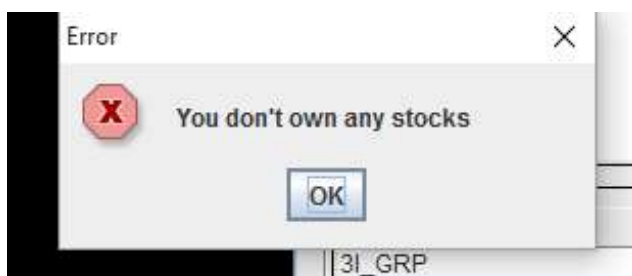
1.2 Artefact running (Stock System)



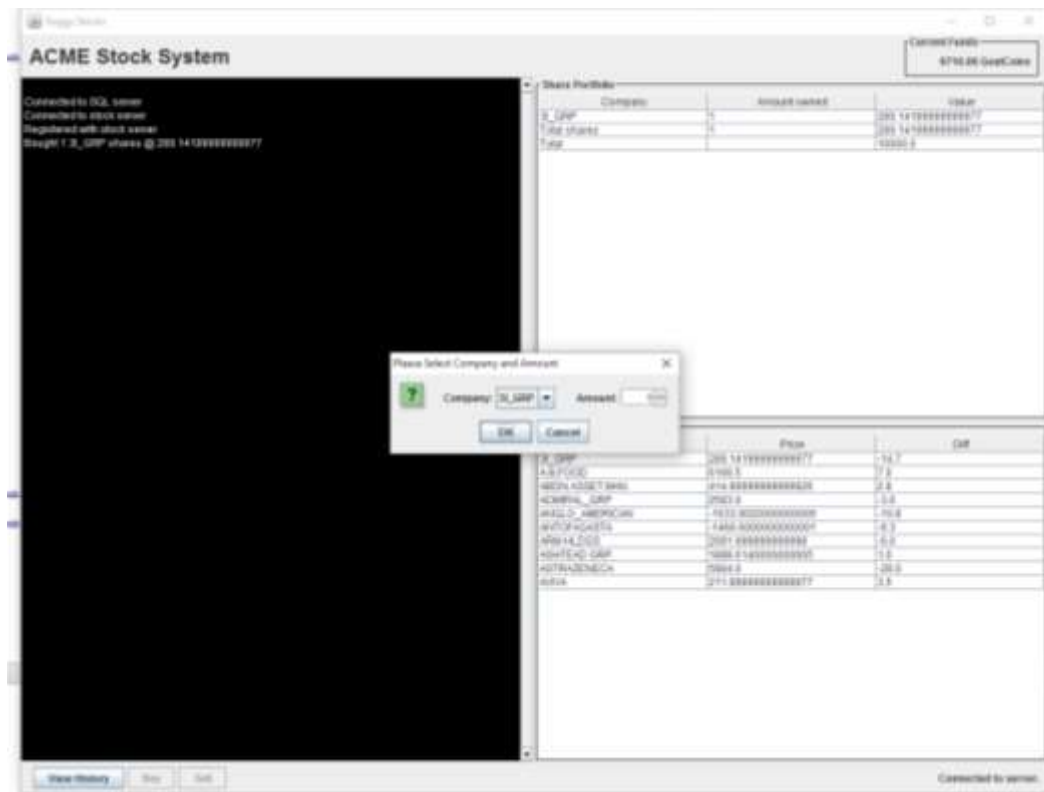
Buy Stocks



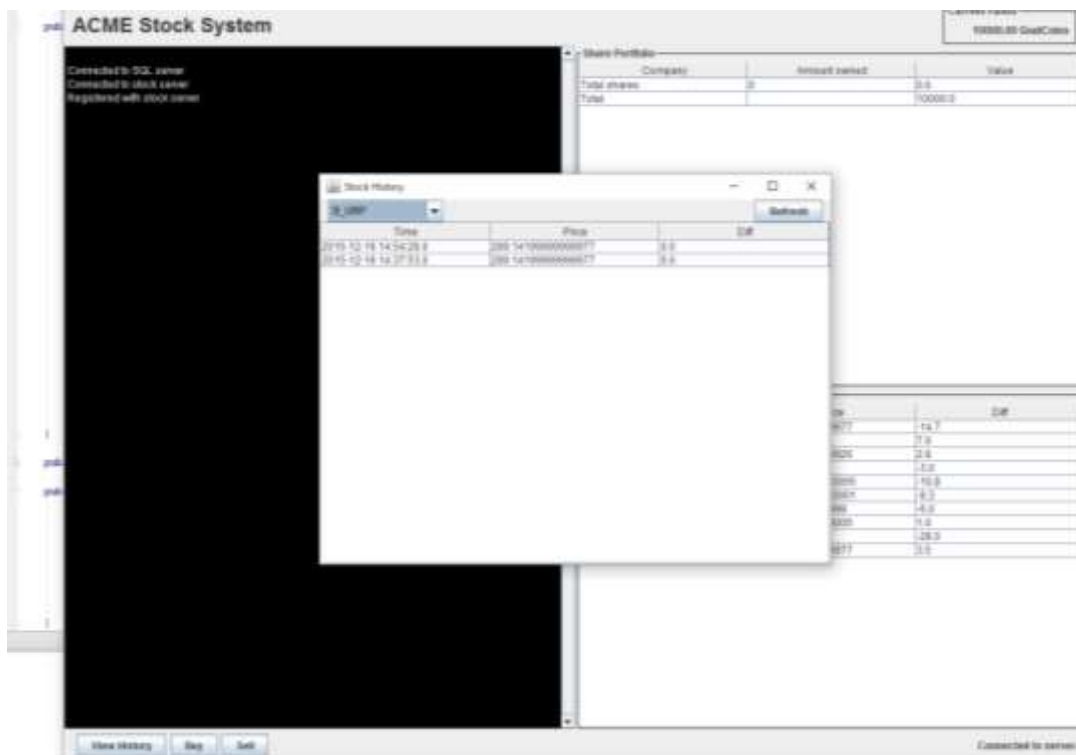
Sell in no stocks



Selling Stocks and viewing Stocks owned



[View history](#)





1.3 User Requirements

Initial

Must

1. The system must automatically buy and sell stocks to make money.
2. History of previous stock prices must be viewable, longer than 1 day.
3. Must provide manual ability to buy/sell stocks.
4. UI for interfacing with the system, should show stocks, their price, and their delta.

Should

1. Run for 24 hours continuously, without requiring manual input.
2. The system must log all transactions it does.

Could

1. Graphs displaying stock over time available
2. Stock predictions
3. Automatic sanity check for an operation the system seems 'bad'.
4. Confirmation of manual operations, selling stock / buying stock.

W2

Must

Create Account Framework

Create User Trends

Filter of Global list based on owned user shares

Show past Five Changes of Filter

User Interface for Choosing Options

Authentication system framework

1.4 Testing

Server Connection Testing

Test case SC1.1 - Establish a connection to the server

Description:

The client program utilises a Java socket to connect to a server program running on the same network. The server responds to connections on port 5000 and leaves the connections open until a specific command is received.

Input Data:

None

Feature Expected Result:

Upon running the program, it should connect to a server with an IP address 192.168.0.48 and port 5000. A string is printed to the console notifying the user of a successful connection.

ACTUAL OUTCOME

Test performed on a single machine (localhost 127.0.0.1)

Client Output:

```
C:\Week1>java Main
Connected to server
ACK:/127.0.0.1:5000:/127.0.0.1:1967
Disconnected from server
```

Server Output:

```
C:\Week1>java STSServer
Listening for connections.
Listening for connections.
New client has connected, new thread started.
Client IP is: /127.0.0.1:1957

Client: /127.0.0.1:5000 : HELO
ACK:/127.0.0.1:5000:/127.0.0.1:1957
Client: /127.0.0.1:5000 : EXIT
ACK:EXIT:Goodbye!
```

Test case SC1.2 - Incorrect IP address used to connect

Description:

The server can only be located at a specific IP address and port, 192.168.0.48:5000. If the connection is unsuccessful because of a wrong IP, the socket class will throw a `java.net.ConnectionException` which should be handled.

New Input Data:

IP Address - 192.168.0.42, Port - 5000

Feature Expected Results:

The program should suffer a `ConnectException` which is caught, informing the user that it was unable to make a connection to the server before automatically exiting or asking to reattempt a connection.

ACTUAL OUTCOME

```
C:\Week1>java Main
java.net.ConnectException: Connection timed out: connect
Failed to connect to server.

C:\Week1>
```

The program prints out the exception along with a message explaining that it failed to connect before exiting the program.

Test case SC1.3 - Connection to server closed on program exit*Description:*

The socket client requires that the connection be closed via `socket.close()` when finished sending data. This should happen after the user has finished inputting commands and types 'EXIT' or uses the terminate shortcut CTRL + C.

New Input Data:

- 'EXIT'
- Keyboard shortcut CTRL + C

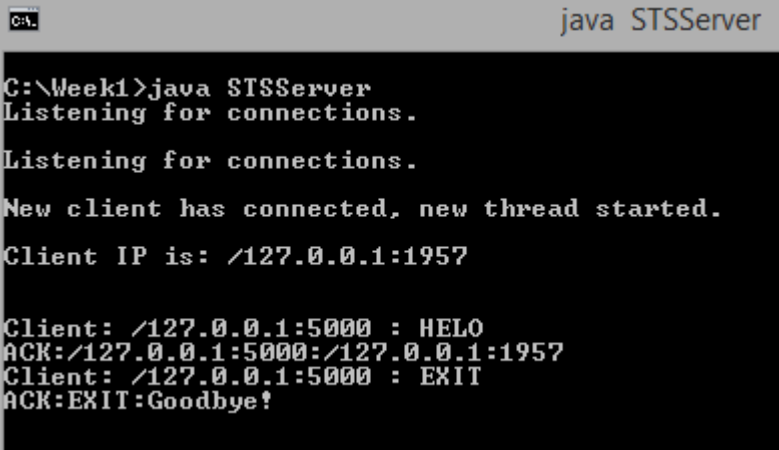
Feature Expected Results:

The program successfully terminates the active socket connection, the program execution ends and returns the user to the command line.

ACTUAL OUTCOME**Client Output:**

```
C:\Week1>java Main
Connected to server
ACK:/127.0.0.1:5000:/127.0.0.1:1957
null
Disconnected from server
```

Server Output:



```

C:\Week1>java STSServer
Listening for connections.
Listening for connections.
New client has connected, new thread started.
Client IP is: /127.0.0.1:1957

Client: /127.0.0.1:5000 : HELO
ACK:/127.0.0.1:5000:/127.0.0.1:1957
Client: /127.0.0.1:5000 : EXIT
ACK:EXIT:Goodbye!

```

Unexpected outputs:
The client outputs null on the EXIT command.

Command Testing

Test case CM1.1 - After connection established, user inputs a command

Description:

The program takes as input simple text commands such as 'HELO' or 'EXIT' and the server responds. Instead, input lower case versions of these commands.

Input Data:

- 'helo'
- 'exit'

Feature Expected Result:

The server only responds to the upper case commands 'HELO' and 'EXIT'. Using inputs a and b would result in no response and the program would hang. To avoid this, error handling should be in place to prevent any invalid commands being sent to the server, where the program continues to accept input after warning the user of an invalid command.

ACTUAL OUTCOME



```

Client:
C:\Week1>java Main
Connected to server

```

Program hangs, since no response is ever sent by the server.



```

Server:
Listening for connections.
New client has connected, new thread started.
Client IP is: /127.0.0.1:1971

Client: /127.0.0.1:5000 : helo
DEBUG:helo:

```

Note that the server never receives the second command since the client has stopped working.

Reading and sending messages Testing

Test case - send message

Description:

The client program uses a send message class to send a writeline to the server. The server should then respond to this and output the writeline.

Input Data:

"Testing Send Message"

Feature Expected Result:

The server should output with "Testing Send Message"

ACTUAL OUTCOME

Test performed on a single machine (localhost 127.0.0.1)

Client Output:



Server Output:

```
UPD:APPLE:104.81911215900458:1.108756754766277
UPD:Google:7091.162065546406:0.0
Client: /127.0.0.1:5000 : Testing Send Message
DEBUG:Testing Send Message:
Stock Market updated each 5s.
UPD:IBM:1403.6017299253099:3.9574051122250276
UPD:Microsoft:5238.281014993254:-6.168537229231415
```

Test case - Read message

Description: The client program should read the message initially inputted in the send message. Through the socket to get stock information.

Input Data: "Testing Send Message"

Feature Expected Result:

The server should output with "Testing Send Message" Which should then correspond with the client program to show the image has been read successfully

Feature Expected Result:

ACTUAL OUTCOME

Test performed on a single machine (localhost 127.0.0.1)

server Output:

```
UPD:APPLE:104.81911215900458:1.108756754766277
UPD:Google:7091.162065546406:0.0
Client: /127.0.0.1:5000 : Testing Send Message
DEBUG:Testing Send Message:
Stock Market updated each 5s.
UPD:IBM:1403.6017299253099:3.9574051122250276
UPD:Microsoft:5238.281014993254:-6.168537229231415
```

client Output:



Test case - Read Messages with Socket connection*Description:*

The client program uses a read message class and a socket connection to connect to the server to read the messages inputted

Input data: "Testing Send Message"

Feature Expected Result:

The server should successfully connect

ACTUAL OUTCOME

Test performed on a single machine (localhost 127.0.0.1)
server Output:

client Output:

**Facade Testing****Test case - Register***Description:*

The client program calls a register method to send command "Regi" to the server . The server should then respond to this and return ID back to the client.

Input Data:

"Regi"

Feature Expected Result:

The client should show the message with "Registered to Stock System with ID: "

ACTUAL OUTCOME

61596

Test case - Buy*Description:*

Buying X of a stock Y

Input Data:

BUY Y X ID

e.g. BUY:COMPANY:AMOUNT:ID

Feature Expected Result:

Deducted balance, gain X stocks of Y.

ACTUAL OUTCOME

ACK-BOUGHT:4 shares: In Google: @7118.089700414694

**Test case - Get stocks***Description:*

The client program calls a getStocks method to send command "DISP" to the server .
The server should then respond to this and return stock information lines back to the client.

Input Data:

"DISP" + ID from REGI

eg

DISP:61596

Feature Expected Result:

List of stocks delimited by END:EOF

ACTUAL OUTCOME

Example:
STD:IBM:1362:-0.6
..
STK:Google:7102:-5
END:EOF

1.5 MEETINGS

WEEK 4 (20/11/15) MINUTES

Stand up meeting:

Last week was spent catching the project up due to a server error causing the program to hang when receiving responses from the server. The team recapped the users requirements, and decided that a facade should be implemented next in order to expose a simple API for using the socket server connection.

Client proved disruptive in form of verbal communication

WEEK 3 (13/11/15) MINUTES

Stand up meeting:

Issues arose during the multi-threading implementation last week that required extensive programming time to overcome. An error in the server code prevented responses being terminated, resulting in the program hanging when reading a response from the server. This caused the project to fall behind, and so it was decided that week 3 would be used to catch up on the previous week.

WEEK 2 2/11/2015

The task this week is to implement threading into the system, this will allow for sending and receiving messages to the server. This will require the programmers to implement code to allow for the threading, designers to plan how it will be implemented and testers to create test cases and then test the code.

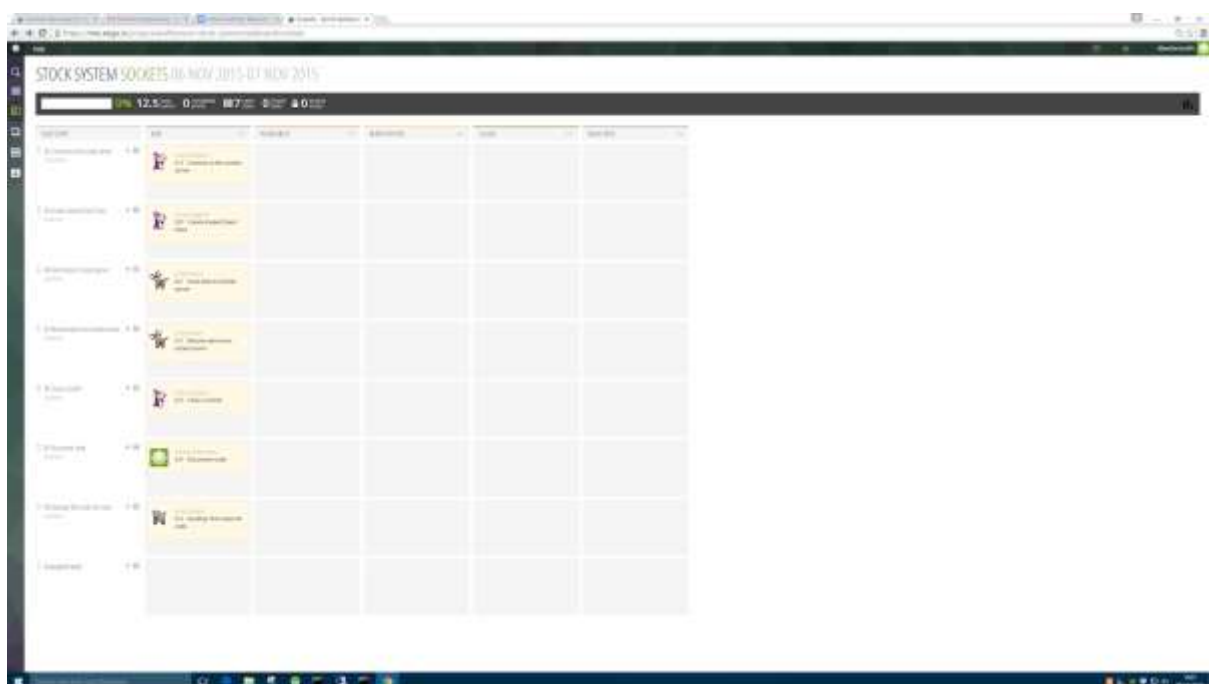
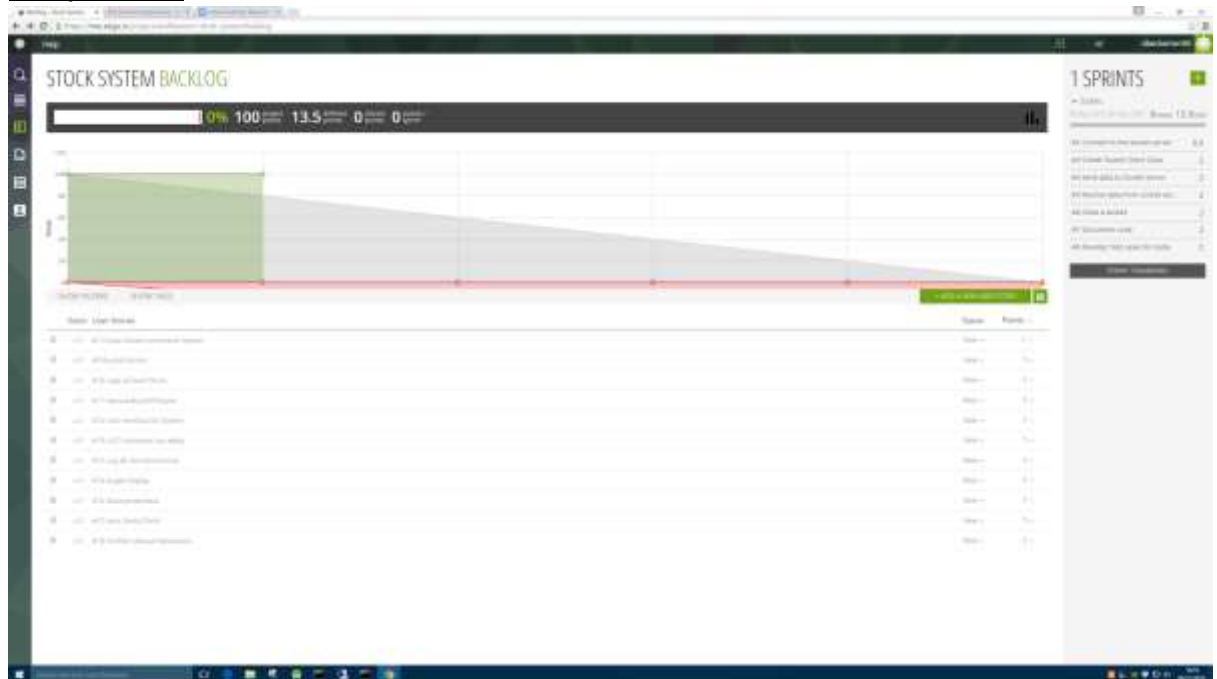
Missing from this week: Dean. Illness

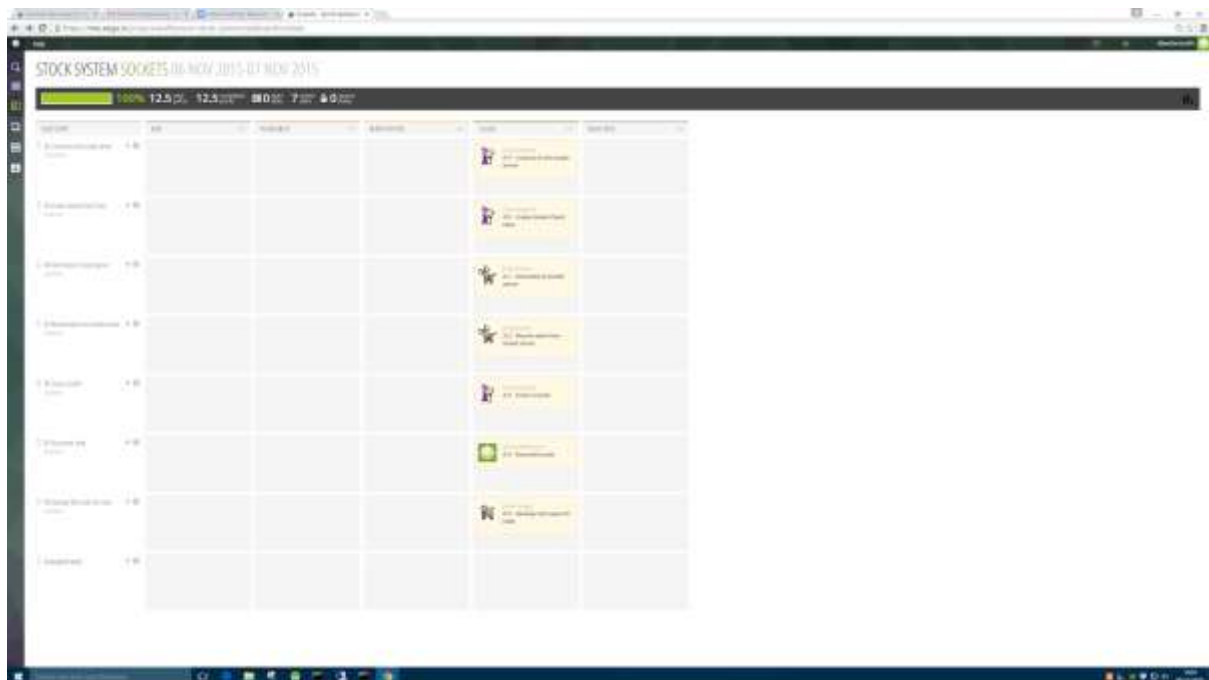
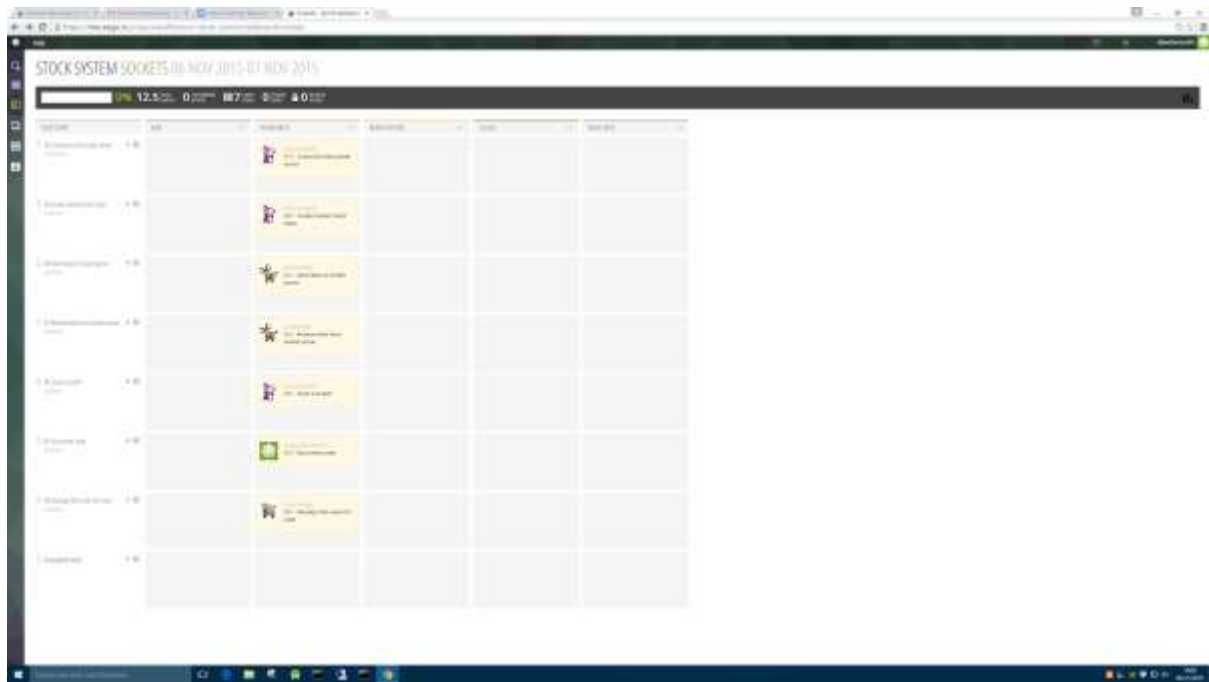
Week 1

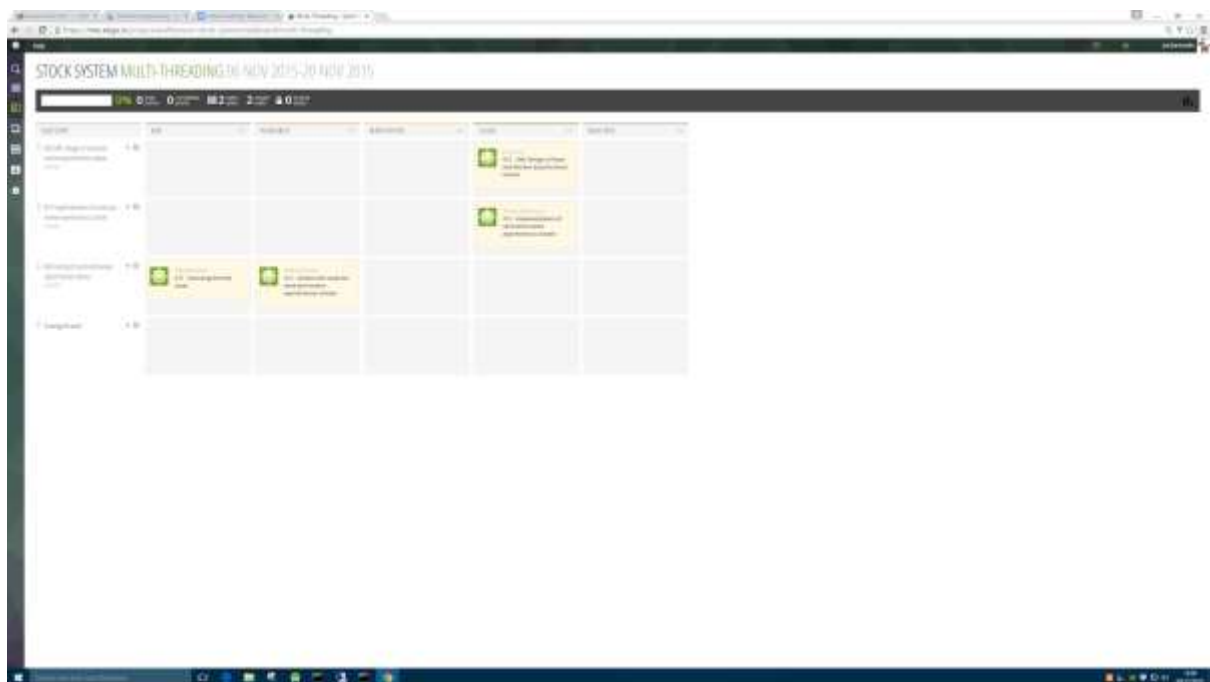
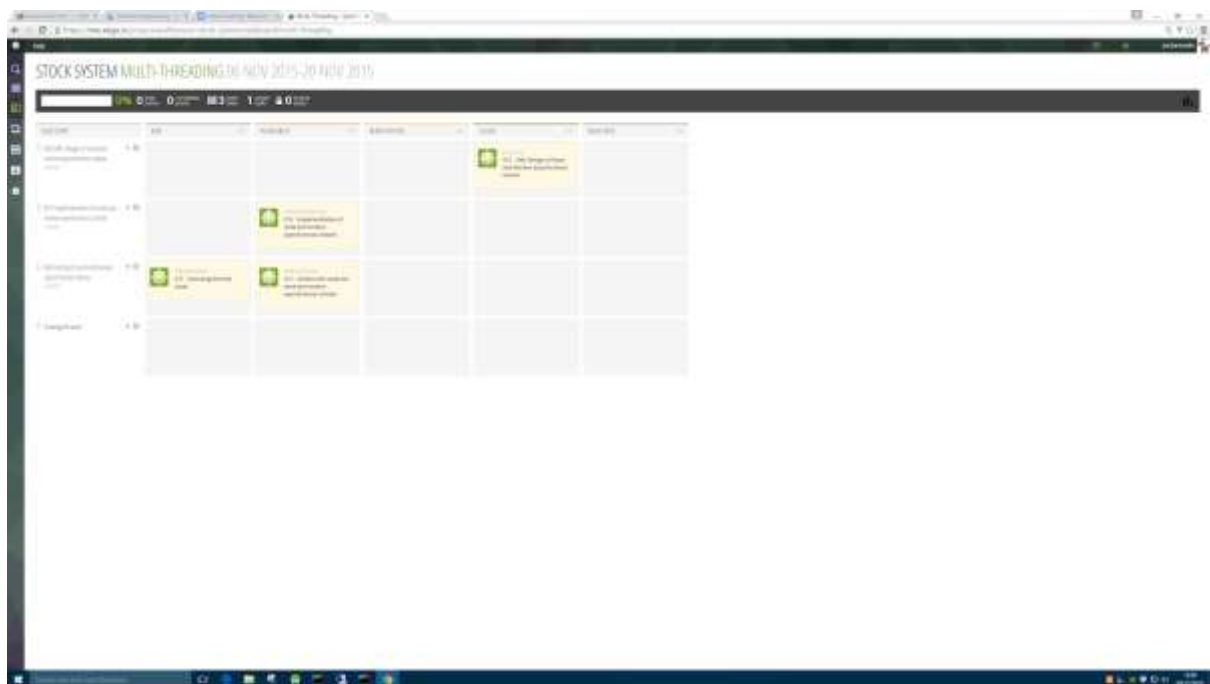
Team meeting

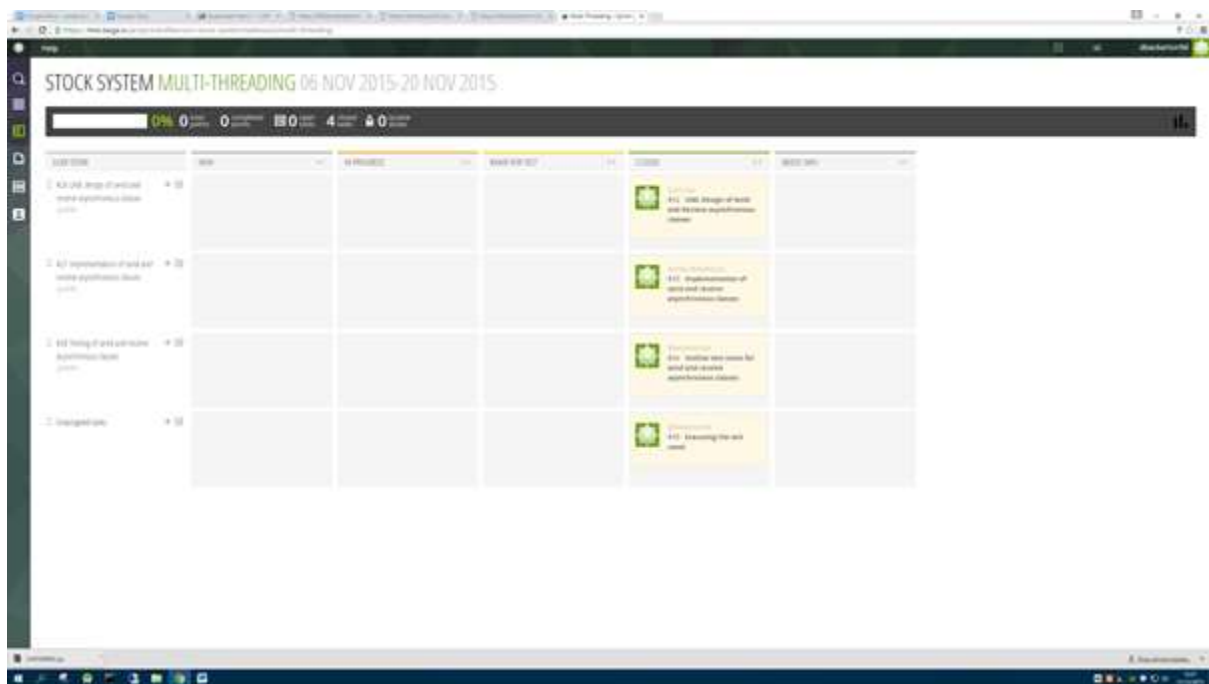
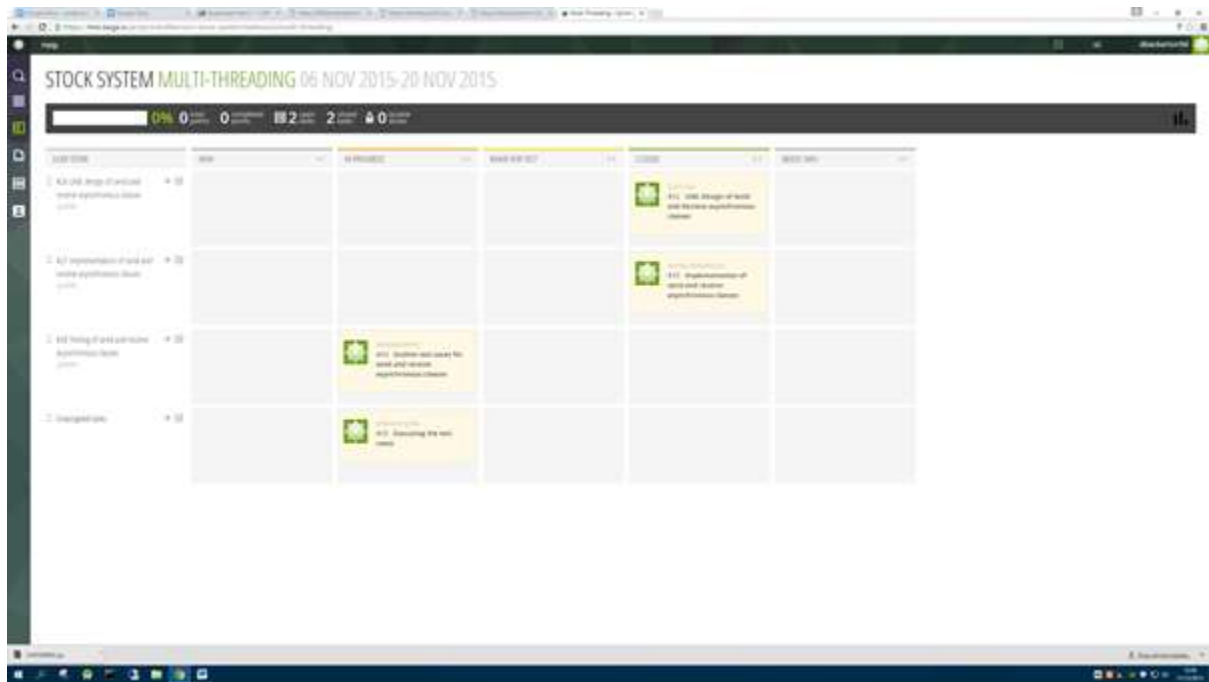
After the client meeting we got together and discussed initial conceptual designs for connecting to the socket server provided by the client. This week's requirement (Socket Server Class) was split into various tasks. We got together and signed up to Taiga.io, a SCRUM management online service, which allowed us to define a SPRINT and outline our tasks and assign them to members.

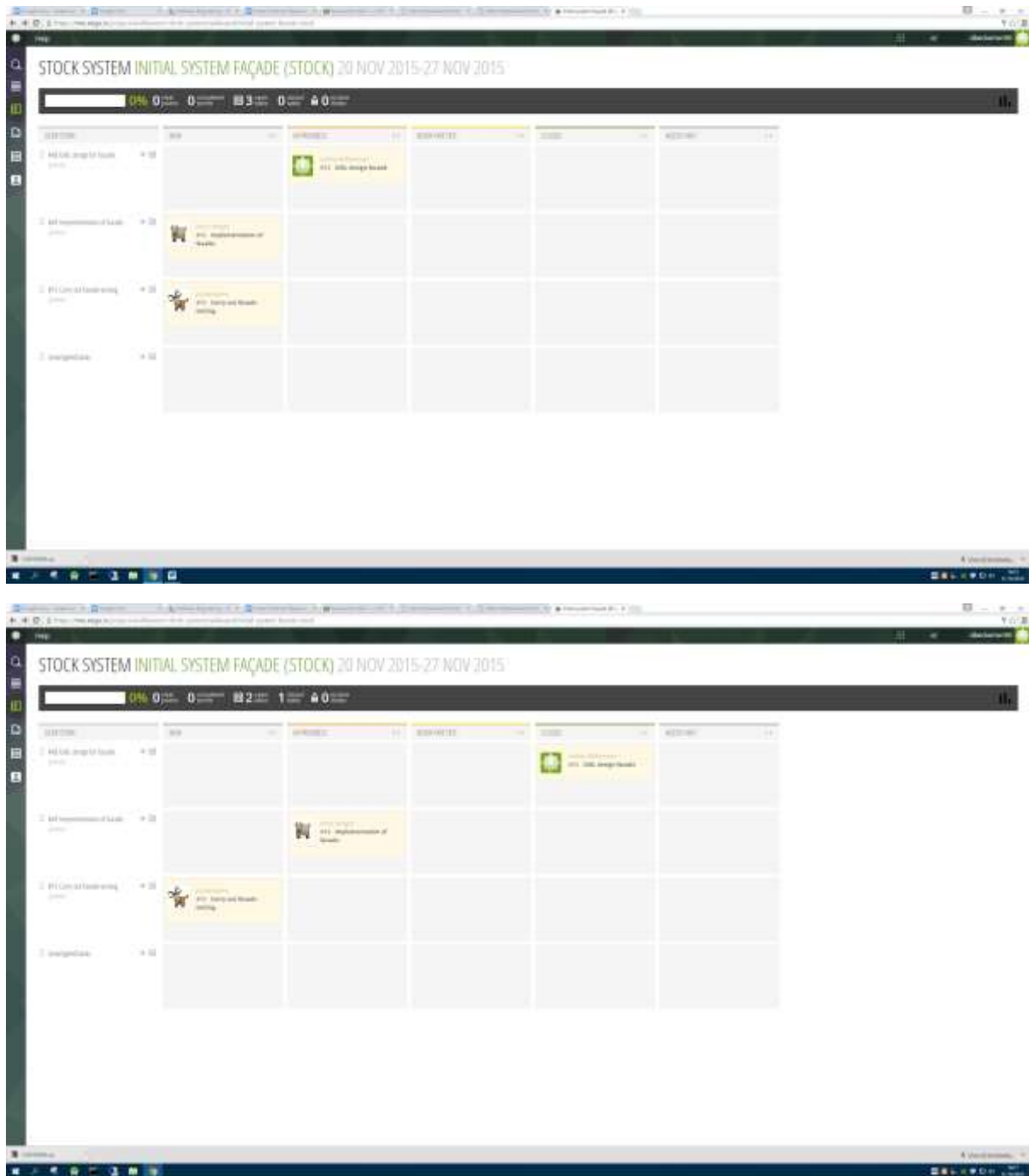
1.6 Sprint One

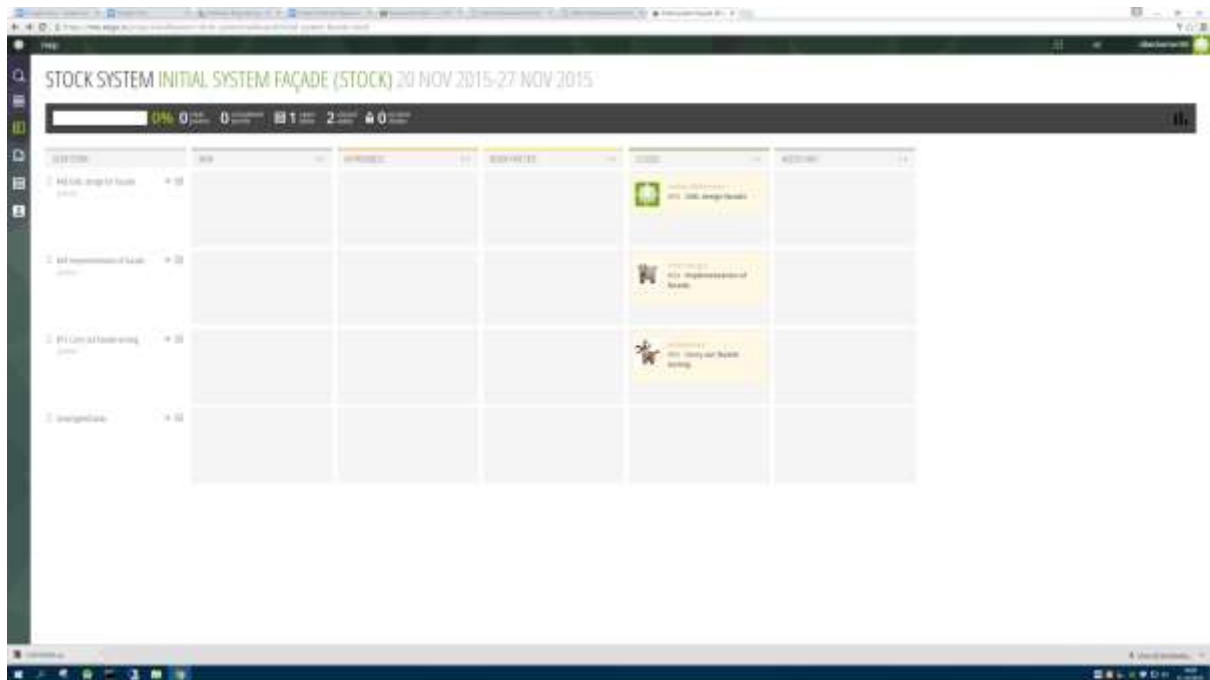
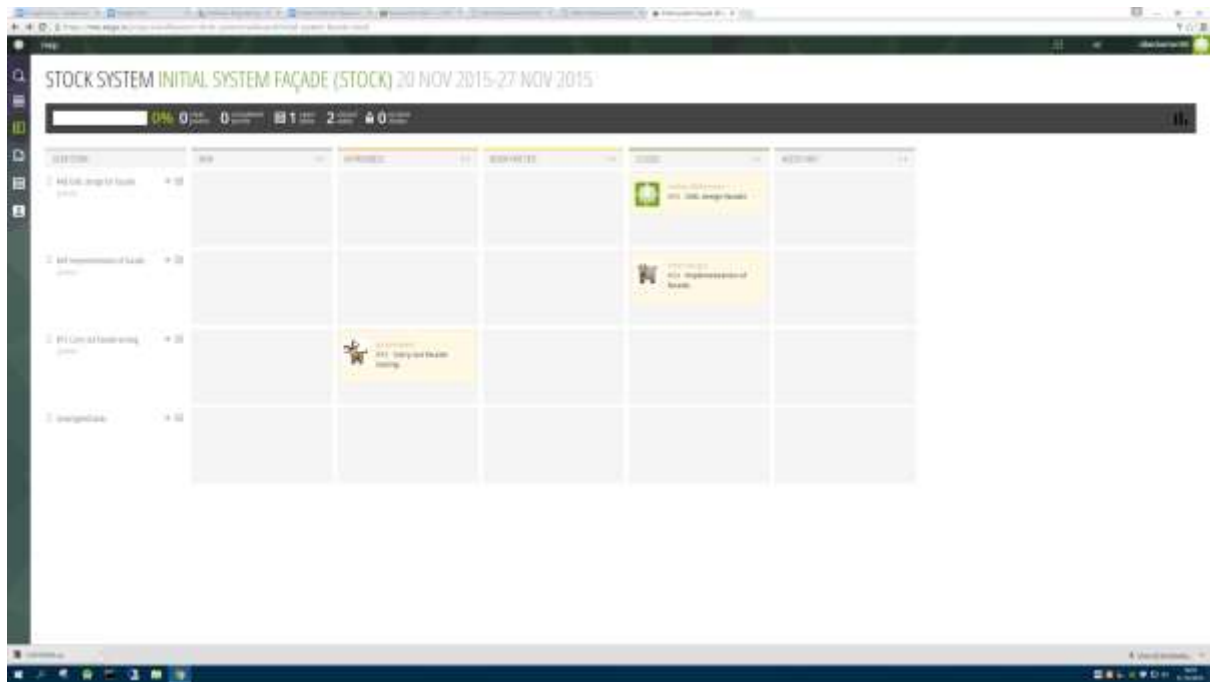




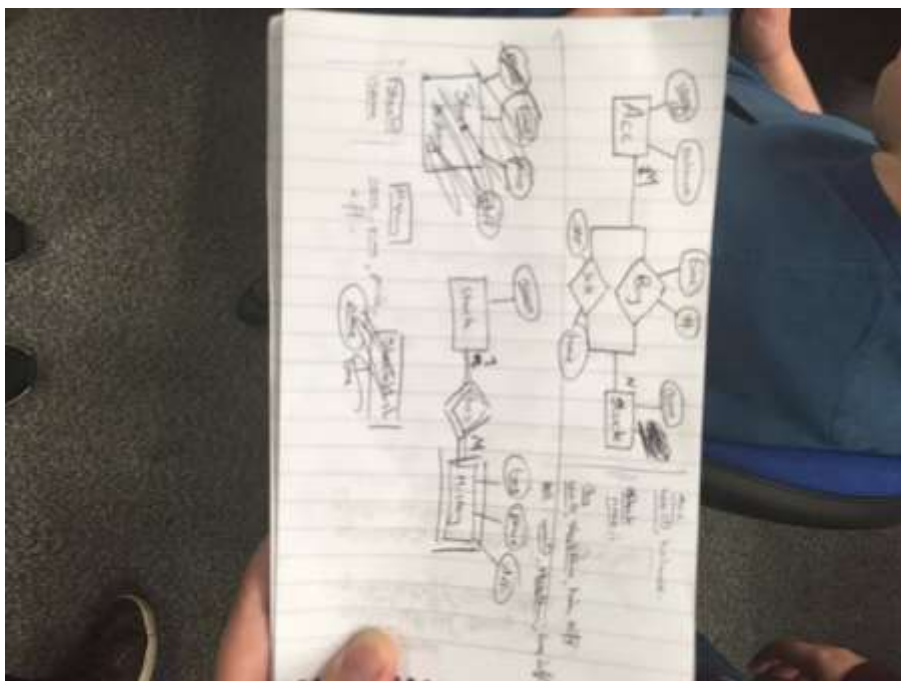
Sprint Two

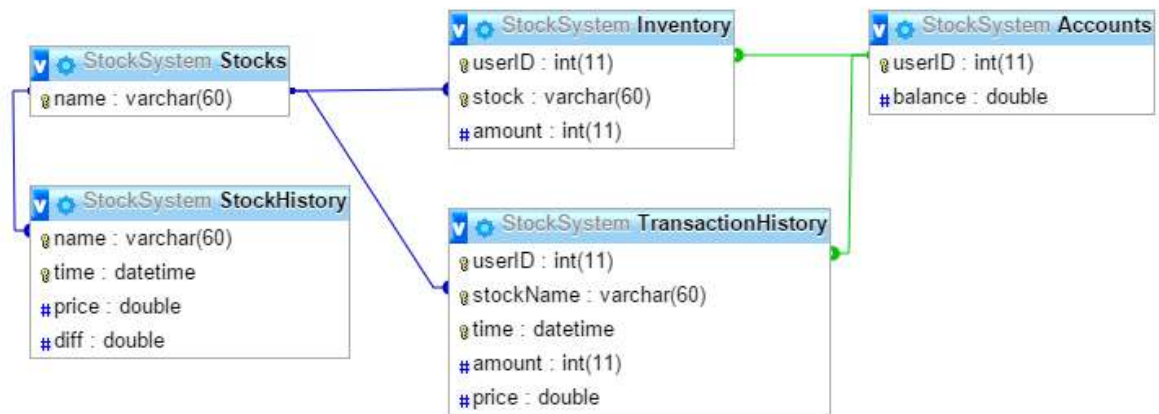


Sprint three

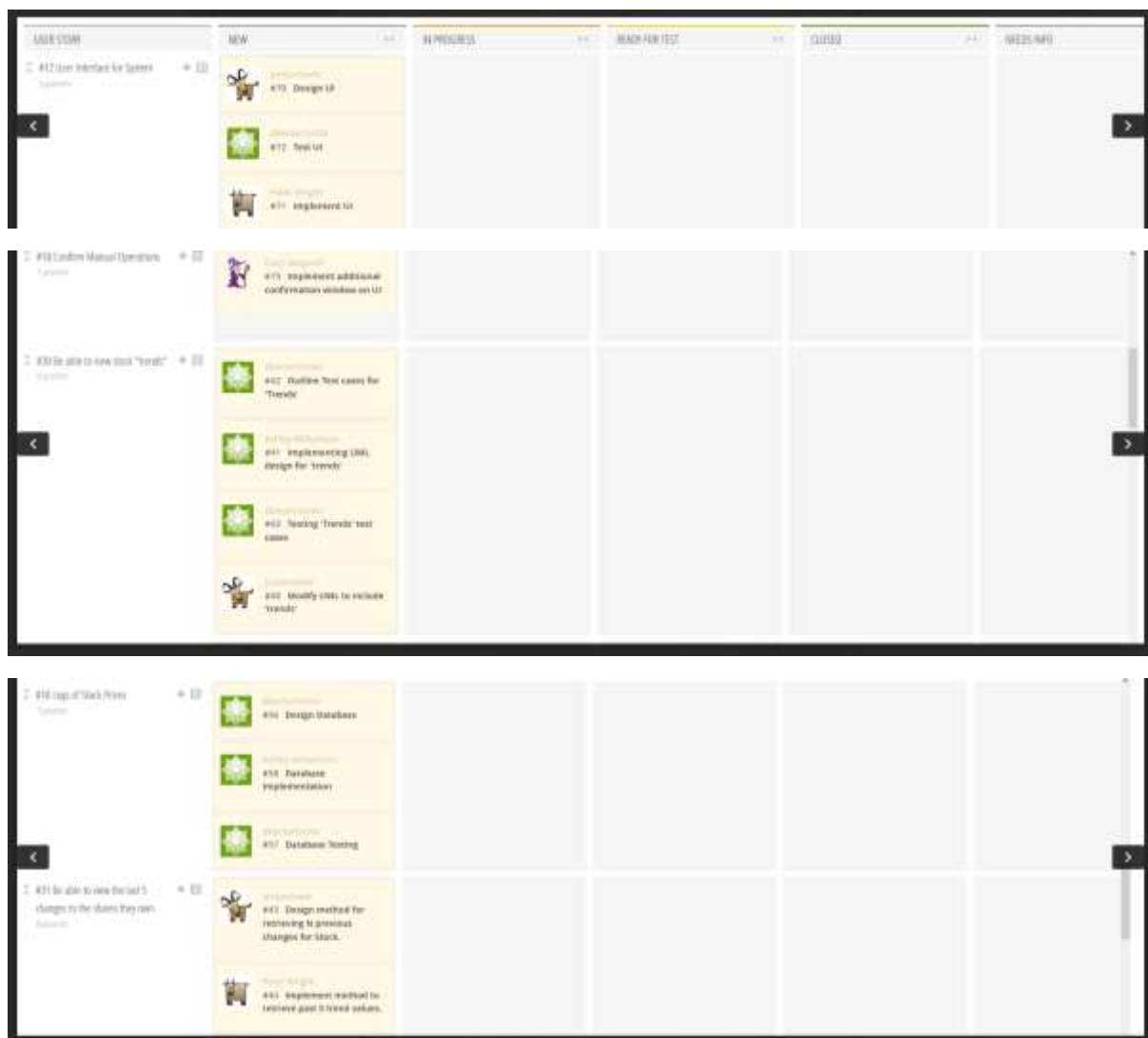


Sprint four






START










USER STORY	NEW	IN PROGRESS	READY FOR TEST	CLOSED	NEEDS INFO
2. #12 User interface for System System	 #12 Test UI	 #17 Implement UI		 #18 Design UI	
2. #18 Confirm Manual Operations System	 #18 Implement additional confirmation window per UI				
2. #18 Be able to view stock "trends" System	 #13 Testing "Trends" test cases			 #12 Outline test cases for "Trends"  #11 Implementing UML design for "Trends"	

2. #12 Log of Stock Prices System	 #17 Database Testing	 #16 Database Implementation		 #12 Modify UML to include "Trends"	
2. #11 Be able to view the last 5 changes to the shares they own System	 #17 Perform testing for 5 previous changes  #16 Implement method to retrieve past 5 trend values			 #13 Design method for retrieving 5 previous changes for stock  #14 Outline test cases for 5 previous changes	

2. #11 Manual Buy/Sell Stocks System		 #13 Implement StockHelper Facade		 #12 Redesign StockHelper Facade	
2. Log all user interactions System		 #14 Implement SQL Connector  #11 Implement SQL helper facade		 #14 Design SQL helper (Facade)	

END

USER STORY	NEW	IN PROGRESS	READY FOR TEST	CLOSED	NEEDS INFO
2. #12 User interface for System System	 #12 Test UI	 #17 Implement UI		 #18 Design UI	
2. #18 Confirm Manual Operations System	 #18 Implement additional confirmation window per UI				
2. #18 Be able to view stock "trends" System	 #13 Testing "Trends" test cases			 #12 Outline test cases for "Trends"  #11 Implementing UML design for "Trends"	

#10 Log all Stock Prices System					<div> <div> <div>Task Name</div> <div>#10 Modify UI to include trends</div> </div> </div>	
					<div> <div> <div>Task Name</div> <div>#16 Design Database</div> </div> </div>	
					<div> <div> <div>Task Name</div> <div>#17 Database Testing</div> </div> </div>	
					<div> <div> <div>Task Name</div> <div>#18 Database Implementation</div> </div> </div>	
#11 Be able to view the last 5 changes to the shares they own System	<div> <div> <div>Task Name</div> <div>#17 Perform testing for 5 previous changes</div> </div> </div>				<div> <div> <div>Task Name</div> <div>#15 Design method for retrieving N previous changes for Stock</div> </div> </div>	

	<div> <div> <div>Task Name</div> <div>#14 Implement method to retrieve past 5 trend values</div> </div> </div>				<div> <div> <div>Task Name</div> <div>#16 Outline test cases for 5 previous changes</div> </div> </div>	
#11 Manual Buy/Sell Stocks System					<div> <div> <div>Task Name</div> <div>#12 Redesign StockHolder Facade</div> </div> </div>	
					<div> <div> <div>Task Name</div> <div>#13 Refinement StockHolder Facade</div> </div> </div>	
#14 Log all past transactions System					<div> <div> <div>Task Name</div> <div>#19 Implement SQL Converter</div> </div> </div>	
					<div> <div> <div>Task Name</div> <div>#11 Implement SQL helper Facade</div> </div> </div>	

Sprint Five

USER STORY	NEW	IN PROGRESS	BACKLOG TEST	CLOSED	NEEDS INFO
#12 User interface for System System	<div> <div> <div>Task Name</div> <div>#12 Test UI</div> </div> </div>	<div> <div> <div>Task Name</div> <div>#11 Implement UI</div> </div> </div>		<div> <div> <div>Task Name</div> <div>#15 Design UI</div> </div> </div>	
#13 Be able to view stock "trend" System	<div> <div> <div>Task Name</div> <div>#13 Testing "trend" test cases</div> </div> </div>			<div> <div> <div>Task Name</div> <div>#13 Outline Test cases for "trend"</div> </div> </div>	
				<div> <div> <div>Task Name</div> <div>#17 Implementing UIML design for "trend"</div> </div> </div>	
#11 Be able to view the last 5 changes to the shares they own System	<div> <div> <div>Task Name</div> <div>#17 Perform testing for 5 previous changes</div> </div> </div>			<div> <div> <div>Task Name</div> <div>#15 Modify UI to include trends</div> </div> </div>	
				<div> <div> <div>Task Name</div> <div>#15 Design method for retrieving N previous changes for Stock</div> </div> </div>	

#11 Be able to view the last 5 changes to the shares they own System	<div> <div> <div>Task Name</div> <div>#17 Perform testing for 5 previous changes</div> </div> </div>			<div> <div> <div>Task Name</div> <div>#15 Design method for retrieving N previous changes for Stock</div> </div> </div>	
	<div> <div> <div>Task Name</div> <div>#14 Implement method to retrieve past 5 trend values</div> </div> </div>			<div> <div> <div>Task Name</div> <div>#16 Outline test cases for 5 previous changes</div> </div> </div>	
Outline Manual Operations Task	<div> <div> <div>Task Name</div> <div>#19 Implement additional confirmation window on UI</div> </div> </div>				
Unassigned tasks					

CLOSED