Nicholas Venis
Software Engineering
CMP3111M
Assessment Item 1

Preface:

In this module the effectiveness of SCRUM development has been demonstrated greatly and is an immense resource to learn about with direct insight to industry. Although throughout the project we faced challenges within the team and difficulties with the artefact itself given enough time to overcome and possibly under different circumstances the artefact would have been a fully capable dummy system. Personally I believe that given the challenges we produced an amicable attempt at the artefact and have further developed the skills needed to collaborate as a team in a "real world" development environment. We decided like every good piece of software we needed a codename to work under. I chose the name Pegasus as it provides connotations of being hardworking and wise. I look forward to taking practices learnt is this module further into my professional career and working life.

[LO1] synthesise concepts derived from current theories of advanced software engineering

---

PEGASUS ITERATION 1.0

Sprint 1:

Length of Sprint: 3 Weeks (17/09 – 31/09)

Feasible Sprint Goal: Connectivity between server and client using threading

Story point Value: 3

Non-functional Requirements:

- The server and client must have low latency between transmissions.

Summary of Daily Scrum:

- Working socket code and connection to the local server.
- Research java development principles and syntax, make sure scrum members understand working environment.
- Implement threading to deal with client requests and server data transfer.
- Familiarise ourselves with scrum methodology including terminology and agile development principles.
- Show evidence of sprint including updating user stories and producing documentation.
- Recognising team strengths and weaknesses and applying relevant people to relevant roles to boost productivity.
- Testing the newly implemented sprint goal before release

Product owner present.

Sprint Backlog:

Features from Product backlog	Tasks for Sprint Backlog	17/10/2017 (hours spent)	24th (hours spent)	31st (hours spent)
A user wants to connect to stock market server.	Research Java syntax and methods including how to implement sockets/multithreading.	1.5	1	0
	Create a test platform to practice socket design.	1.5	0	0
	Create a test platform to implement asynchronous design principles.	0	1.5	0
	Combine both test platforms to create a single platform that connects to a dummy server asynchronously	0	0	2
	Test the finished connection platform	0.5	0.5	0.5

Pair Programming Log:

Date	Time Started	Team members	Activity	Errors Spotted	Comments
17/10/17	15.30	Nick/Rob	Creating dummy socket test platform	0	Platform working correctly.
24/10	15.00	Rob/Nick	Creating dummy asynchronous test platform	0	Platform working correctly.
31/10	15.00	Nick/Rob	Combine platforms to create system that connects to dummy server in a thread	0	Implemented the feature that connects the client to the dummy server.

					And Successfully tested it.

Sprint Test logs:

Test	Expected Result	Actual Result
Socket started and connected	Visible connection on both dummy server and client console.	No errors. Both connections visible.
Thread initialised correctly	Thread starting without errors	Thread started without errors.
Platforms work when combined	Thread is initialised successfully without errors and connection is created via socket contained in thread.	Thread starts and connection is visible in client console and on server.

Sprint Retrospective:

Product owner present.

Scrum team:

- Sprint goal completely completed
- Sprint goal was feasible in the time period
- Team not overworked
- Goal was limited in scope however this prevented overworking the team with an unfeasible goal
- Skills and knowledge of java and syntax greatly increased
- Pair programming spotted small errors in code such as missing semi-colons
- More time on planning needed
- Sprint took longer than expected due to team unfamiliarity with programming language and new concepts
- Information learnt in this sprint carries over well to next sprint.
- User roles spread out not well enough, too many resources were spent researching java whereas could be used more in the design of the system
- Artefact progressing in line with project timeline

Product Owner:

- Team progressed well with their understanding of java and the development environment
- Happy with progress made as test system works without errors
- Project on schedule and team is working at a steady pace

- Team is not overworked and ready to start the next sprint

In this initial sprint I believe our understanding of both Java and the Agile development process has greatly improved. The artefact is in its infancy but shows promising signs as features that represent a good framework of the end product are starting to develop. The core implementation of both multithreaded code and sockets work in tandem perfectly. Pair programming has also shown its benefits this sprint and has allowed errors such as missing semi-colons and other small, non-syntaxical errors to be spotted, therefore not wasting time. The finished artefact looks to be achievable within the timeframe if the team continues to work at this level, which I also believe to be sustainable.

Moving forward there are a few issues I'd address within this scrum team. The first is language barrier; half of our team are non-native English speakers and so language barrier has become an issue. I believe that the confidence in their English skills has also stopped them being as confident in the team and therefore are not as willing to participate. Taking this into a further sprint, more needs to be done to break down some of the potential language barriers to get the full team working to potential. I also believe prior to the sprint a lack of planning and design caused progress to be a little bit slower than expected. In future sprints more emphasis needs to be put onto the planning and design aspects before the implementation as it gives the team more direction and a clearer understanding of the end goal. Finally, I believe that user roles need to be spread further and not just awarded to those who like to code for instance; randomising roles could also keep team morale high as work isn't repetitive but could also force team members out of their comfort zones and to engage more.

PEGASUS ITERATION 1.1

Sprint 2:

Planning:

- duration: 1hr
- Start Time: 15.00
- date: 21/10/2017

Sprint Started:

- Time: 16.00
- Date: 21/10/2017

Sprint Finished:

- Time: 17.00
- Date: 06/12/2017

Feasible Sprint Goal: Implement logging features to test client with documentation

- implement error log feature
- Fix array index out of bounds when using cash command error.

- Have use case diagram for system done
- have class diagram for system done
- have sequence diagram done
- have server running locally

Non-Functional Requirements:

- Error logging should have no discernible effect on performance of client and server

TIME:

- 3 weeks

Story Point Value: 3

Summary of Daily Scrums

Daily Scrum Meeting 1:

Roles and Focus:

- ScrumMaster = Patrick
- Coding = Nick(lead), Patrick(secondary) (Get server working locally)
- Design = Xu, Yuan, He – UML (class/sequence diagram)

- Get a local server host running as a test platform
- Implement a logging feature to use for error checking and testing
- Design UML and class sequence diagrams for the client for use next sprint

Daily Scrum Meeting 2

Role and Focus:

- ScrumMaster = Patrick
- Coding = Nick(lead), Patrick(secondary) (Fixing cash error in server)
- Design = Xu, Yuan, He, Rob – UML (sequence diagram)

- Fix errors within the cash command, possibly located in another function causing a knock on effect
- Continued work on documentation.

Daily Scrum Meeting 3:

Role and Focus:

- ScrumMaster = Patrick
- Coding = Nick(lead), Patrick(secondary) (implementing)
- Design = Xu, Yuan, He, Rob – UML (sequence diagram)

- Getting the error logging feature implemented for the client and server platforms

- Finishing work on UML documentation

The programming team focussed on implementing the error log feature, whilst the design team continued their work on the UML design documentation.

Product owner present.

Sprint Backlog

Features from Product backlog	Tasks for Sprint Backlog	21/10/2017	28th	05/12/17
A user wants to buy shares from the stock market.	Create local platforms of stock market server and client	2	0.5	0
	Design buyShare sequence diagram	1	0	0
	Design sellShare sequence diagram	1	1	0
	Attempt to fix indexing error in cash command	2	0.5	0
	Implement error log feature	2	2	0
	Test implementations	0.5	0.5	0

Pair Programming Logs:

Date	Time Started	Who's Involved	Activity	Errors Spotted	Comments
21/11/17	16.00	Nick, Patrick	Running stock server locally	1. Index out of bounds error (shown below).	
28/11/17	15.00	Nick, Patrick	Fixing Cash function in server		Attempted to fix error log function
05/12/17	15.00	Nick, Patrick	Implementing error log Function		

```
Client: /127.0.0.1:5000 : CASH
Exception in thread "Thread-1" java.lang.ArrayIndexOutOfBoundsException: 1
        at ClientConnect.run(ClientConnect.java:139)
Stock Market updated each 15s.
UPD:3I_GRP:459.36:-14.39
```

Test Logs

Test	Expected Result	Actual Result

Server runs locally	Server runs and allows for localhost client connection	The server ran, allowing our test client to connect successfully.
Log feature	Log updates with each change or display of stocks. Log also logs anything the user does as well as their IP address and registration.	Log is created and data is stored correctly
CASH function works	CASH function returns how much money a user has to invest.	It doesn't return that successfully.

Sprint Retrospective:

Product owner present.

Scrum team:

- Spring goal completed
- Sprint goal proved feasible within the sprint period
- Team was fairly overworked due to lack of team members
- Smooth transition of previous sprint to current.
- Planning stage of sprint took longer and as a result was more comprehensive
- Due to more planning more tasks were completed and the team were more focused as they knew what they had to be doing
- Team worked together more effectively than last sprint, could be due to knowing each other more
- Both design tasks were completed
- Error log feature proved useful diagnosing the issue with the cash command
- Artefact progressing in line with project timeline
- Even though the cash error was not fixed, an issue was created on GitHub to report back to the team creating the server platform, this should be fixed by next sprint.

Product Owner:

- Artefact is coming along to an expected standard if a little bit slowly
- Progress on the artefact is promising and looks to meet deadlines
- Team worked better together this sprint than last
- Team seems to be a little bit overworked
- More team engagement but not at potential

In this following sprint I believe the artefact has progressed, however not at the rate the team hoped for. This sprint we were struck with many bugs and implemented features necessary for correcting them. Although this did take programmer time away from the main features this time was small and necessary to move on. The artefact itself contains several features not seen in the previous sprint that are fundamental to the finished system. This is especially true with both client and server communicating between one another with commands rather than simple parsing of data. Certain commands such as HELO and REGI work perfectly however an error in the indexing of registration

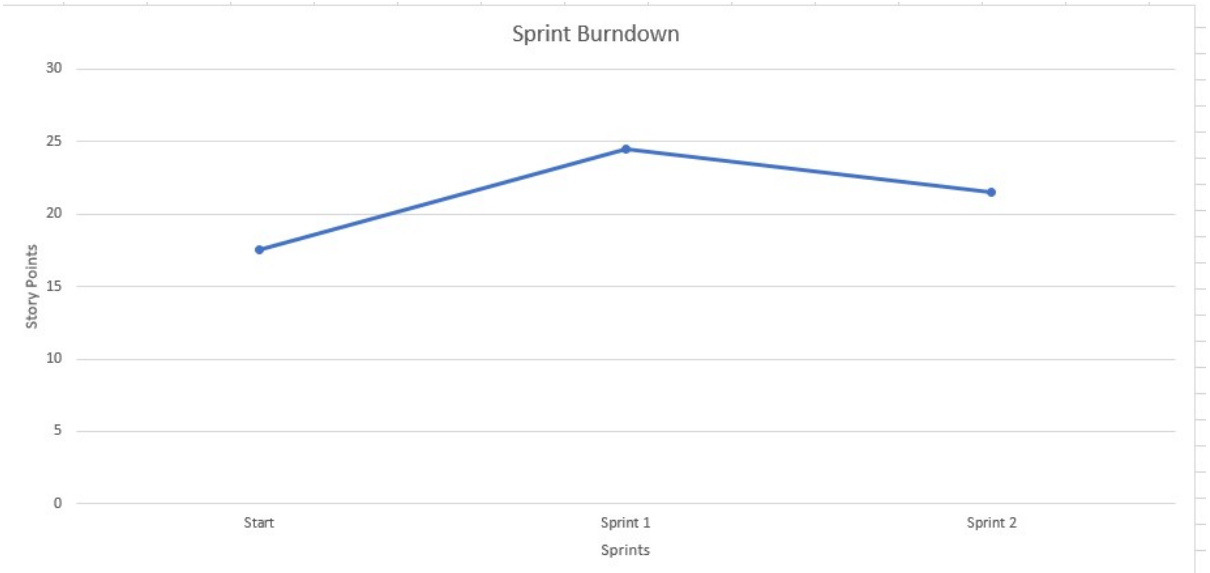
data is causing big issues for the team, this error is also present across multiple features. To remedy this an issue request has been sent to the open source server developer and we are currently waiting on a bug fix. If this bug fix is not prompt the team will work more on diagrams and planning for further sprints.

This combination of errors in the open source software and bug fixing has caused this sprint to last longer than expected. The artefact however does show promise as much of the core implementation of features is present. Once we have received a bug fix in the open source server progress can continue as before. A further issue in this sprint was once again team members being missing or not participating fully, although more than last sprint. This definitely delayed the sprint as work was spread to only a couple of people for most days and so were overloaded in their responsibilities. This level of work is not sustainable and these people will burn out before the sprint is finished.

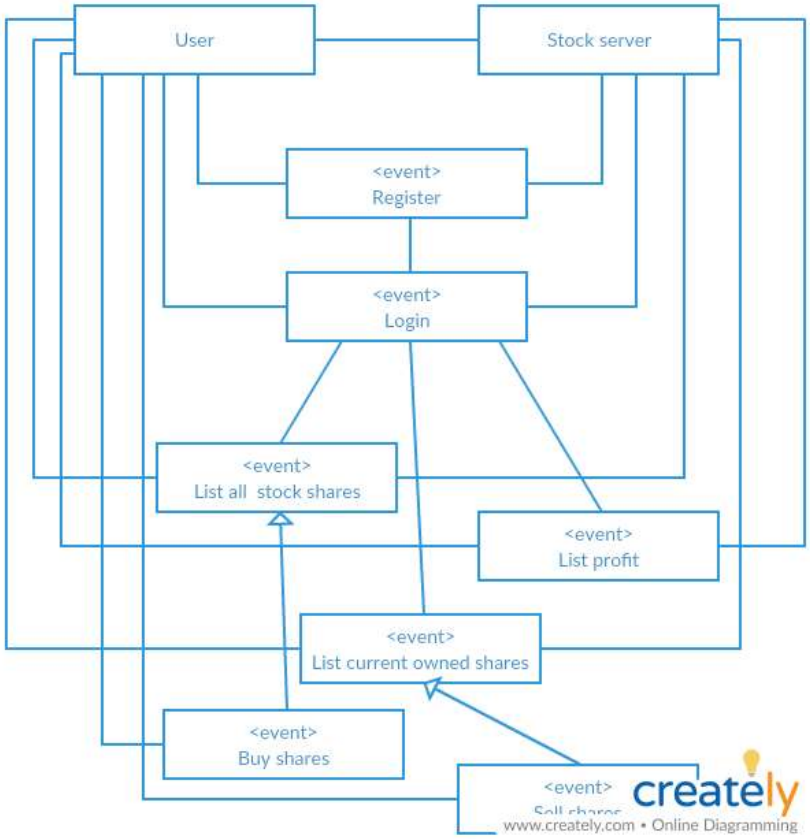
Over the last sprint I believe a number of important changes have occurred that improved the quality of the current sprint. Firstly, planning and design played a key role in each team members vision of the final product. This planning and design allowed each team member to be on the same track and as a result work was more driven. Following this as each member worked on something different and possibly out of their comfort zone work was more productive with team members participating more. However, this sprint took longer than the previous, this wasn't due to the goal being larger but more limitations within the team such as participation of members slowed progress and encountered errors.

Continuing into the next sprint the language barrier and team participation once again needs addressing as the artefact will fall behind schedule. More needs to be done to get the outliers to participate borderline forcing them to work as part of the team. A scrum team should not consist of 2 members as it puts too much strain on those in question and forces them to double up on roles. In my opinion having a few members of the team collaborating once or twice is no better than not collaborating at all. The level of planning and use of design this sprint was ideal and needs to be carried into the next sprint, this can also be said with testing the platform. The user roles were correctly spread this sprint and it reflected with much more design and documentation being produced, this also needs to be carried forward to next sprint.

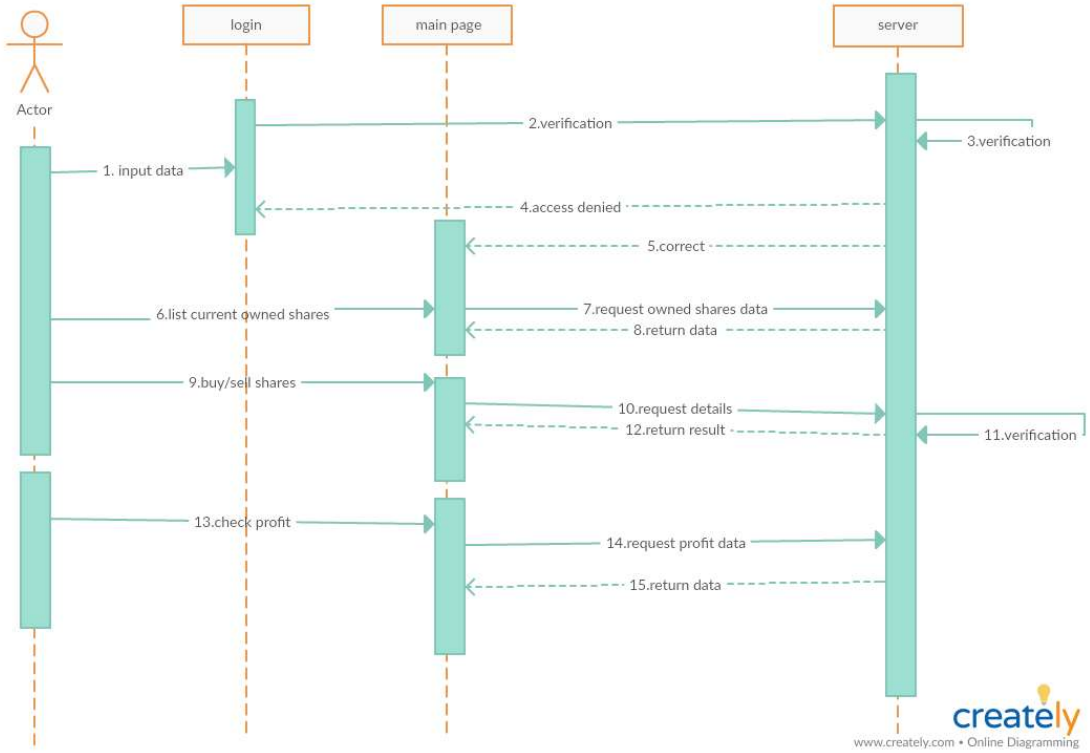
Sprint Burndown Chart:



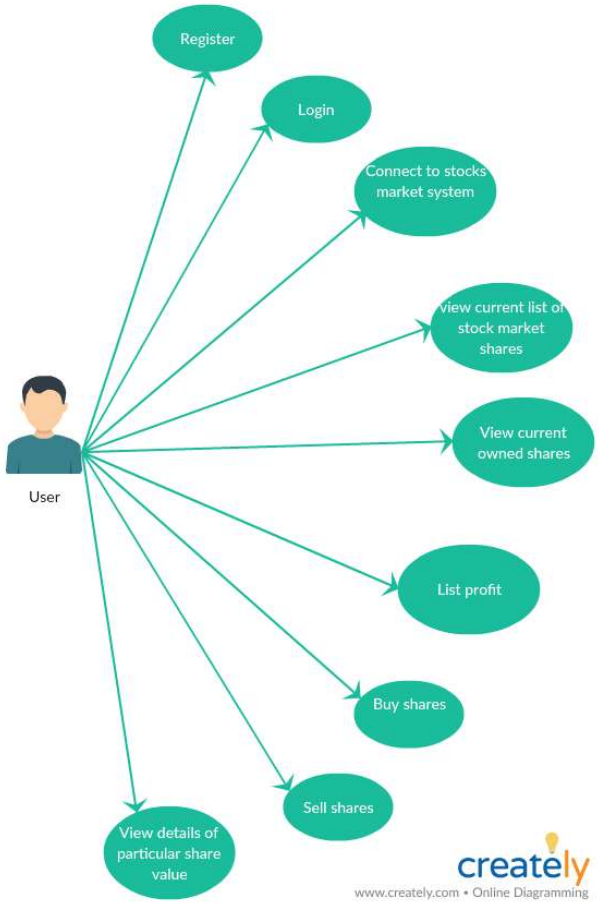
Domain Modelling of the Artefact:



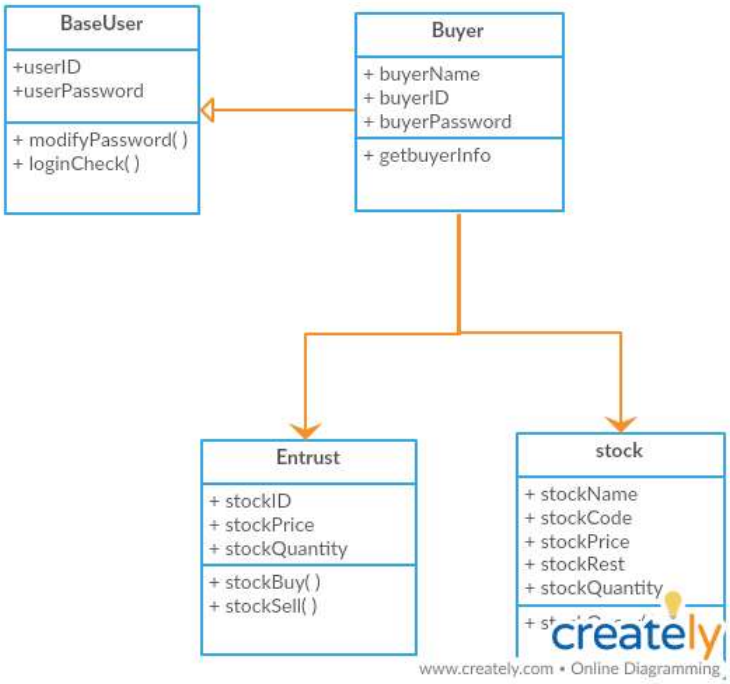
Sequence Diagram for the Artefact:



Use Case Diagram of the Artefact:



Class Diagram of the Artefact:



[LO2] analyse the empirical nature of software engineering and the application of empirical methods in software engineering development.

The scrum methodology first appeared in a Harvard business school journal article in 1986 by creators Hirotaka Takeuchi and Ikujiro Nonaka. The original idea was to create a system that improved over previous development routines of the time. Scrum was designed to increase the flexibility of the project whilst also increasing the speed of which the project took place. The authors compared the development process to rugby whereby multiple people were working on overlapping features and phases akin to a rugby scrum, where the methodology gets its name. The scrum methodology would then go on to be written about by various authors and by 2010 an official guideline for Scrum development was released aptly named “The Scrum Guide”. Scrum is an iterative process meaning that the process repeats many times over the development of the artefact and uses an Agile framework for software development.

Scrum development has a number of roles that can be assigned to team members. Scrum teams are usually between 3 to 9 persons in size with multiple teams working on the overall platform. The product owner represents those funding the project and the end user of the system, this role is imperative in the platform being economically viable and providing direct feedback to those invested in the project. Communication is a key part of this role with the product owner steering the scrum team into the correct direction in respect to the outcome of the artefact.

The next role is that of the development team. This role encompasses multiple elements such as programming, design and testing. There is no hierarchy in this team and is therefore self-organising, this helps alleviate power struggles and aids collaboration. The third role is Scrum Master. The Scrum Master is responsible for overcoming obstacles the development team might face to make sure the team meets its goals. This is not akin to a traditional managerial role as the Scrum Master acts as a way to alleviate problems rather than dictating to lower level employees. The Scrum master is also responsible for the team following the scrum development methodology and not deviating. A final responsibility is leading meetings such as the scrum meeting or scrum retrospective.

So what is actually involved in Scrum development? Scrum is a complex process that is best explained by decaying the process into categories. Workflow is one of the most interesting parts of the scrum methodology. Whereas in waterfall each step of development follows a linear path with little to no regression agile development follows an entirely different process that is iterative rather than linear which does allow for regression. In agile development the basic unit of development is called a sprint.

A sprint is finite in the fact is has a time cap and deadline for completion. This deadline is not flexible and needs to be adhered to. The duration of a sprint is predetermined during the planning of a sprint and is created based on how much work is deemed to be needed to complete the task. For each sprint one item is chosen from the product backlog. The product backlog is a list of features desired in the final artefact. Each feature is given a small description and is prioritised based on its importance in the final product. To create the product backlog the product owner will sit down with the scrum team at the conception of the project and list every feature they desire. The product backlog morphs throughout the development cycle to encompass more than just features. By the end of the project the backlog could contain bug reports, knowledge acquired and work on the product that is non-functional (technical work).

However, one issue that arises out of this is what has been nicknamed the “Santa’s list” of project features. This problem arises when every single thing that anyone in the scrum could ever think of to do with the project is written down as a feature. The backlog is no longer a product backlog and more a database of requirements for the project. This makes the features categorically harder to prioritise and severely limits how much the product can grow and evolve over the development period. This in essence turns the development into a linear struggle that isn’t shaped too much by product owner feedback. To circumvent this the most critical parts of the system should be considered and put into the backlog, the rest should be discarded.

As a continuation of this priority of items in the product backlog needs to be discussed. Items in the backlog need to be sized up and prioritised to enable sprints to have realistic deadlines that do not overwork the team too much. This quantisation of effort is recorded as story point values. Story points are abstract relative measures of the effort and size needed to complete a task in the product backlog. Story points are needed over absolute measures of time as the development process is constantly evolving. For instance, the team learns some new knowledge that indicates an item in the product backlog is now going to take longer than expected. A story point value is easily changed based on the item size and how the team perceives it, this flexibility based on the evolving of the product means we have a measure that is quick and easy to use. Story points use non-linear increments denoting sizes of items to prevent discussions of what is the “right value” during the planning phase. As the story point estimates are relative to the items in the team’s current backlog they cannot be compared across teams. This works in conjunction with real time estimates of items in the product backlog to formulate deadlines.

Story points represent a great was to quantify effort needed for an item but are not enough alone. A gamified technique exists in agile development called “planning poker”. When creating story point values, a large part can be played by anchoring. Anchoring is a cognitive bias that means that someone will latch onto the first piece of information offered and use it (the anchor) to compare future decisions to. When creating story point values, the anchor can skew the value of story points as the person does not see the full scope of the project. The product owner will explain each item to the team members and give their definition of when they consider the item to be complete. Each team member has a set of cards with a values representing their estimate of the story points needed for the item. Each team member picks a card relevant to the size of the item just described and places it face down. At the same time all cards are shown and the two estimates furthest apart explain their reasoning why. This process of drawing cards is repeated until a consensus has been reached between the team (Cohn, 2012).

After items have been sized up appropriately they can be chosen for as a sprint goal. However, prior to any sort of work taking place a sprint has to be planned. Sprint planning allows the team to plan its work accordingly and therefore achieve some sort of organisation during the sprint. If the team has a plan to follow and is organised work will be massively more productive than without. As the development cycle continues planning may become more in depth as the team learns more of any problems that may arise during a sprint and plan for them. The first part of planning a sprint is having a meeting between the team.

This meeting is referred to as a stand-up meeting or daily scrum. The daily scrum allows the team to concatenate their work for the day. Ideally these meetings are held in the same location and time, most likely in the morning so that the team knows what they’re doing that day. Each meeting is time boxed (restricted) to 15 minutes to not become a tiresome meeting (Cohn, 2017). This meeting is informal and little to no preparation including slides should be prepared. All team members are required to attend the meeting. The product owner is encouraged to attend these meetings as they

can give further information if the team has any questions and can also see first-hand how the team is progressing with the artefact.

During the meeting each member stands up and cannot sit down, this is done to keep the meetings short as people are not willing to stand for extended periods of time. During the meeting each team member answers 3 questions; “What did you do yesterday? What will you do today? Are there any problems in your way?”. By understanding what each team member has contributed and will contribute a better understanding of what has left to be accomplished can be had. This also had the added benefit of being able to compare progress to yesterday’s stand up meeting. Any problems a team member faces becomes the ScrumMaster’s responsibility to sort in order for productive work to resume (Sutherland, 2017).

Further planning of a sprint also has to be undergone. For instance, one item from the product backlog should be picked by the team based on priority. Planning aims to create a sprint backlog showing all work that needs to be done during the sprint and create an estimated deadline for the completion of the sprint goal using story point values. The sprint backlog comprises all tasks needed to be completed for the team to reach the sprint goal. Following this planning especially with the sprint goal this can lead to grooming and reorganising of the product backlog as tasks are moved about.

Sprint goals should not be too large in size in order to maximise productivity. Sprint development is ongoing and there are no breaks between sprints. Therefore, a goal should be able to be completed during the allocated time without overworking the team. Developing an artefact can be compared to running a long distance race, if you want to finish the race you need to keep a steady pace. Pressuring the scrum team to do more work during a sprint can lead to the team burning out. This may result in a short term jolt in the speed of development but this is unsustainable.

A sprint is an iterative process and therefore repeats many times before the product is finished, but what actually goes into a sprint? Sprints follow agile software development methodologies and have distinct characteristics. Scrum teams are cross functional therefore meaning members of the team are assigned different roles during a sprint. These roles change from sprint to sprint but allow the artefact to progress faster than if the team was all working on the same thing at once.

Agile XP or agile extreme programming is methodology used to improve software quality and allow the artefact to change based on feedback given. XP is continuous process whereby for each iteration code is tested rigorously. XP contains some core values that are applied in a sprint. The first is pair programming, in which programming is done in teams of 2. In this two programmers will work at one workstation with one driving (writing code) and the other navigating (reviewing code as its typed). These two roles are swapped frequently to avoid burning out. Logs are kept of each programming session with any bugs noted. Pair programming does have its problems however, for example with two people being assigned to one code base it takes longer to deliver the same code one person would, this therefore increases man hours and costs for the company. At the same time however code is of a higher quality and can be more efficient. It could be argued however the higher quality of the code and fewer faults does offset the man hours’ problem as they would be used correcting faults in code otherwise.

Another key value of XP is testing within each iteration. Unit testing is the processing of testing blocks of code rather than the code as a whole. This is done within the current iteration (sprint) and for the code developed during the iteration. Unit testing allows for flexibility and modularity of code

as each block has shown to work and so code refactoring can be used at a later date if necessary. Unit testing can also be used to build up documentation for each block of code eventually becoming documentation for the entire system. Testing logs are kept for each test of each unit and are recorded in the sprint. Unit testing also has its flaws, for example once the blocks have been integrated into the system further bugs could be created with two units working together to produce an error within the system. A further challenge of unit testing is creating tests to look for errors that would equate to real actions the end user would make, this problem of creating useful tests devalues any testing done.

Once this iteration(sprint) is over the team has another meeting to discuss the outcome and give an overview of the sprint, this is called the sprint retrospective. Sprint retrospectives allow the scum team to identify problems they had an in the sprint, the causes of these problems and what measures need to be taken to prevent these from happening in the future. The product owner has a large part in the sprint retrospective as it allows them to review the work being done and provide feedback on the work and any changes that need to be made. Constructively discussing what went well and what could be improved actively improves the quality of all future sprints as ideally the same mistakes would not be made twice. During the retrospective a demo of the features implanted may be shown to the product owner. This retrospective is also led by the scrum master. Not having the product owner present at sprint retrospectives can be problematic. For example, if the product owner missed meetings and returns to find the artefact sub-satisfactory work has to be regressed and valuable time is lost, this is a waste of time and effort for the team and is often disheartening.

Agile software development has many benefits over a more linear development process such as waterfall for instance. In our development process of our artefact we found the iterative process and testing to be of extreme usefulness. We found bugs in blocks of code that not be pushed system wide and integrated however dude to rigorous testing during the sprint these were fixed. A large part of the productivity of the group stemmed from the distribution of team roles however due to missing team members I often had to take on multiple roles. This completely conflicted with the ideals of scrum development when I had to be both product owner and scrum master and I personally believe this could have affected the outcome of the finished artefact.

Pair programming did however alleviate a lot of issues stemming from programming. A lot of the errors we as a team experienced were not code breaking however did reduce the quality of the code. During my time pair programming as both driver and navigator I believe the focus of code and quality was vastly increased as we encountered little to no errors as the navigator would always point out solutions and give advice on the direction the code was taking.

The benefit of unit testing during our project also can be very much seen in our sprint logs. We broke features down into code blocks and sub sprint goals with testing in mind. From this wider integration was more of a possibility and combined with GitHub merging and integration proved to be a big success. However, this also became a problem in sprint 2. In sprint 2 we found a bug we could not resolve within the confines of one sprint and therefore extended over the deadline. Scrum is an evolving process that is flexible, however this was core functionality and the next sprint couldn’t happen without it. This therefore delated future sprints and left us in a less than optimal position. In essence this was a failed sprint. Scrum luckily accounted for this and as part of the sprint retrospective we analysed what went wrong and how to combat this in future.

However, it can be argued that some scrum practices did indeed impair our development of the artefact. For instance, pair programming when team members are missing pulled man power from other aspects of the development process such as design causing a shortage and therefore less

work. This is a further critique of scrum as it heavily relies on everyone in the team pulling their weight in terms of work. A scrum team is akin to a precise machine; if one cog is out of line or missing the machine doesn't work properly and progress grounds to a halt. We found this to be a problem multiple times throughout development.

However, these problems we experience can be also outweighed by the good parts of our sprints. For instance, our daily scrums really set the team on the correct track and kept focus on the end goal of the artefact. They also served as a pseudo reality check of whereabouts we were on the product roadmap and gave a useful insight into what each member had been up to and what we could expect from them.

Breaking down items from the product backlog into smaller tasks in the sprint backlog also made tasks seem far more manageable and less gigantic and overbearing. This also aided team morale as goals suddenly seemed accomplishable and less daunting.

Compared to a linear methodology such as waterfall we definitely found this iterative process far more useful. I believe that if we did not use unit testing and instead build testing there would have been far more, program breaking bugs that would have resulted in failure. I also thing that the linear nature of waterfall promotes more stress as the end goal isn't broken down as much as in agile development and so the final goal with a final deadline seems a lot more daunting than several deadlines of far smaller units. This idea of smaller units also promotes a far more flexible development environment that relies on feedback and changes that ultimately promotes more quality in agile software than waterfall.

[LO3] utilise and evaluate advanced software engineering techniques and processes in the development of a software artefact.

A fundamental part of SCRUM software development is collaboration. Collaboration in this context can take many forms; for instance, code merging and pulling to some members of the team, sharing of diagrams to others. One common theme these all share however is a centre. Without centralised place where everyone can come together to collaborate the team would be in dismay due to disorganisation. If every member stored their work on a different medium or service every time the project called for collaboration or retrieval of work; differences in versioning or retrieval or files can cost valuable time and resources. This idea of centralising files and resources has existed in business practices and government for centuries. It was for this reason we decided to choose a centralised service to aid us in scrum development.

Many tools exist for this exact service with varying degrees of features and depth. Initially as a team we did not know what would be required from service specifically for scrum development. Some in the team called for a more familiar format such as google drive. This however lacked the specialist tools and layouts used specifically for scrum development. With guidance from the delivery team we were pointed towards scrumdesk.

Scrumdesk was completely unfamiliar with us and seemed to have a very steep learning curve. Although it contained many features that could be used it also presented itself as an information overload. Scrumdesk contained unfamiliar syntax such as "Epics" not heard within the context of scrum, this use of proprietary terms confused the team further. The website under proper training and with some documentation could have proved itself a powerful tool at our disposal however was just too complicated for the ideas we had in mind.

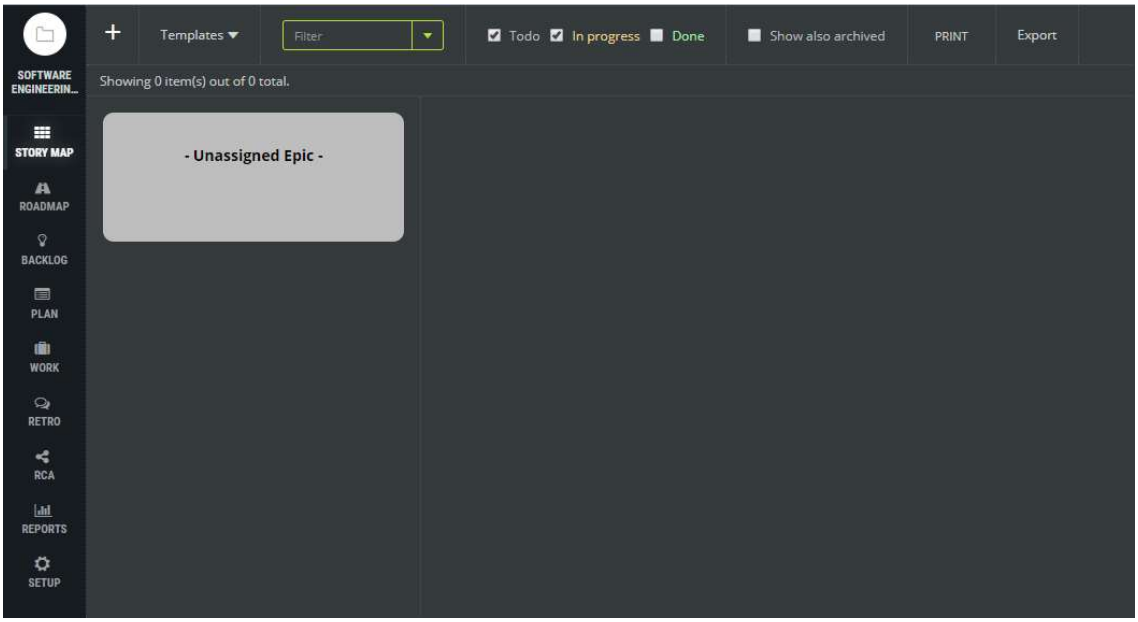


Figure 1: Scrumdesk home screen

Upon deciding to not use scrumdesk we began searching for more lightweight alternatives. We needed essential features for scrum but not the bulk of a more in depth solution. We found trello by searching for scrumdesk alternatives. Trello had the collaboration potential of scrumdesk without the added bulk. We could collate files and provide a board for discussion of the project. Trello also allowed us to keep a good track of the user stores and their stage in development.

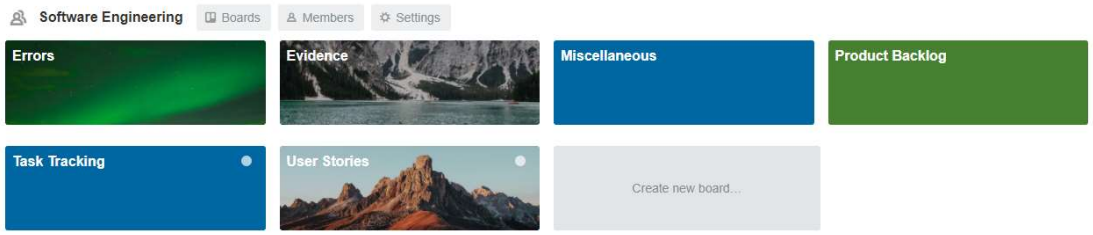


Figure 2: My home screen of Trello displaying all our group boards.

Trello represents each subject or area with a board. A board on Trello is very similar to a board on a forum whereby all talk on said board should adhere to one subject area. For our use we broke up the project into separate boards each relation to a core theme related to scrum or the assignment. This however is not part of scrum development and could be argue confuses things especially after arguing *FOR* one centralised resource. However, what this does allow us to do is to separate content into more manageable chunks to deal with. For instance, if you wanted to find the user stories in a centralised resource you would have to scroll past all the information not relevant to you to find the information that is, this takes up valuable time and adds unnecessary complexity.

Errors, the first board, was for us as a group to keep track of any errors encountered whilst programming. As each user is assigned different roles during a sprint the same person locating errors one week might not be programming the next. It is therefore important to provide one resource whereby the team can track errors. This can be used in connection with GitHub to create a multilayer system whereby issues requests on GitHub can be used to externally report large, platform breaking errors and smaller non code breaking bugs can be tracked and kept in house.

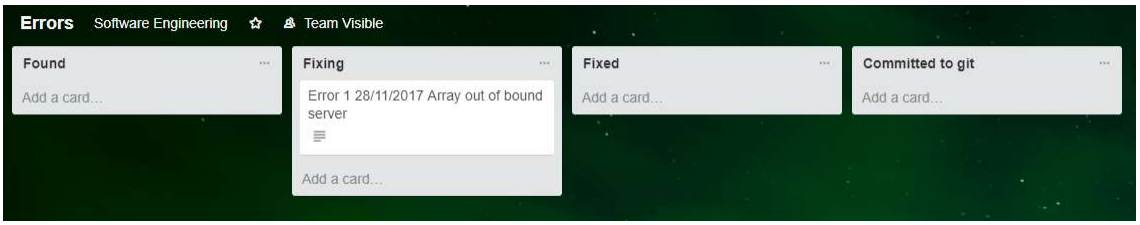


Figure 3: The error tracking board

Contained on the board itself is the single error we encountered whilst coding. This was with the open source server that we were also trying to fix. The bug was moved from found to fixing as during the last sprint we tried to fix the fault in the open source software alas to no avail. As the bug was code breaking we created an issue report on GitHub for the open source software.

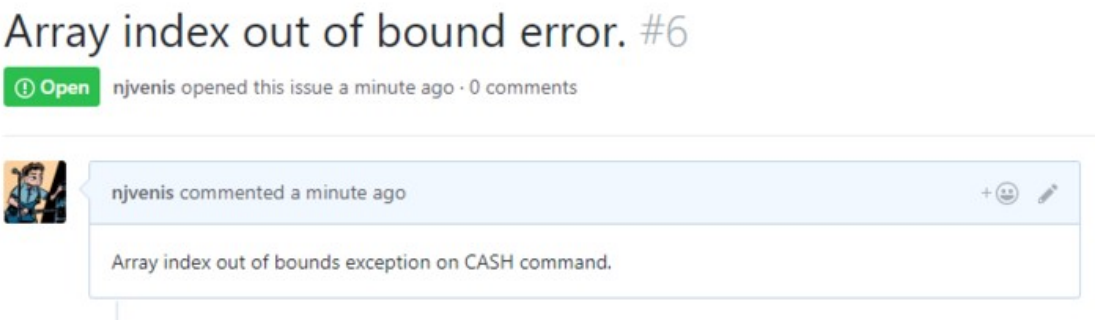


Figure 4: The issue report on the open source GitHub

Our next board was our evidence board. This board was used to track and store evidence in relation to the final report. This board also had 3 separate lists to store cards for tracking. This game the team a great idea of what needed to be done and allowed us to separate and store finished work. This board was mainly used to store finished diagrams to be later used in the report. It also came in very useful for creating sprint goals as we could identify what was finished and what needed completing. This board also served as a pseudo task board as many of the tasks needed to show evidence also are very similar to the scrum goals and documentation for the scrum.

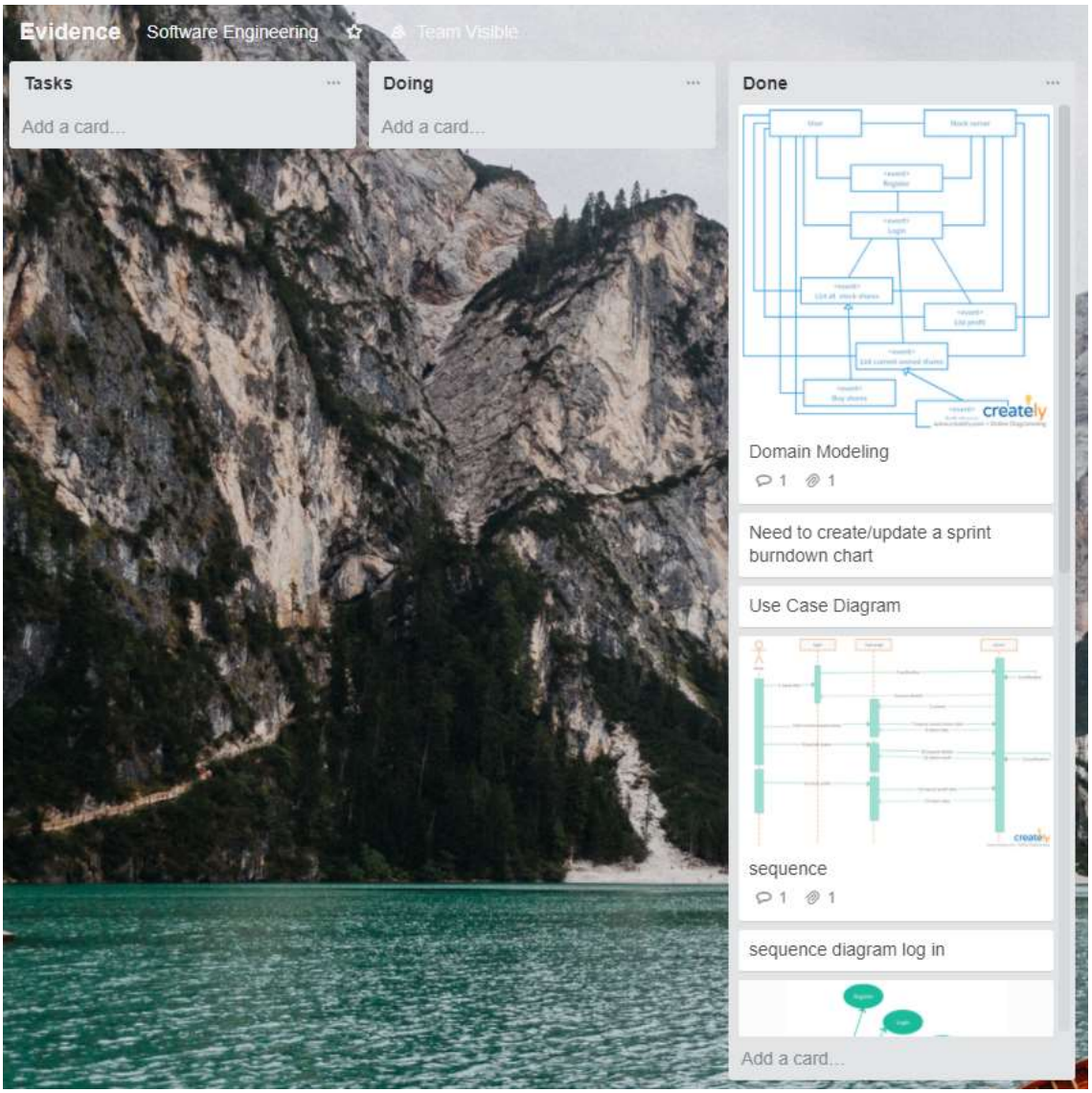


Figure 5: the Trello evidence board.

Our third board miscellaneous was only used as a way to keep together links or pieces of useful if not completely related data. We actually found that even though this was a tiny resource it proved very useful as it allowed team members to not waste time asking for links to software or data and so made the scrum lightly more productive. The board itself contained two links, one to our artefact's GitHub and one to the open source servers GitHub

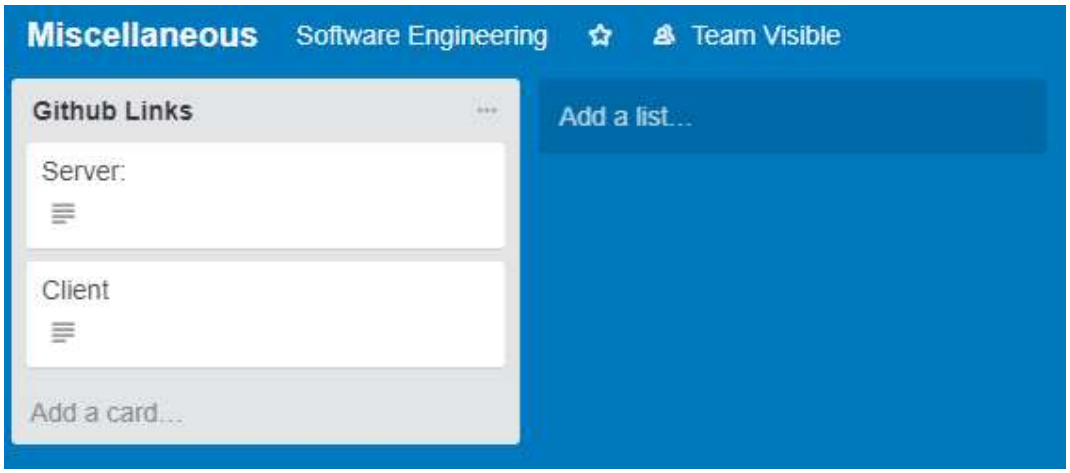


Figure 6: the miscellaneous board

Our fourth board is one of our most important. Our product backlog board allowed us to easily keep track of the product backlog of the artefact. The first part of this board is our “icebox”. In agile development an icebox is a collection of items that whilst you won’t be working on anytime soon you don’t want to get rid of and therefore keep a record of them. For us this gave us not only a way of keeping track of items in the product that were either far off in development or low priority but also a way to view the overarching scope of the artefact and allow us to generate a timeline of how the project was progressing.

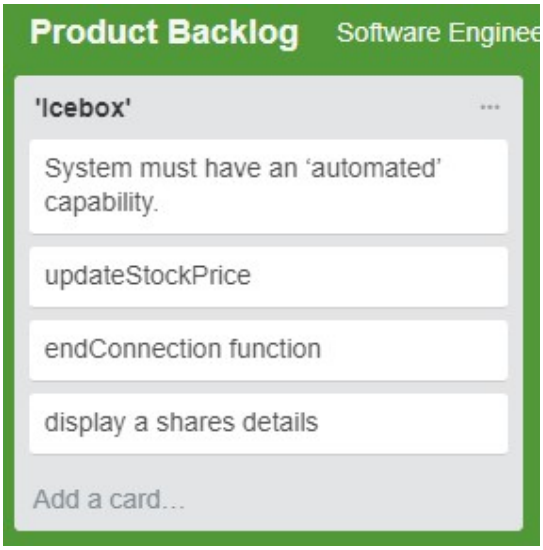


Figure 7: The Icebox list

Our next lists allowed us to track items in the product backlog through their development through sprints. These lists proved incredibly useful in giving the team an idea of what to expect from each sprint and what still needs to be completed in the next sprint. We also attached a priority for each user story using colour coordinated labels with red being the most important and green being least. This priority system further allowed us to prioritise user stories from the backlog to work on during our sprints.

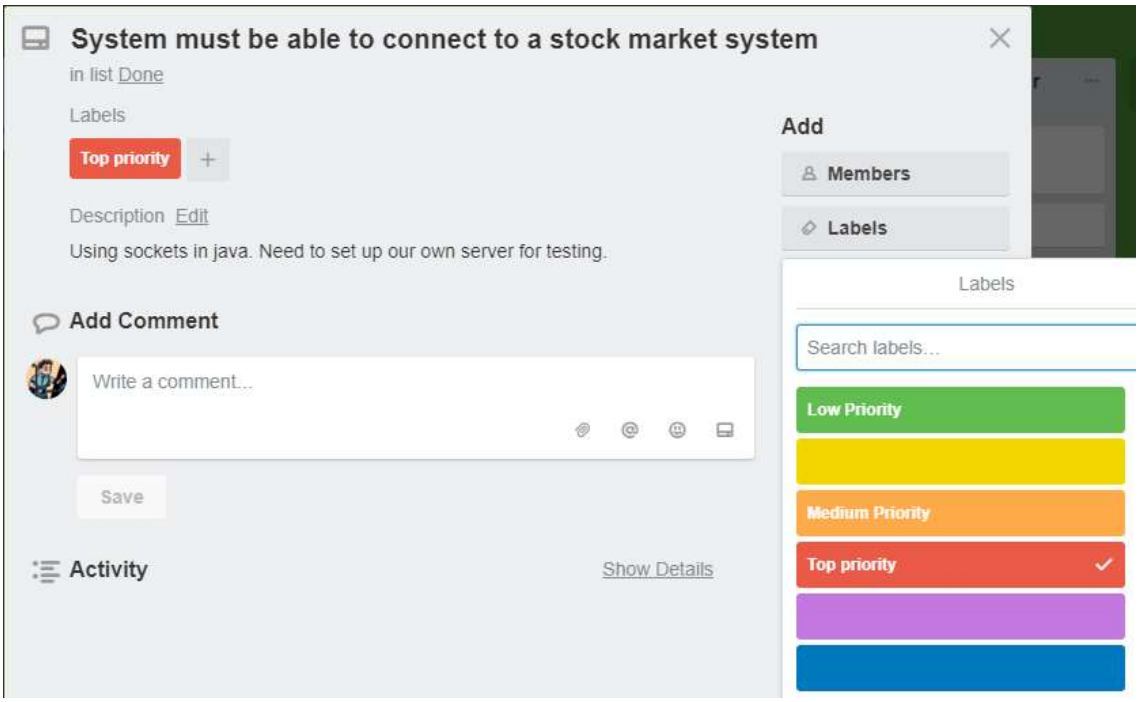


Figure 8: A user story labelled “Top Priority”

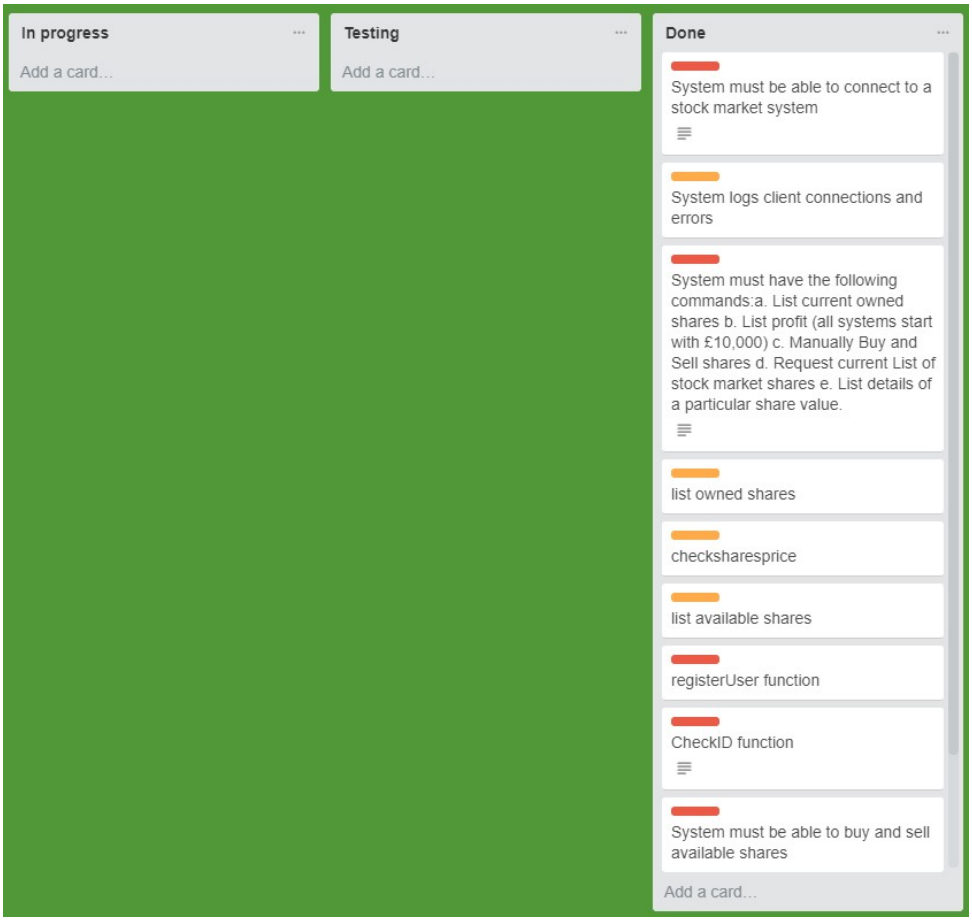


Figure 8: The Product backlog tracking board

A further list was used to keep track of the changes to the open source server platform using GitHub tracking. This proved to be an important resource as the team could compare changes in the completion of user stories with updates of the open source server. Any discrepancies could then be reported and turned into issues.

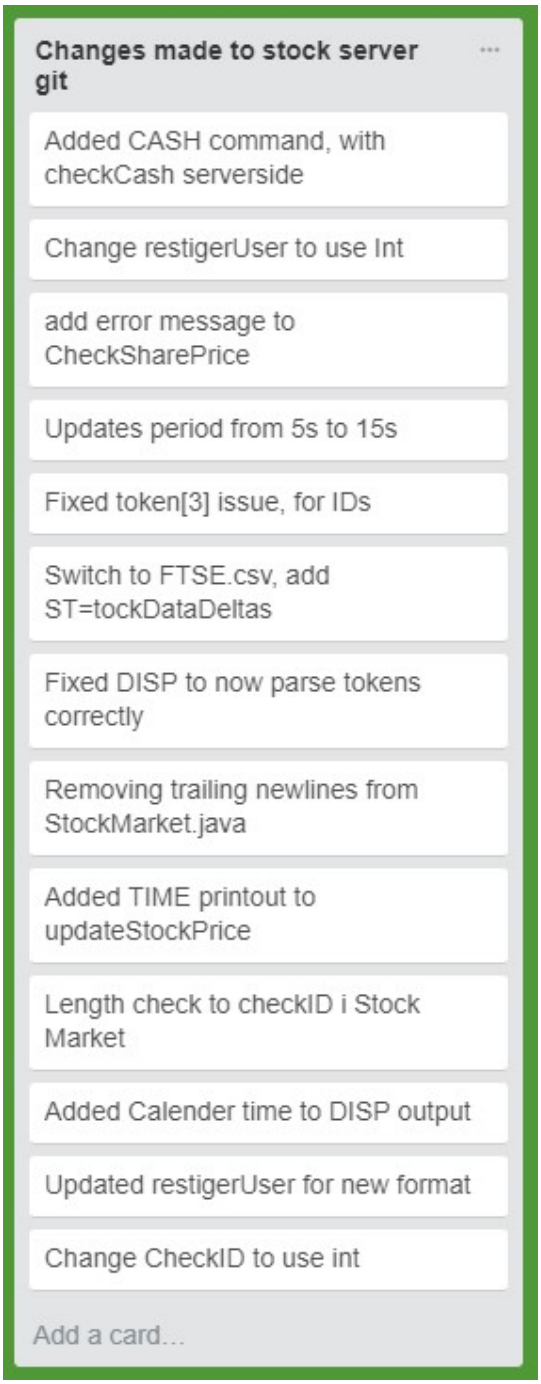


Figure 9: the GitHub server tracking list

The next board initially was created with the Trello when the team didn't have much knowledge if scrum and thereby serves little use/is not used anymore. The task tracking board evolved into

several other boards including miscellaneous and user stories. The board was not deleted as it did not need to be, it was not taking up unnecessary space and wasn't an inconvenience.

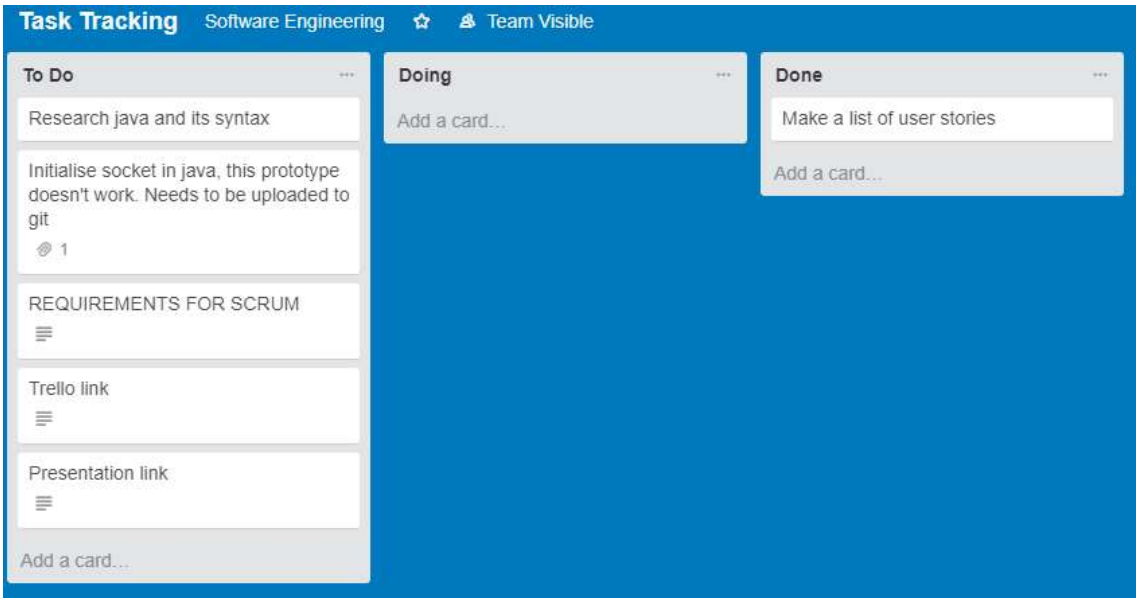


Figure 10: The relatively unused task tracking board

Our final board was tracking user stories and their implementation into the artefact. A user story is a feature to be implemented that the end user would like in the finished system. This board contains lists containing all the user stories of the artefact separated into categorised by their stage in the development cycle. After each sprint this board was updated to reflect the outcome of the sprint with relevant stories being moved around appropriately.

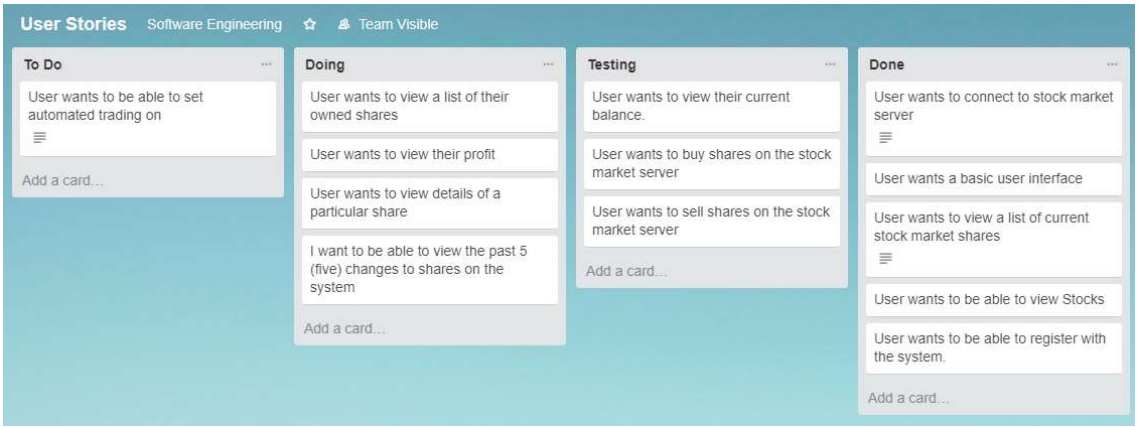


Figure 11: The user stories board

Overall Trello proved an unparalleled tool in our development of the artefact. It allowed us to collate ideas, files and provide feedback to other members of the team working on the project. One of the greatest strengths of Trello is its usability and gentle learning curve. On the outside Trello looks basic and lightweight, however there is plenty of hidden functionality such as using labels for priority and commenting on lists allowing to provide feedback and additional data. In a smaller team such as this Trello is manageable, however I believe Trello would be hard to scale up for a larger development team. One way around this would be to have separate Trello rooms for different scrum teams that

then feedback. Trello does have a basic permissions system that would facilitate “higher ups” managing projects however it is limited in scope and we personally found that members would need admin privileges anyway to do some basic tasks. It has to be noted however that Trello is completely free and fairly unrestrictive which is a definite strong point of the platform.

Trello itself is prone to some weaknesses. It doesn’t have any inbuilt tools to explicitly help with agile development. A tool to create a burndown chart for instance based on deadlines imposed on user stories would be immensely useful to gain a scope of the project. Another issue I encountered was no explicit priority system; instead I had to resort to setting coloured labels on user stories to designate priority. This is an only minor gripe with Trello however. One further issue is the lack of explicitly creating subtasks for user stores without putting them in the comments for each user story. Although this could technically be done with comments, comments are not immediately viewable and so can be overlooked.

Earlier I spoke about how important communication is in project such as this. Trello does not offer any kind of real time messaging which I believe is important in a professional environment. It is not professional to use something like facebook or other social media as it not seen as a professional environment by an employee and so would not be treated like one. Therefore, we had to look for a professional environment to communicate on. At first I suggested IRC as it is very lightweight and but also very versatile and a tried and tested standard. However, I found that I was the only one with experience with IRC and the other team members found it to be quite confusing. I therefore decided in the best interest of the team that we look for a more user friendly communication platform with a gentler learning curve. After researching different team collaboration tools we settled upon the popular service “Slack”.

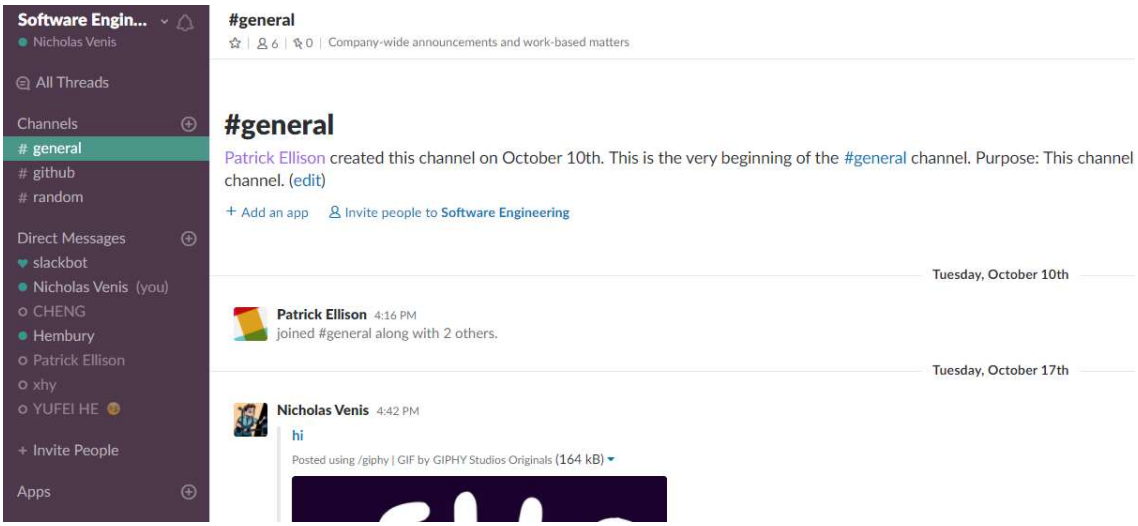


Figure 12: Our Slack home channel

Slack was where 100% of messaging between the team outside sprint’s took place. Immediately after creation I created two channels; random and GitHub. Random was a channel created for the sole purpose of discussing anything non work related. I hoped that by the separation of work talk and casual talk slack would be used more productively and not like a social network. Thankfully this idea proved to be true as all discussion inside the general channel was on topic and related to the work in hand.

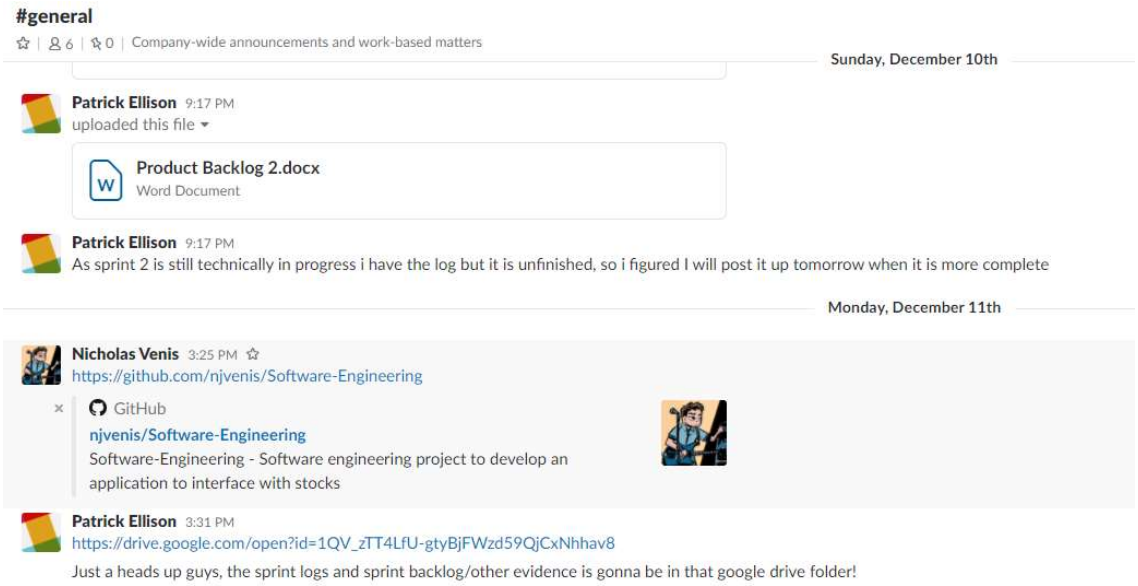


Figure 13: An excerpt from our general channel

The Second channel I created was GitHub. I had envisioned this channel for the sole purpose of members subscribing to this channel to be notified to any commit or change to our artefact. To do this I linked my GitHub and Slack account together and added a prebuilt bot to the channel that tracked the changes of our artefact. This worked perfectly with the bot notifying anyone subscribed to the channel to any changes to the master branch of the artefact. This served not only as a tool for tracking versioning but also if anyone stole any of our keys, especially on lab machines and pushed malicious code we would be notified. This is also of great use/importance for when the artefact would go live and we would receive issue tickets, these would show in the channel and so we would be notified to our mobiles instantly.

(screenshot of GitHub channel overleaf).

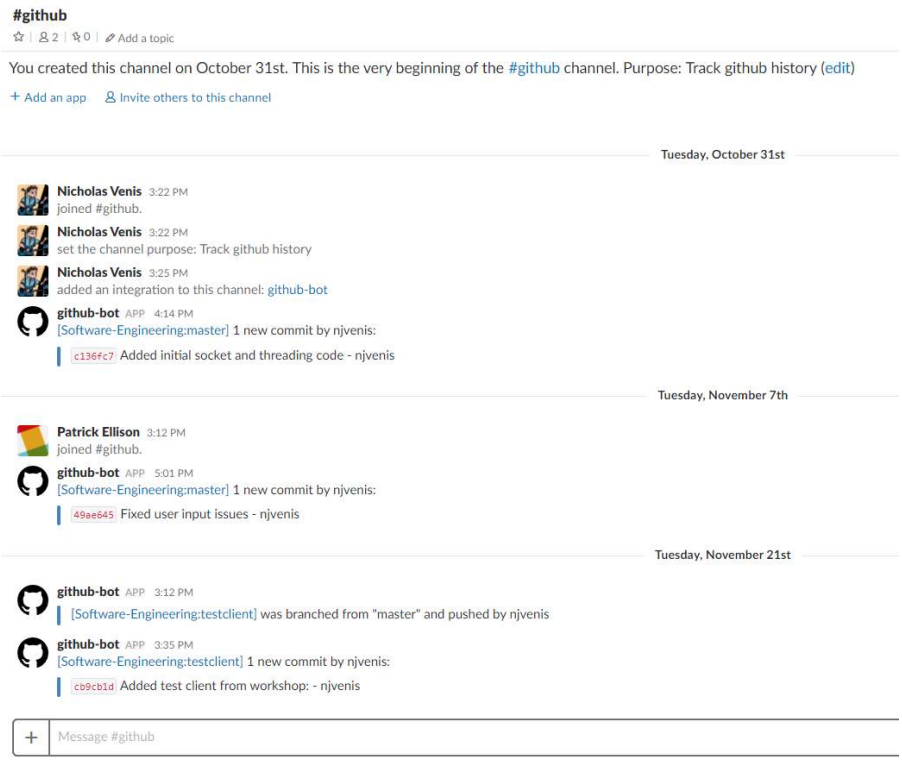


Figure 14: GitHub channel with bot added, publishing updates

Slack also had the added benefit of users being able to set a status for themselves or tell us they wouldn't be there for a sprint. Although members did not always tell the team they wouldn't be there it is a helpful addition telling the team they were out ill or whatnot as roles could be divided appropriately and the team would be less overworked during a sprint. However, in an ideal world this should be done every time a team member was absent, in reality this was done once or twice and some status' such as "out sick" lasted for several weeks even though team members were present. In an actual software development environment this would be managed with HR and the appropriate holiday or sick day and so in a real development environment this would be less of an issue.

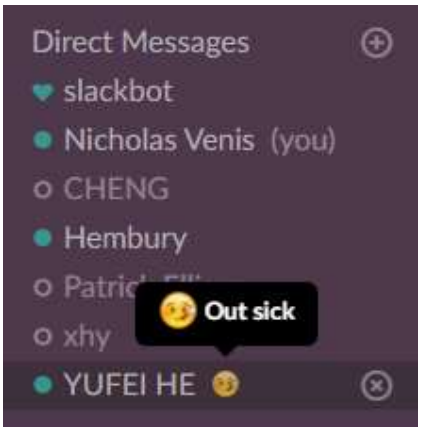


Figure 15: A team member indicates they are sick

It can be said however that our slack was quite bare. There is little messaging in the general channel with communication proving to be a key weakness of our group. Most of the communication I had on slack was between me and another member Patrick. I didn't think the rest of the team took to the platform like us and so were uncomfortable communicating. Communication between me and Patrick occurred on a regular basis and Slack's inbuilt file sharing features were used to send relevant files back and forth.

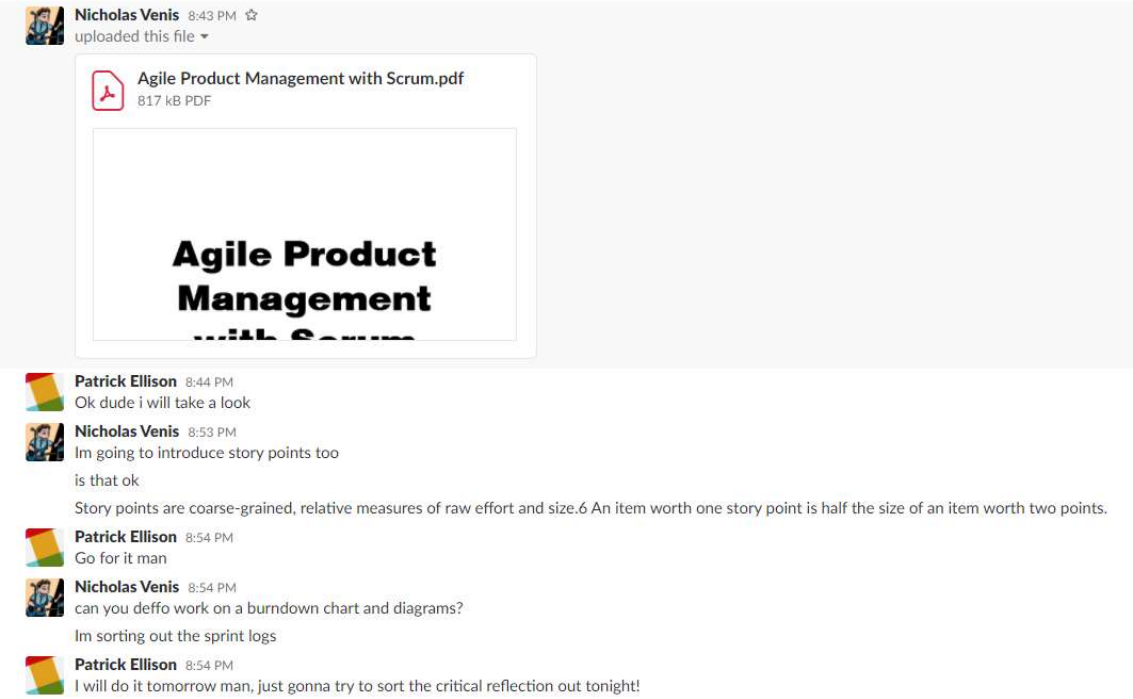


Figure 16: private messaging between me and Patrick

Slack provided a perfect platform for what we needed and combined features from several popular formats of messaging such as IM and Email into one easy to use service. One of the key points about slack I really appreciated was its intuitive UI and inbuilt features. For instance, the GitHub bot showed all changes as soon as they were made and allowed us a backup of versioning other than navigating to the GitHub project page itself. We definitely faced problems with Slack however, channels are very hard to delete and the permissions system is counter-intuitive with little to no documentation. To circumvent permission problems, Me and Patrick had to make ourselves admins of the entire board. This creates issues with upscaling as it only takes one rogue user with admin permissions to cause massive damages to a system; especially deleting logs or using the inbuilt file transfer system to distribute malware. This could be seen as one benefit of IRC over a more robust IM service such as slack. Slack further ties in with the idea of centralising people and data as in terms of adding people and separating teams the platform is massively upscalable. Furthermore, slack has the massive advantage of being a free service and so has a massively low barrier for entry.

Slack provided us with integration with GitHub but also has the possibility for many other services used in industry as well as more social plugins such as Giphy. Slack also gave us the useful option of searching content based on date. On a larger, longer project with everyone communicating data can get buried easily, therefore it's important to have some form of feature for finding older data. This is one of the major drawbacks with IRC. Not only did slack have the option of inbuilt file sharing but

also to add code snippets, this function was used between me and Patrick to share client code on multiple occasions. The most prized possession in my opinion of slack is its portability. Around 70% of the interaction I had on slack I had was on my mobile with others in the group sharing the same view. Being able to communicate with team mates out of office hour straight to their mobile does have its drawbacks however as it raises the point of when does the work end. Slack does try to combat this by automatically silencing notifications after 8pm however you can still opt in to send a notification. This is more of an overarching problem with mobilising work and specifically slack's fault.

The final tool in our arsenal was GitHub. Coming into this project none of the team had any experience with GitHub and so were definitely in unknown territory. I initially created a private repository on my personal GitHub but ran into multiple problems committing to the repo. The solution was eventually found to be with Windows credential manager on a lab machine but also highlighted the plight of using a private repository for the team being able to clone it on their machines. For this reason, I recreated the repository as public and committed the initial readme.

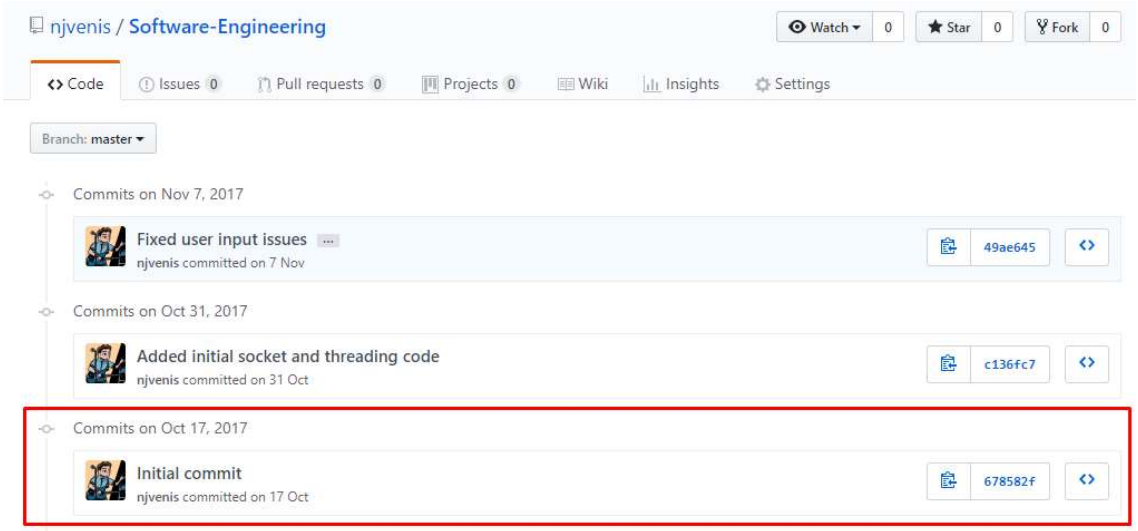


Figure 17: the initial commit of the repository

The local repository existed on my personal flash drive. The flash drive was encrypted with BitLocker, an inbuilt encryption suite that uses a password to access data. This was necessary in the event the USB was stolen or lost as in a software development environment this could result in corporate espionage or other malicious activity.

Following the initial commit Two further commits on the 31st and the 7th added further functionality and user stories to the artefact and were pushed to the master branch. Using Vim, I gave each commit an apt description hopefully detailing the changes made to the master branch, these commit descriptions were also present on the slack GitHub channel.

Figure 18: The two further commits with verbose description

Following this in workshops we were given a test client with more inbuilt functionality their own current artefact. I did not want to push these changes to our master branch and so created a new branch “testclient” to store the new artefact. For one merging the test client and ours would cause a number of code breaking bugs however we wanted version controlling of both. The idea was to get our client to a phase whereby it was suitable to merge with the test client branch to update the master client branch to include full functionality. Unfortunately for us due to lack of manpower and time we were unable to implement this.

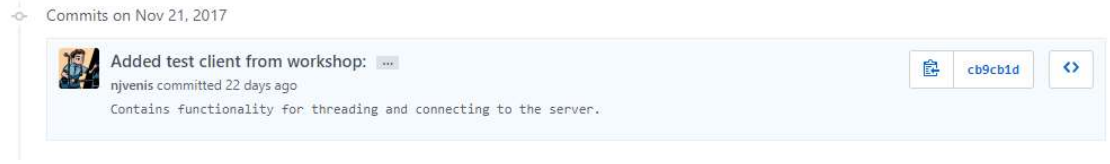


Figure 19: the first commit on the test client branch

Another large part of this project was the open source server to go with the client. This was located on GitHub with a local cloned repository on the aforementioned memory stick. I forked the server to my own repo so that I could change the files locally without affecting the original master many people were using.

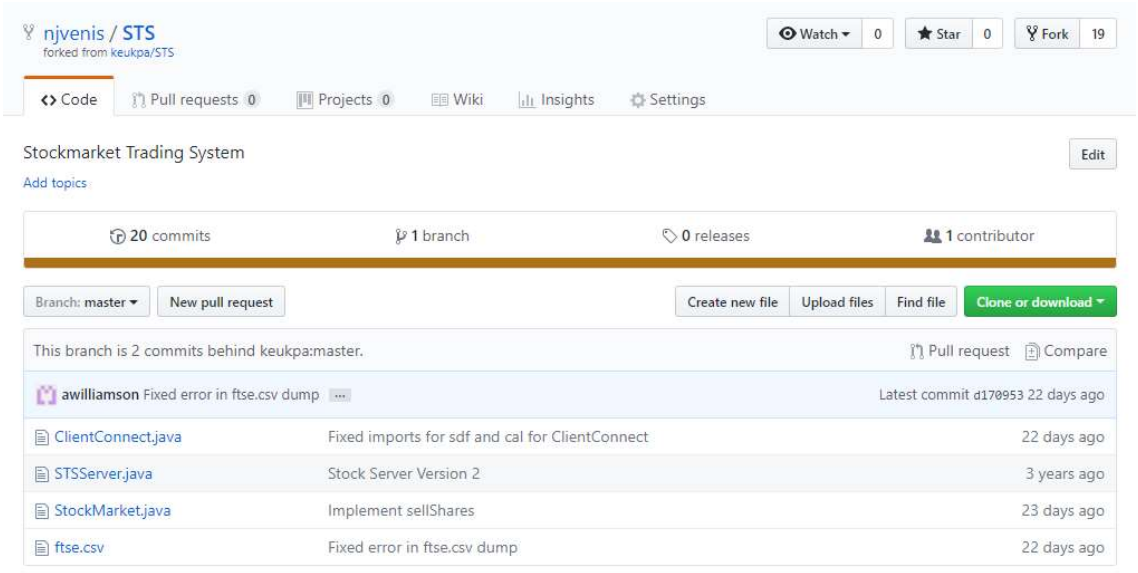


Figure 20: The forked repo in my personal account

As the repo was forked I could look for bugs and issues in the code that were challenging the team without damaging the original master branch. This exact scenario came into effect when an indexing bug was found within a function. After some experimentation I could reproduce the bug fairly reliably and so submitted an issue ticket to the writer of the project.

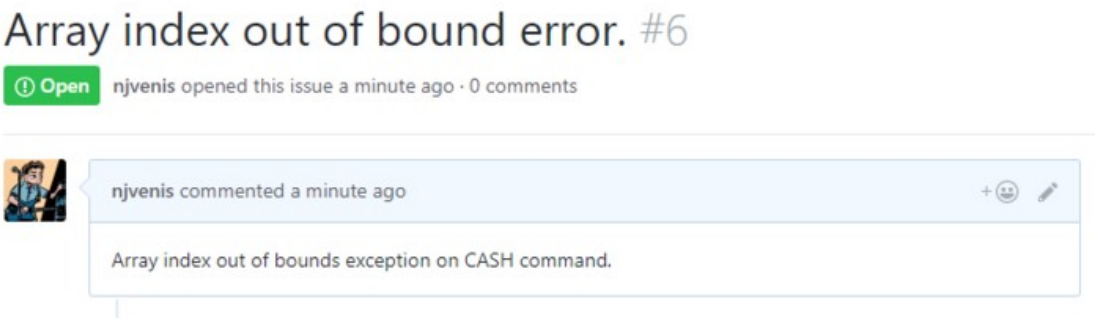


Figure 21: The issue ticket created.

If the team has more time we could have worked on our own fix for this issue and committed it to our fork of the project. As the software is open source the creator may not have that much time to work on errors especially if there are many. If we implemented a fix that worked, we could then create a suitable pull request which the creator of the software could then merge into his own code if he thinks it is up to standard. This is a massive part of open source software development and a massive advantage for many reasons. For instance, as multiple people can fork the repo and add changes the software stays up to date with minimal work for the original creator. This idea of collaboration can also be applied to a team working on software such as in a scrum. The scrum master could veto changes in the master branch from pull requests and thereby multiple team members can work on the same artefact at once.

The software being open source also has the added benefit of being free with an abstract idea of free updates coming from commits and pull requests. There are however some issues with making software open source. Integrity of the artefact could be challenged with many people committing their changes to an artefact. Anyone with knowledge of the source code has the potential to commit changes if approved and so some issues can occur whilst merging changes. As the initial developer is not essential to continually update the project the platform can last longer that some closed source programs, this could be due to running out of funding or other problems with staffing. Furthermore, as the program is not closed source and if popular it could be assumed has a decent amount of committing to it any bugs would theoretically be patched faster as more people are working on said problem at once.

In our project I personally used GitHub to version after every important update. In industry however best practice is to commit after every change no matter how significant. However, in this project I believed I was the only one that engaged fully with GitHub and saw it as a powerful tool for version controlling. I was the only one to commit to our repository and therefore we have no pull requests. I therefore think in the overall scope of this project it has a less than desired effect and the project technically was open source however did not benefit too much from it. On reflection if I had explained more of GitHub and how to use it more to the team it would have possibly resulted on more team collaboration of the artefact with not only me having committed. The links to repository were public on both Trello and Slack therefore availability was not the issue.

References

Cohn, M. (2012). Agile estimating and planning. Upper Saddle River, NJ [u.a.]: Prentice Hall PTR,pp.56-59.

Cohn, M. (2017). The Daily Scrum Meeting. [online] Mountain Goat Software. Available at: https://www.mountaingoatsoftware.com/agile/scrum/meetings/daily-scrum [Accessed 11 Dec. 2017].

Sutherland, J. (2017). AGILE DEVELOPMENT: LESSONS LEARNED FROM THE FIRST SCRUM. [ebook] Cutter Agile Project Management Advisory Service, pp.3-5. Available at: http://www.scrumalliance.org/resources/35 [Accessed 11 Dec. 2017].