Software Engineering (CS3051) Group Project Team A presents

StockStalker

a friendly stock tracking system

Matriculation IDs:

100004351

120023193

100011408

100008800

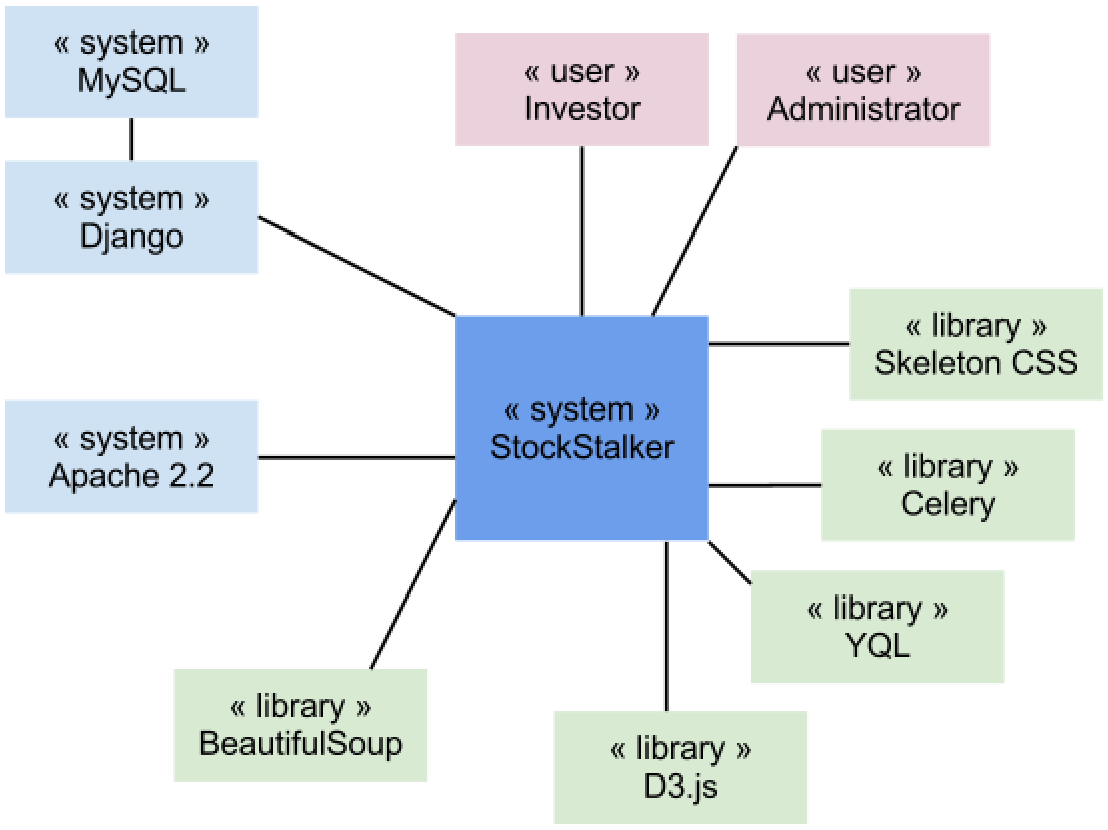
100014525

Introduction

The aim of this practical was to create a web-based stock tracking application for long-term investors, while familiarising ourselves with the SCRUM software development methodology.

1: Architectural Design

Context Model

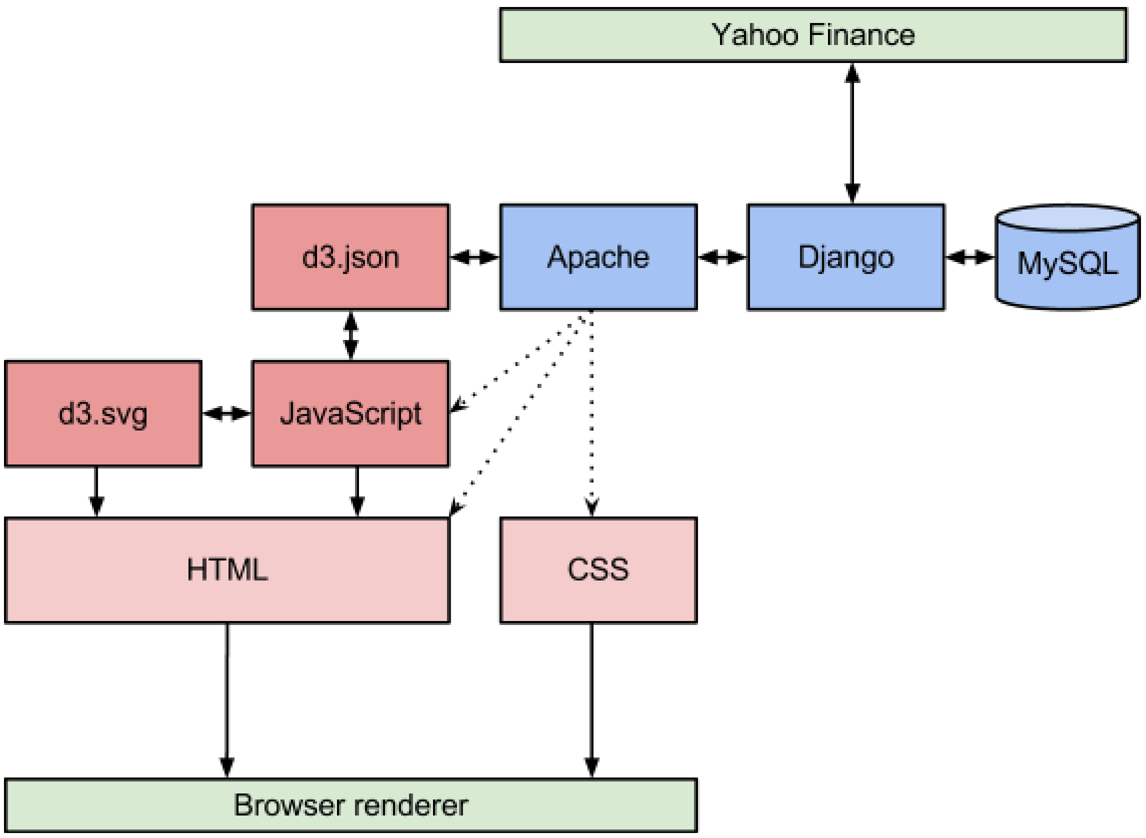


The above diagram is the context model for our stock recommendations system, StockStalker. Apache was used as a web server. Django was used for web development, database abstractions and page templates. We used the following libraries:

Skeleton (CSS) as a mobile-friendly starting point for the site's style sheets
D3 (JavaScript) for client-side code
YQL (Yahoo Query Language) (Python) for querying Yahoo Finance servers
BeautifulSoup (Python) for automatic HTML parsing of a stock ticker list

Celery (Python) for scheduled server tasks (such as cache management).
The users we considered were the stock investor and the system administrator.

Architectural Diagram



Our system has a traditional client-server architecture. All communication is shown with solid arrows. Dotted arrows show the contents of the reply to an initial “bootstrap” HTTP query to the web server from the client, for purposes of rendering the initial page layout.

The server (subsystems shown in blue) communicates with the Yahoo Finance server and with client instances, while keeping a cache in a MySQL database. The client (subsystems shown in red) communicates with the web server and produces an HTML DOM structure (with the help of the D3 library) which can be rendered by the browser to the end user.

All communication with the MySQL database is handled by Django based on our specified “models” in the models.py files. Using a tested framework for this allowed us to focus on other features, while knowing that we were building on an already robust and well-documented framework.

Implementation overview

To give the reader a brief understanding of the structure of the submission folder, we provide a summary of some of the more important sources. Most of our functionality is implemented in the view modules:

/demo/stockstalker/views.py - Defines content of user interface, and handles updating users watchlist.

/demo/stockdata/views.py - Requests data from financial data sources and generates JSON output in response to asynchronous requests from the client. As a part of this it also handles caching.

/demo/stockstalker/stock_evaluator.py - calculates scores to all stocks in the system based on stock recommendations algorithm.

/demo/stockdata/index_manager.py - Scrapes the list of stocks in a supplied index. Updates the database accordingly.

/demo/templates/ - Layout of the interface.

/static/javascript/ - Graph generation and asynchronous request handling. Automatic updating of financial data on page. Sending requests for adding/removing stocks from watchlist and handling the result.

/demo/stockstalker/tasks.py - periodically updates stock recommendations.

/demo/stockdata/tasks.py - periodically clears the cache and scrapes the list of stocks.

2: Team Structure & Role Assignment

SCRUM Roles

Product Owner

Fred volunteered to be the product owner, because he is a stock investor himself and frequently uses web-based stock screening applications such as Yahoo Finance, Globe Investor or Bloomberg to find information. Hence he felt that he had a good idea of what the finished product should look like.

SCRUM Master

Ivars volunteered to be SCRUM master, as he was keen to take a leadership role in the project and wanted to make sure the group closely followed SCRUM procedures. He also felt comfortable leading daily stand-ups, sprint planning, sprint reviews and sprint retrospectives.

Development Roles

Team roles reflect the fact that we wanted to make good use of our members' individual skills and backgrounds.

Antti had experience with web development, client-side programming and data visualisation and wanted to learn the D3 library, so he was responsible for writing the main client-side code, including the historical stock price chart and the search box back-end code. He worked closely with the product owner to demonstrate designs and ideas and with the server developers to discuss necessary APIs.

Conrad had experience with server management, therefore performed initial server configuration. He pulled together parts written by other people (such as the YQL interface, the client-facing data API and the profile template code) into a cohesive server. He worked Fred's initial user interface designs into a Django template. Finally, he supported Antti in client-side scripting and design roles and worked with Fred to try to ensure the correctness of displayed data.

Fred created and designed the initial StockStalker user interface and wrote the structural template code for all pages. Having a knowledge of investing strategy, he developed the algorithm StockStalker uses to make its recommendations. He also worked closely with all group members as product owner to make sure their work reflected his vision for the system.

Ivars wrote lots of back-end server code. This includes the Yahoo Finance interface for downloading current and historical data, code for web-scraping the names of companies from a given index, the recommendations back-end and user profiles. He ran some of these tasks (index scraping, cache clearing, recommendation updates) periodically through Celery.

Luke had a collaborative role, working with others when the need arose, mainly on the server. For example, he cooperated with Fred on Django templates for user and stock profiles, and worked on caching with Conrad. He also set up the initial round of PyUnit tests.

3: Implemented and Omitted Features

Included Features

Stock Price and Volume Charts

Our system can plot the closing price and trading volume on the same axes for each stock that our system is tracking, for time intervals that can be chosen by the user. The time increments we chose to display are: 1 week, 1 month, 3 months, 6 months, 1 year, 5 years and 10 years. The default time interval is 1 year. The chart's display interval can be changed without refreshing the page. We focused on longer time intervals, as they are most relevant for long-term investors.

Lots of attention was paid to user experience. The axes are scaled to emphasise changes in the selected interval. A subtle animation on page load indicates to the user that the chart is interactive. Moving the cursor over any point on the chart creates rule lines and a rule label which show the exact date, closing price and trading volume on the selected data point. The axis labels automatically fade out when they would obscure the rule label and an aura around all labels ensures legibility even when overlapping with chart data. To help users understand the chart faster, the y-axes are different heights and their text colours correspond to the matching data. The colours were chosen to be easy on the eye.

The chart was also designed to be suitable for colour-blind users: All information can be deduced even just by perception of binary luminosity; hue is ever only supplementary.

Using vector graphics has great advantages for web development: The SVG format is a widely-adopted W3C standard. It embeds into the HTML DOM tree, which makes it manipulable with JavaScript, styleable with CSS and easy to parse by browsers. It is also human-readable and compact in comparison to image formats storing pixel data. Vector graphics also render cleanly at any zoom level, making them suitable also for mobile devices and for users with impaired eyesight.

User Profiles

StockStalker user profiles are created by the system administrator from an administration web interface. The user profile page enables registered users to follow or unfollow a stock, change their details (name, last name, e-mail and password), to search for particular eligible stocks and to see StockStalker's trade recommendations. When the user has searched for and selected a stock, they are redirected to that stock's page, where they can view the price chart for that stock and see the associated financial information (such as its current price or price-earnings ratio).

Stock Recommendations

Our system provides stock recommendations by a value-based investing approach. This approach is conducive to long-term investment success, as requested by the product owner. Our algorithm uses financial metrics to assign a score to each stock. These metrics include current price, 50-day moving average, 200-day moving average, price-earnings ratio, price-book ratio, price-sales ratio, price-earnings to growth ratio, price to expected earnings for the next fiscal year, and the dividend yield.

We will explain the significance of these metrics for long-term risk-averse investors before outlining our scoring algorithm.

Moving Averages

The relationship between the current price and the 50- and 200-day moving averages provides an indication of the stock's price momentum. A stock's current price being above its 50- or 200-day moving average is a signal that the price has upward "momentum" of increase. This is a buy indicator. In contrast, stocks trading below their 50- or 200-day moving average may signal downward momentum, which is a sell indicator. These metrics are less significant to long-term investors as the following ones, but are so common that they are worth including.

Value Investing Metrics

The *price-earnings*, *price-sales*, *price-book*, *price-earnings to growth* ratios and the *price-earnings to expected earnings* for the next fiscal year are very significant value metrics. All of them indicate that a stock could be cheap. For instance, a stock with a low price-earnings (*pe*) ratio, all other things being equal, means that an investor is paying less per pound of earnings. Stocks with low *pe* ratios have generally outperformed those with high pe ratios for that very reason — they are typically safer.

Dividend Yield

Finally, dividend yield is significant for long-term investors because studies have shown that a large component of long-term portfolio appreciation is due to reinvesting dividend earnings. The higher the dividend yield, the more money a company is paying an investor to be a shareholder, which bolsters their returns. There is a catch here. Stocks with dividend yields above 6% are generally considered risky because the market is often pricing in the fact that this company will cut their dividend in the future.

Our Algorithm

The algorithm assigns scores to stocks based on the above financial metrics and combines them using a significance multiplier. Higher scores are better; these stocks are more highly recommended.

Stocks with a price of 0 (indicating bankruptcy) are immediately assigned a large negative score. Next, points are assigned to a stock based on its 50- and 200-day moving averages as outlined above. This metric (owing to being less important for long-term investment) has a lower multiplier than the other metrics. For the price-earnings ratio, we assign more score to a stock if its price-earnings ratio is below 15 (a bit below the historic market *pe* ratio; indicating it is cheap) and and deduct score if the *pe* ratio is above 25 (indicating it is expensive and more susceptible to a price drop). Similar threshold numbers are used also for the other metrics. For example, a price-book ratio under 1.5 is a buy signal. Finally, dividend yield contributes significantly to the score assigned to a particular stock because of the role of dividends in generating future investment returns. As noted above, stocks with dividend yields above 6 won't get any points because they are often risky and likely to cut their dividend in the future.

It is important to note that for some stocks Yahoo Finance returns incorrect data for the financial metrics listed above. In particular, we found that it sometimes reported price-earnings ratios in the thousands, which is extremely rare, for almost every company in the FTSE 100/250. This is without question incorrect and was confirmed to be so by comparison with data provided by Bloomberg. This limitation unfairly biases our algorithm away from many companies listed on the FTSE index. See below (under *Omitted Features*) for our attempted solutions.

Stock Financial Information Summary

If a user searches for a stock and selects it, they will be linked to a page where they can see a summary of all important financial details for that stock. These include price, trading volume, the price-earnings ratio, and many others displayed in a table beside the chart.

Stock Searching Capability

Our system includes functionality to search for stocks on the indices we are tracking, using a dropdown box with an auto-completion feature. The search matches both ticker symbols and company names. Multiple search terms, separated by spaces, are handled as separate terms.

Automatic Updating of Stocks

As the current price and volume changes, all relevant financial information on StockStalker is updated periodically with an asynchronous query to the server.

Automatic Updating of Recommendations and Stocks in Indices

Since the financial information of the stocks changes, the recommendations are updated every hour automatically by the system. Same applies to the stocks in indices, since companies can be added or removed from an index our system, updates the stocks in indices automatically every day, also the administrator of the system can click on a link to execute the indices update.

Tracking Multiple Indices

StockStalker has the capability to track stocks from the FTSE 100/250, CAC 40, DAX 30, DOW 30, and NASDAQ 100. The system is easily extensible, so more indices can be added from the administration interface. The product owner was satisfied with this selection.

Caching

Caching responses to historical stock data queries reduces latency by occasionally making it unnecessary to query Yahoo Finance's servers. Hence, we considered it a high priority to cache when we could.

We found problems with our initial caching model, in which data was sent to the database for caching in a separate task. Because the message queue managing the tasks is delayed, a race condition was established wherein the data would be cached twice if requested in quick succession. Although this would rarely be a problem with a small number of users, increasing use would increase the risk of failure - a thoroughly unsatisfactory situation.

The caching was revised to a simpler model which cached the data atomically during each client request.

The reduction in latency from caching was measured at about half a second for most data sets - for the amount of data that needs to be requested, the time it takes to establish a connection with Yahoo Finance is much greater than the marginal time of a much longer query. Caching is therefore a safeguard against temporary network failure and to reduce the number of requests to Yahoo Finance.

To ensure that the cache does not grow too big and slow, it is automatically cleared once a day to remove cached data that has not been accessed for more than a week.

Omitted features

Reliable Supplementary Financial Information for FTSE Index

As stated in the recommendation section, Yahoo Finance occasionally returns incorrect financial information for companies from the FTSE 100/250. We explored the possibility of switching to Bloomberg, Reuters or Google Finance. We found that access to Bloomberg's data, while free through their Web interface, requires a prohibitive fees to access through their "open" API. Google Finance's API had closed completely, and more importantly there were higher priority tasks than writing another back-end.

Ability to Compare Two or More Stocks

It would be useful for investors to be able to plot the price charts and financial information tables of two or more stocks together on the same page, to better compare them. The usefulness-to-complexity ratio was again too low given time limitations.

Index Overview

We noticed that most websites providing financial information have charts to plot index averages against time. This additional chart could have been very useful for a long-term investor, but time stops for no one.

4: Task Deduction & Assignment

As product owner, Fred created the initial user stories for the project. After seeing the details of the system from an administration perspective, stories were added also by others with Fred's consent in subsequent sprints.

As a group, before each sprint, we selected the highest priority features from the remaining user stories and then added them to the current sprint backlog. We would then decide on the highest priority tasks within these stories by group consensus.

We would then break the user stories down into manageable tasks. This was also done as a group, with consensus. Each group member was responsible for claiming tasks that they felt comfortable working on. Likely due to the group members' diverse skills and interests, tasks were always willingly claimed.

5: Effort Estimation

Before our first sprint, we played planning poker to estimate the effort required for each user story. We often had large differences between our estimations. Hence, after each round, we would each explain the (sometimes surprising) reasoning behind our estimates and play poker again for the same story, if necessary, until we arrived at a consensual estimate. Particularly in the first sprints, this was often higher than most members' initial estimates. Being forced to explain our reasons led to more realistic estimates.

For the first sprint planning meeting, we also played planning poker to estimate the time for each task in the first sprint backlog. Our guesses were much closer to each other, as the tasks we had developed were very specific. Again, when discrepancies in estimates existed among group members, we discussed the reasons our estimates together.

For the second sprint planning meeting, we opted not to play the game of planning poker for estimating story and task times. Instead, we decided to try an experiment. Because the discussions (rather than the poker) had been the most useful part of the previous estimation round, we decided simply to informally discuss how long we thought tasks would take. By this point, each group member had become better acquainted with the parts of the project they had been working on, letting them make more accurate predictions and act as a sort of authority on that part of the project. Through their input the we were better able to understand each component in the project and arrive at a better estimate for overall effort required.

For a larger or beginning SCRUM team, the additional structure provided by playing actual planning poker may be helpful. However, we found the less structured (without poker) discussion of the tasks just as helpful but more effective and decided to substitute poker with simple discussion for the 3rd and 4th sprints.

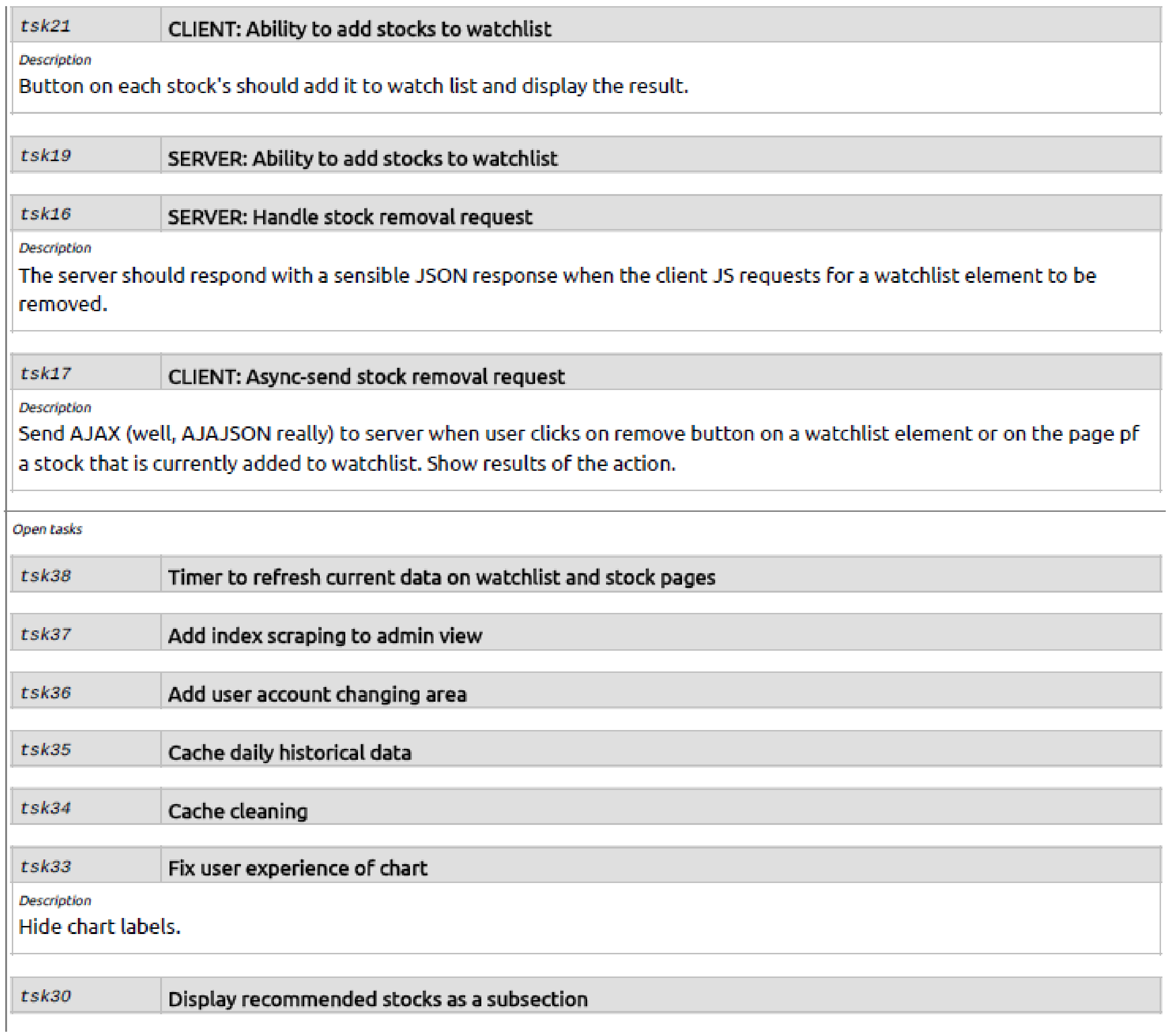
6: SCRUM Tools

Kunagi

Having evaluated similar tools such as *Agilo for Scrum*, we decided to use *Kunagi* as a substitute for the traditional *Post-It*s and whiteboard. This was done to ensure our work would survive even in our absence (strange things happen to lab whiteboards at night) and to ensure we could relocate easily if necessary. *Kunagi* proved to be a useful automation tool, in that it automatically generated a sensible "whiteboard" and burn-down chart.

Kunagi is not perfect though: even using a modern browser, we frequently faced rendering issues. These could be fixed with a refresh, but were sometimes an annoying interruption to sprint planning meetings.

We couldn't entirely replace the physical whiteboard: it was a great "to do list" or "brain dump" for small, current tasks. This made for a good balance, allowing us to see the big picture on Kunagi and the currently relevant details on a whiteboard in the lab. Some photos and screen captures from the project are below.

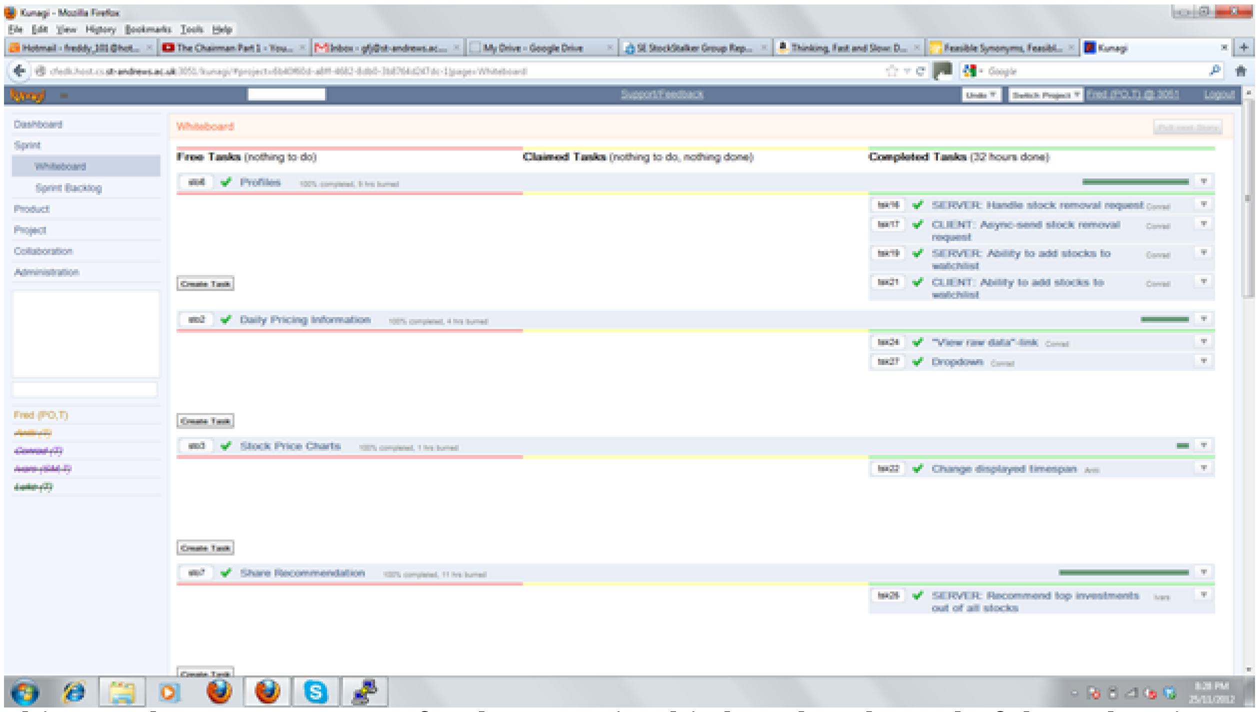


tsk21	CLIENT: Ability to add stocks to watchlist
	Description Button on each stock's should add it to watch list and display the result.
tsk19	SERVER: Ability to add stocks to watchlist
tsk16	SERVER: Handle stock removal request
	Description The server should respond with a sensible JSON response when the client JS requests for a watchlist element to be removed.
tsk17	CLIENT: Async-send stock removal request
	Description Send AJAX (well, AJAJSON really) to server when user clicks on remove button on a watchlist element or on the page pf a stock that is currently added to watchlist. Show results of the action.

Open tasks

tsk38	Timer to refresh current data on watchlist and stock pages
tsk37	Add index scraping to admin view
tsk36	Add user account changing area
tsk35	Cache daily historical data
tsk34	Cache cleaning
tsk33	Fix user experience of chart
	Description Hide chart labels.
tsk30	Display recommended stocks as a subsection

The above figure shows some of the tasks from sprint 3, downloaded from Kunagi.


Whiteboard

Free Tasks (nothing to do) | **Claimed Tasks** (nothing to do, nothing done) | **Completed Tasks** (32 hours done)

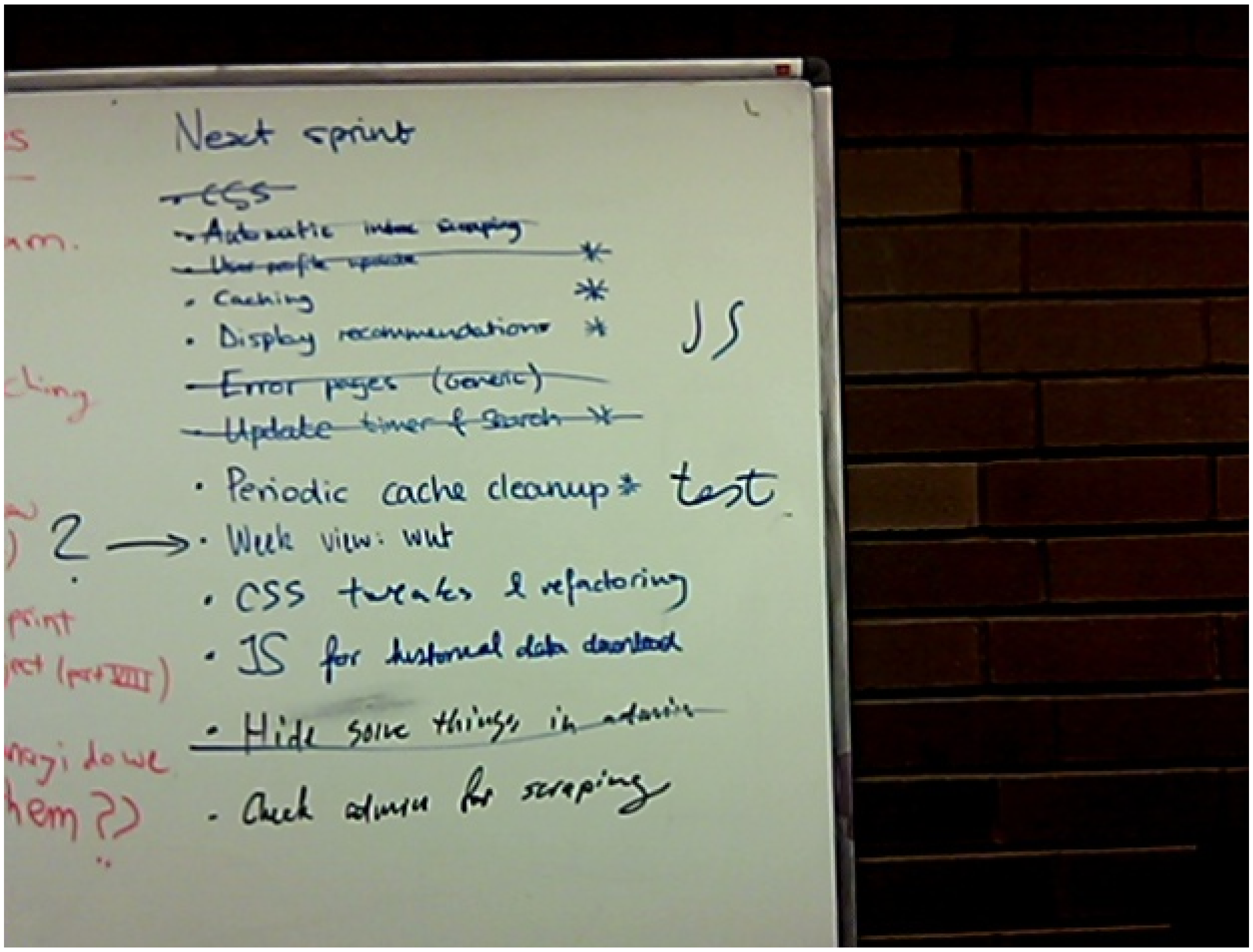
sto6 ✓ Profiles 100% completed, 9 hrs burned
- tsk16 ✓ SERVER: Handle stock removal request Conrad
- tsk17 ✓ CLIENT: Async-send stock removal request Conrad
- tsk19 ✓ SERVER: Ability to add stocks to watchlist Conrad
- tsk21 ✓ CLIENT: Ability to add stocks to watchlist Conrad

sto2 ✓ Daily Pricing Information 100% completed, 4 hrs burned
- tsk24 ✓ "View raw data"-link Conrad
- tsk27 ✓ Dropdown Conrad

sto3 ✓ Stock Price Charts 100% completed, 1 hrs burned
- tsk22 ✓ Change displayed timespan Armi

sto7 ✓ Share Recommendation 100% completed, 11 hrs burned
- tsk28 ✓ SERVER: Recommend top investments out of all stocks Ivars

This was the screen capture for the Kunagi whiteboard at the end of the 3rd sprint. It demonstrates all of the completed tasks at the end of the sprint.


Next sprint
~~CSS~~
~~Automatic scraping~~
~~User profile update~~ *
• Caching *
• Display recommendations *
JS
~~Error pages (Generic)~~
~~Update timer & Search~~ *
• Periodic cache cleanup * test
2 → • Week view: wut
• CSS tweaks & refactoring
• JS for historical data download
~~• Hide some things in admin~~
• Check admin for scraping

Our whiteboard in the lab for small tasks during sprint #4.

Mercurial

School Mercurial servers were used for version control. In retrospect, it may have been better to use a system such as GitHub with other product management features such as issue tracking and code reviews. All of this had to be done verbally, which was fine when the person you wanted to talk to was in the lab sitting next to you, but frustrating otherwise.

7: Meeting Outcomes

Overview of Daily Stand-up Meetings

We met almost daily for stand-up meetings; about five times a week on average. Each meeting was similar in structure: Everyone would stand and each member would explain what they had worked on the day(s) before, what they would work on next and what problems they were encountering. Team members would offer advice on problems others were facing if they had any. Weekend meetings were relaxed; we would sit.

Sprint 1 (November 5 - November 12)

Sprint Planning

The goal of this sprint was to provide a **proof of concept**, including:

- A functional Django server (with user authentication) and development environment
- Ability to pull data from Yahoo Finance
- Displaying a stock chart with placeholder data
- User interface conceptual design

Sprint Review

During the first sprint review, Fred as product owner was shown several completed tasks. First, Antti showed him the stock price charts he had created using placeholder data. Fred suggested adding the trading volume to the graph, and to display further details of a data point when on pointer hover. Conrad showed that the server, along with the Django environment, was up and running. Finally, Ivars and Luke showed that they were able to pull data from Yahoo Finance using server-side code. Fred had created a conceptual design for the user interface incorporating the features that he wanted.

Sprint Retrospective

We discussed a number of points on which we thought went well and which could be improved. Conrad noted that his accidentally catastrophic misconfiguration of the server meant he'd had to consult the school's IT engineers. This had delayed him significantly. This also explains the spike in the burndown chart for this sprint, as he was forced to re-evaluate his effort estimate. We all had some trouble getting used to Kunagi at first, but became more comfortable with it by the sprint's end. Everyone was content that the sprint had finished on time with all of the tasks completed.

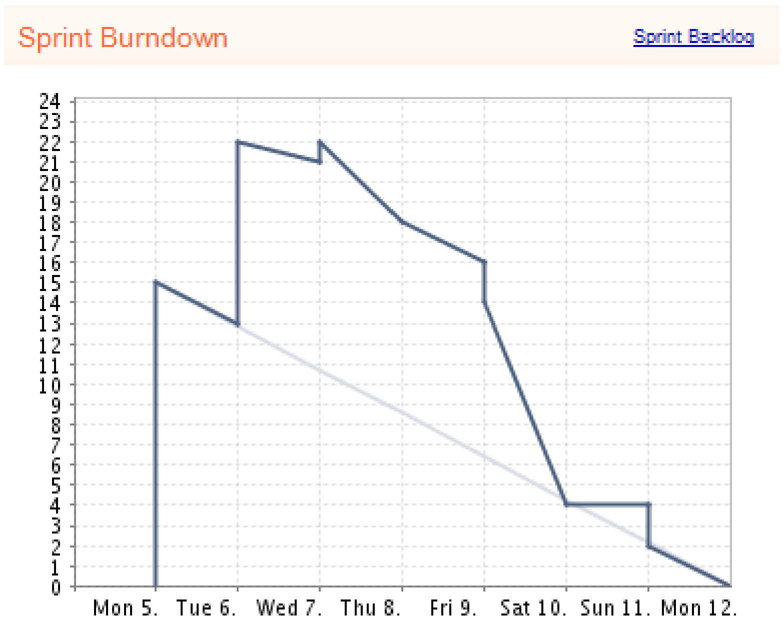
Continue

- **Emphasising task-oriented mindset**. The SCRUM process' explicit concern for tasks to be completed in the immediate future helped us to structure the project and keep on track during this sprint.
- **Using the SCRUM daily meetings to share ideas and knowledge**. Every team member potentially has a greatly time-saving bit of knowledge to contribute. For example, Ivars discovered the BeautifulSoup HTML parsing library from another team member during a meeting.

Change

- **Improve preparation for the daily stand-ups**. A lack of preparedness for some of the meetings caused them to last longer than necessary. Everyone agreed to make an effort to be more prepared for future stand-ups.
- **Work on the project more continuously**. In the first few days of the sprint less work was done than at the end of it, making it harder to use burndown charts to keep on track. This is apparent in the first-burn down chart (see below). However, we did agree that due to the constraints imposed by our schedules, this would be difficult tough to improve upon.
- **Have daily stand-ups at the same time of the day** as required by the SCRUM procedure. We agreed to plan a time of the day when we could meet for most of the days in the sprint. But due to limitations of being in university and vastly different schedules we acknowledged that this would not always be feasible.

Our Kunagi-generated burndown chart looked like this by the end:



Sprint 2 (November 12 - November 19)

Sprint Planning

The goal of this sprint was to produce a **functional demo of the system**, including:

- Displaying stocks in an investor's profile
- Presentable stock information
- Charts showing price and volume using real data

Sprint Review

In the second sprint review, the website was displayed to demonstrate the completed tasks in Sprint 2. Individual stock pages could now be displayed, with functional interactive charts. Some of the links between the pages were also working. For example, by clicking a link the investor was now able to navigate from their watchlist to the page of an individual stock. As product owner, Fred was happy with all of the features in their current form and accepted the tasks in the second sprint as being completed.

Sprint Retrospective

In the second sprint we had worked more on the architectural design and on pulling different parts of the project together. As a result, we got used to the SCRUM workflow. The daily meetings and working together became even more beneficial. We encountered some unexpected issues with the work done for Sprint 1 — these required extra time that we had not captured in the tasks! For instance, we found that the financial data provider we were using (http://opendatatables.org) was restricting access to the data. We therefore had to re-implement the existing functionality using Yahoo Finance as an alternative source.

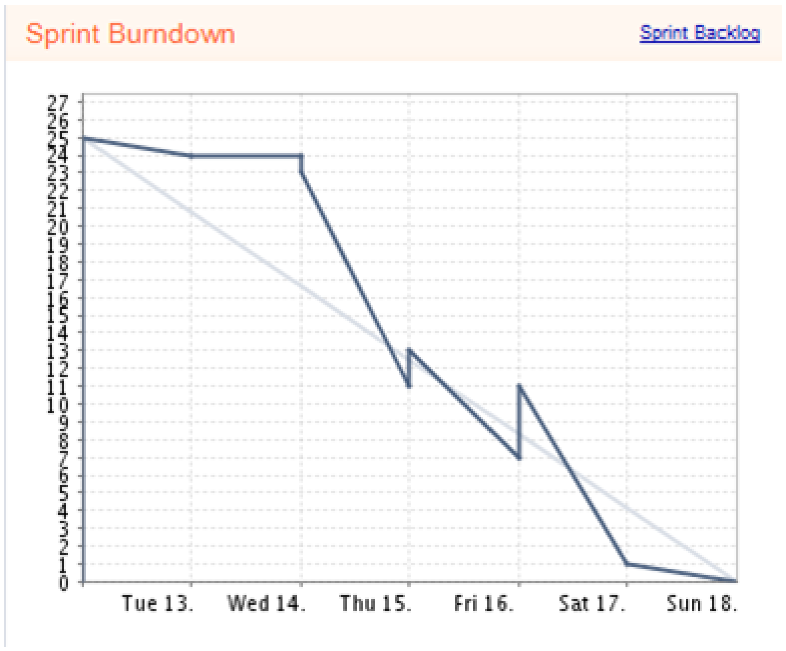
Continue

- **Working together in the same room**. Having a little bit of a meeting going on all the time helped solve problems quickly as they arose and allowed us to make make design decisions on the fly.
- **Using Kunagi**. Everyone was completely comfortable with it by this time.

Change

- **Include refactoring and integration times in task estimates**. In the first sprint, the impact of this was not very significant, as we had little code. However, the code-base grew significantly during the second sprint, and we hadn’t reserved enough time for properly refactoring code and integrating components.
- **Define tasks precisely**. Some of our tasks were vaguely defined. Thinking through these as a group (by having more thorough sprint planning sessions) would give us more confidence in that we are doing the right thing.

Our second sprint’s burn-down was much more on track. It looked like this:



Sprint 3 (November 19 - November 24)

Sprint Planning

The goal of this sprint was to **finish the core functionality**. We worked on:

- Stock recommendations
- Managing an investor's stock watchlist
- Asynchronous client requests to update stock data
- Automatically updating the list of stocks the system is aware of

Sprint Review

By the end of the third sprint, several things had been demonstrated to the product owner. The capability to add and remove stocks from a user's watchlist was working; the stock searching feature was complete and implemented on all pages. The recommendation algorithm was successfully providing recommendations from stocks on investors' watchlists. Fred noticed a bug where the 200-day moving average and 50-day moving averages were showing the same result. In addition, he suggested ideas for further CSS styling to improve layout to better reflect what he would expect as a user. Overall, Fred was very encouraged, particularly after seeing the data updating correctly in real time.

Sprint Retrospective

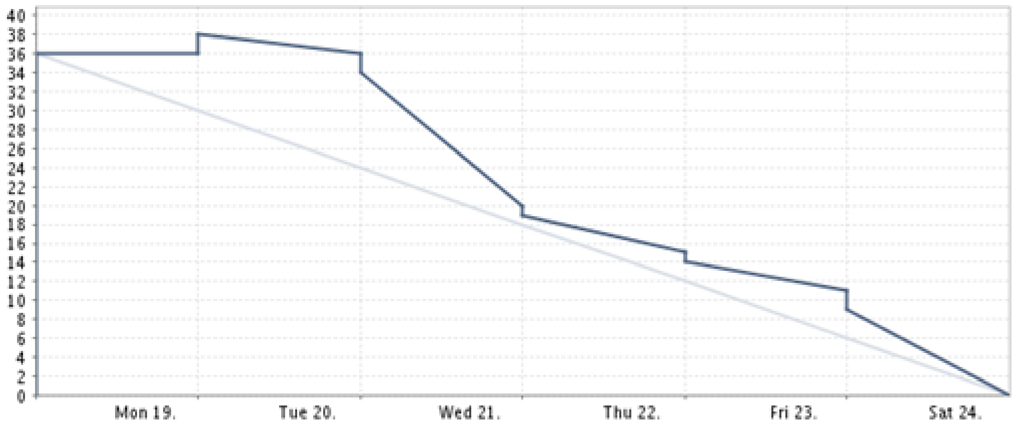
It is important to note that the Burn trip was between the 19th and the 21st of November, making the burndown chart predictions less useful for this sprint.

We found problems with Yahoo Finance as a historical data source. The same query sometimes returned a different number of data points! In other cases, the data itself changed - for instance producing spikes in trading volume 6-8 orders of magnitude greater than the rest of the volume data. This is apparent also when browsing data using the Yahoo Finance interface. Attempts will be made to minimise the impact by re-querying data that is detected to be ridiculous.

Continue

- **Defining tasks precisely**. Compared to the second sprint, the captured tasks were more thoroughly considered. We had more experience and a better understanding of our tools.
- **Accurate estimates**. Estimates were more accurate during this sprint, since we had allocated more time for testing.
- **Meeting at the same time every day**. It was useful to know in advance when and where meetings happen, as we wasted less time on finding each other.

Our burndown chart looked like this at the end of our third sprint:



Sprint 4 (November 25 - November 26)

Sprint Planning

The goal of this last short sprint was to **implement caching**, **tweak the user interface** and **write the final report**. We worked on:

- This report
- Finalising templates & CSS
- Historical stock data cache
- Improving stock recommendations

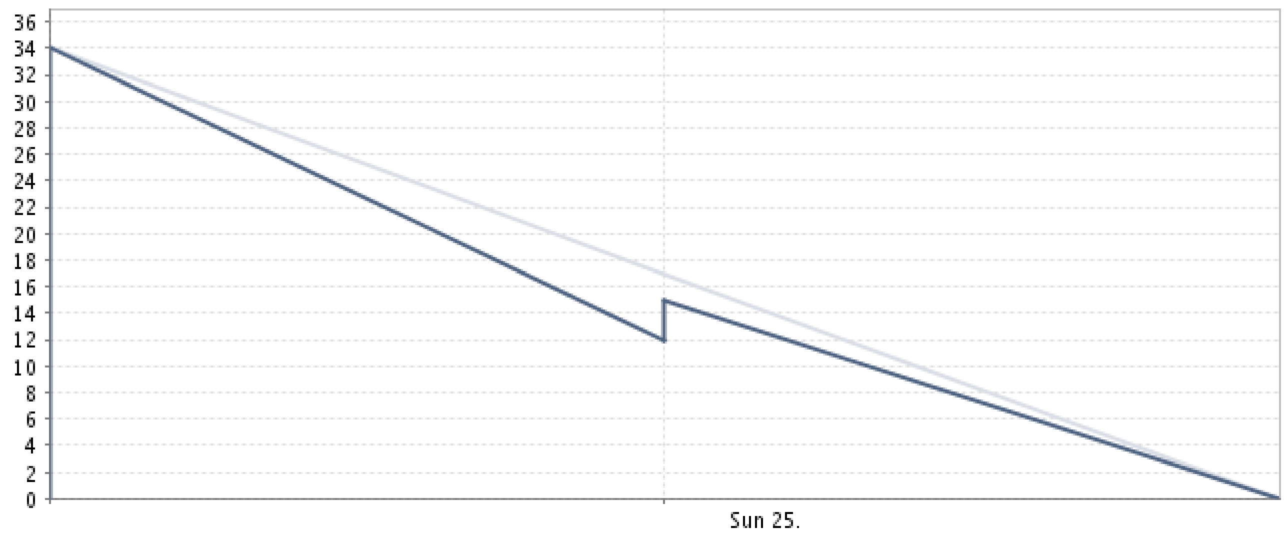
Sprint Review

This sprint was very short and was used to finish the last bit of functionality. Various small issues that needed fixing were pointed out by Fred. These were promptly fixed by Antti and Conrad. Implementing historical stock data caching had kept Conrad, Ivars and Luke busy, but the reduced latency and network load in the final product was worth it. At the end of the sprint, Fred approved the final increment and the product was deemed appropriate for shipment.

Sprint Retrospective

We were impressed with how much we accomplished in such a short span of time and were satisfied with the final product.

Here is the final burndown chart:



8: Reflective Summary

SCRUM Positives

Teamwork

SCRUM’s emphasis on teamwork was the most helpful. We saved lots of time by sharing ideas and expertise as other team members frequently had an idea or solution to contribute when someone got stuck on a problem. For example, Antti suggested Fred read W3Schools to become better acquainted with html code. In addition, we found that impediments were solved much faster if they were brought up in meetings. Finally, pair programming was employed to great effect, particularly when working with interfacing code.

Progress Tracking

Progress tracking is also heavily emphasised in SCRUM. Burn-down charts and sprint logs provided us with incentive to stay on task and increased our confidence in finishing sprints and more importantly, the whole project on time. Sprints were a good way of organising elements of the software system with associated levels of priority. Furthermore, sprints forced us to break user stories into manageable pieces of work. The obvious benefit of this was that tasks were much smaller and easier to tackle and divide among team members.

Daily Stand-ups

We also thought that the daily stand-ups were useful. They enabled the team to share ideas with each other which ultimately led to a better and quicker solution. For example, Ivars learned about the BeautifulSoup library to facilitate the parsing of stock tickers from Antti and Conrad in one of our early stand-ups. Some early standing meetings were not as useful as we would have hoped, because in early meetings, we were occasionally unprepared to speak and to act. As the project progressed, we all made a conscious effort to be more prepared for the standing meetings and overall found them to be useful in staying on task and sharing ideas.

Sprint Reviews

The team felt that the sprint reviews were useful to make sure they were developing the correct product. We felt we saved a lot of time making adjustments from the product owner's suggestions. If he had suggested them at the end of the project as in a plan-driven approach, this could have been extremely problematic and time consuming.

Sprint Retrospectives

We felt the sprint retrospectives were useful to encourage us to look back on the past sprint and discuss what went well, what did not, what we liked in our SCRUM workflow and what we would like to change. As a result, they fixed some issues we had such as preparing for daily stand-ups, having daily stand-ups at the same time, improving our effort estimation, and defining very detailed and thoughtful tasks.

Test-Driven Development

We all thought SCRUM's emphasis on test-driven development was extremely helpful for detecting bugs early and developing a test suite that proved beneficial in later stages of the project.

SCRUM Negatives

SCRUM is difficult to implement *in full* in a university environment: Due to other modules' varying workloads, we could not have a daily stand-up at the same time every day, or sometimes at all. As is clear from our burn-down charts, daily worked hours varied greatly, with lots of work generally being done on weekends when our schedules permitted us to work and meet. In an industry setting, this would likely be different and easy.

The only technical problem we had with SCRUM was its rigidity in not allowing tasks to be added to the sprint backlog during an ongoing sprint. Sometimes tasks took less time than expected and the team members who had claimed them were "out of work" until the sprint's end. They often ended up continuing with a logical addition to their work, the task to which would only be added in the next sprint. Longer sprints with larger tasks would be less susceptible to this problem. Other times unexpected problems would appear and require immediate fixing, so being able to add tasks for fixes would have been useful.

Another issue we found with a SCRUM implementation for this particular project was that the sprints were too short. As a result, it was very difficult to deliver a functional incremental delivery in a week or less.

Would We Use SCRUM Again?

Despite the drawbacks outlined above, our team concluded that SCRUM would be a highly suitable methodology for large-scale development projects. Our group believes that the positives definitely outweigh the negatives outlined above. Furthermore, the main problem we had with SCRUM was that it didn't seem as practical for this current project because it was fairly small. On a much larger scale project, we thought this negative would in fact turn into a positive because of SCRUM's focus on planning and incremental delivery. Thus, in sum we all had a positive experience using SCRUM for this project and would feel comfortable using SCRUM again in the future.