**Dished-Out: Automated Docketing System**

Your-name

Your-Student-ID

Report for CS353 2017

B.Sc. Computer Science and Software Engineering



Department of Computer Science

National University of Ireland, Maynooth

Co. Kildare

Ireland

Lecturer: Mr. Kevin Casey

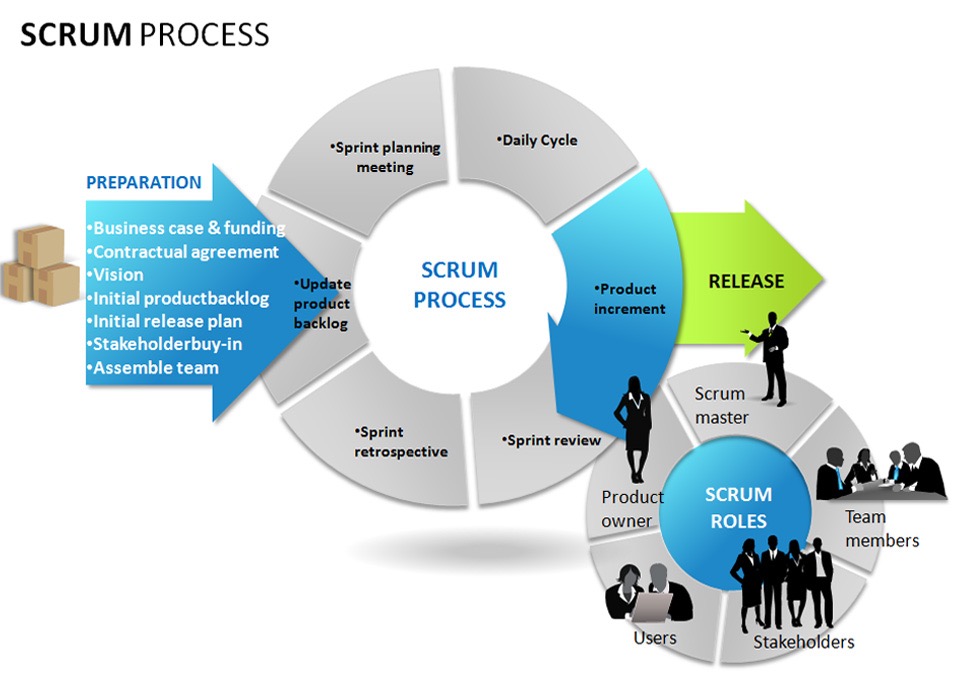
**Chapter 0 - Introduction to Agile Development**

Agile development is based on the idea of iterating on an idea and taking continuous feedback to get the maximum business value in the shortest time. This results in a low cost of change to the project and less wasted time as you can course correct much more efficiently than with other development strategies.

The Agile development method that we used for this project was the Scrum Agile Process. This process allowed us to continuously review our working software, correct and collaborate with each other. We did this roughly every two weeks. Scrum is absolutely reliant on effective communication between all of the people involved in the project, so great communication between team members is key to successfully utilizing the Scrum Agile process.

**What is scrum?**

Scrums origins date back to 1993 when Jeff Sutherland hosted the initial scrums at Easel Corp in 1993. Jeff Sutherland co-wrote the definitive scrum guide with Ken Schwaber. The main benefit of the Scrum Agile Process to the receiving customer is that the customer can, and will, change their minds about what they want out of the software as the weeks go by. With this method of development, you can adapt to change and make the software that your customer wants. The definitive scrum guide details how you can use the Scrum. Ken Schwaber also co-founded Scrum Alliance in 2002 with Mike Cohn, which was initially with the Agile Alliance.

****

The Scrum Agile Process is used by several big companies, for example, Microsoft, Google, Philips and Yahoo all use this method of Agile Development. It is mainly used for commercial software, but also has use cases for video game development, websites and financial applications, among many others. Since the product progresses in a series of month long sprints, with Scrum you focus on the key elements of the software, which are stored in a product backlog. These requirements are captured as items in the product backlog, and Scrum uses constantly changing requirements to create an environment that can quickly adapt to change to deliver the product required. This results in a smoother process than, for example, the waterfall model, which uses inclusive rules, therefore you cannot adapt to change as efficiently with this model. This results in much longer development time. This can be also delayed even longer if the client changes their mind about what they need in the middle of the development process. Scrums low cost of change gives it a huge advantage over Waterfall.

**Sprints**

Scrums are progressed through a series of sprints. This is the crux of the Scrum Agile Process, and they usually last around two to four weeks. Most of the time, you keep the sprint length the same for the duration of the project to ensure consistency in work output. A sprint consists of taking user requirements and turning them into tasks. These tasks are then completed in the sprint, with daily scrum meetings to review what has been done, what the plans are for the future and foreseeing any potential roadblocks that may appear in the sprint ****and how to avoid them.

The product is designed, coded and tested in the sprint. The main goal of a sprint is to have working software at the end of the number of weeks agreed upon, with a use case checked off for the project. You can change the duration of a sprint depending on your goals for the sprint, as you should try to commit to not changing the goal of the sprint in the middle of the current sprint.

There are three different roles in the Scrum framework: The Product Owner, the ‘SCRUM Master’ and the rest of the team. The product owner is the person who defines the features, release date and content of the product, and is responsible for the financial gain of the software developed. They also accept or reject work results after sprints, and prioritize features based on the real world financial benefits of the product that is made at the end of the development cycle.

The SCRUM Master has the managerial role over the team. They must ensure that the principles of the Scrum Agile Process are used to ensure successful progress of the software throughout the development of the project. They should make sure that the team is functioning correctly and that all the team is contributing and communicating. The SCRUM Master also encourages co-operation between everyone involved in the project, which in turn helps the productivity of the team.

The team makes up the rest of the Scrum framework. The team usually consists of around five to nine people, and consists of all the members that contribute to the functions of the project. For example, testers, programmers, UI designers and database administrators. If the members are trained in a multitude of positions, they can swap roles, but this should only be done between sprints, not in the middle of them. Team members can also be a scrum master, and vice versa.

**Sprint Planning**

When planning a sprint, the team agrees upon items taken form the product backlog that they can commit to completing in the sprint. These items must be feasibly completed in the sprint, and then a sprint backlog is created with the goal of completing as many of these tasks as possible within the sprint duration. These tasks are individually identified, and each is given an estimated time in hours by each team member individually, then a time in hours is agreed upon by the entirety of the team.

The daily Scrum is a roughly 15-minute meeting where everyone answers three questions. What did you do yesterday, what will you do today and is anything in your way? The SCRUM Master ensures that the answers are kept quick and to the point, which establish an efficient meeting. This Scrum helps avoid other unnecessary meetings, and provides complete transparency on what everybody goals are and what they have accomplished thus far.

At the end of each individual sprint there is a Sprint Retrospective in which the group periodically look at what is working and what is not working. They are roughly 30 minutes in duration and the entire team participates in the Sprint Retrospective. The group gathers and discusses what they would like to start doing, stop doing and what they would continue doing.

**Product Backlog**

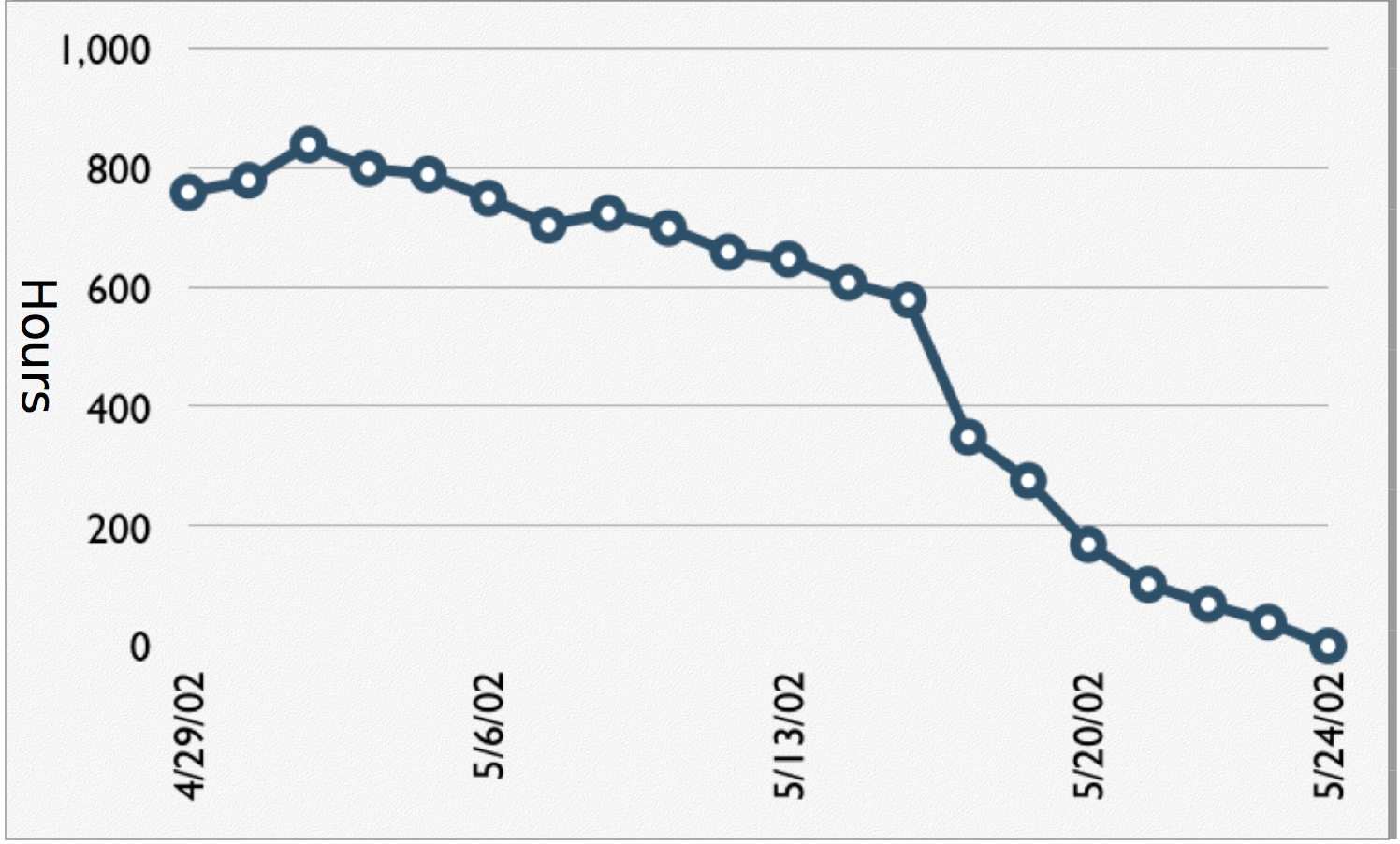
The product backlog is the list of all the required and desired work on the project. The requirements are ideally expressed such that the items have significant value to the end user of the product. The product owner orders the product backlog and prioritizes the more important elements of the product. This can change after every sprint based on feedback and how things are progressing.

To manage the Spring backlog, individuals involved in the Scrum framework chose which requirement they would like to tackle in the Sprint. The SCRUM Master or the Product Owner do not assign any work to the team members, instead the team members decided themselves on what work they would like to do. The team estimates the time that it will take

to complete the tasks in the product backlog using a method called ‘Planning Poker’.

This involves assembling everyone involved and giving each member a deck of cards like those pictured above. The moderator reads a description of the user story and each team member selects a card. The cards are flipped over and the team deliberates which hour estimate they should assign to the user story.

The sprint user story estimates are added up and tabulated into a burnout chart. This chart should approach 0 as the weeks go on and work is getting done. However, there can be spikes in between sprints as the Product Owner can add more user stories to the requirements. A burnout chart usually looks like this:



**Chapter 1 – Project Proposal**

**Chapter 2 – The Actual Project**

The choice of what project to undertake was difficult because every member of the team had interesting ideas, each with their own uniqueness. However, after extensive discussion about the positives and negatives of everyone’s proposals, including further enhancements and feature support in the future, we decided collectively to proceed with Conor’s proposed idea of creating an automated restaurant docketing system, which would be called “Dished-Out”. This is because the idea seemed to be thought right through and focused more closely on the enhanced experience with the potential customers. It took advantage of a gap in the tourism market that is heavily pressured across the globe. In addition, multiple members of the team have worked or currently work within restaurant environments and thus this project had a more personal attachment to us.

After agreeing on what application to create, the team collectively broke down current features that were to be implemented and what new features we could implement if we had extra time at the end. These were recorded and made into a story map later.

**Architecture**

For this project, there was a choice of many architectures to choose from. Some of these included the LAMPP stack, the MEAN stack and the Bitnami Hosted stack. The LAMPP stack (Linux, Apache, MySQL, PHP/Python) was a great option for us all as most of us had worked on it in the past and understood how it worked. However, we were conscious that the application we wanted to create was going to be a multi-page application and the LAMPP stack isn’t the best form of architecture for this. (Carey Wodehouse, 2015). The MEAN stack (MongoDB, ExpressJS, AngularJS, Node.js) was also a promising architecture to choose, and became the option for if we could complete the project using a server. However, we decided to choose a derivative of the Bitnami Hosted stack as it is cloud based and easier to work with. This would be used as a static host if all came to all.

**Features and Functionalities**

For the project to contain basic functionality, certain pages would need a core function or functions. Firstly, it was decided that we would be a creating a multipage application that contained a total of five pages – an initial login and signup screen where the user could log in, sign up, and change their passwords if they forgot their credentials.

The next page would be for navigation and would be portrayed upon logging in or signing up, functioning as a navigator to the user’s respective UI for their work position. This page would contain page transitions to a waiting display, management display and a chef display. A small clock will also be added so all members of staff can manage time across the workplace.

Within the management display, managerial staff could create the restaurant’s menu from a blank canvas, whilst also being able to delete added items if the menu was ever changed in the future. Adding an item should ask for the item and the price it costs, as well as containing a means of knowing whether the entered item is a starter, main or dessert. There should also be a “total table count” button to initialise how many tables the restaurant has.

The waiting staff display will be the centre focus of the application. This page will be a bridge between management and chefs. Firstly, the page needs to be dynamic, being able to fit on a tablet screen no smaller than 7 inches. On the screen, the menu that management created would be pulled and displayed according to starters, mains and desserts. The table count will also be pulled from the database and displayed in a dropdown. Clicking on an item would add it to the customer’s order, and increment and decrement buttons would determine how many of each item is needed. A simple order button would send the order to the chefs.

The chef display should pull all ordered dockets per table from the database and display them within relevant sections, labelled starters, mains and desserts. This split is for organising orders amongst staff. If someone is on starters, they will only see starters for each table. In addition, as the orders come in, they will be displayed as blank dockets until an “Order In” button is pressed, then showing the customer’s order. This will be to determine if the order has been started or not. Finally, a “Done” button will symbolise the order is completed, and clear that docket from the order display while simultaneously sending a notification to the waiting staff saying the order is ready.

**Technologies**

At this stage we all had a clear idea on the end goal of the application, so we then summarised the many possible technologies that we could use to build this product. There were many avenues that could be taken for both front-end and back-end.

Initially, the team had decided that we could utilise the React library as part of our front-end technologies. However, after a meeting with lecturer Kevin Casey and numerous demonstrators, it was found that the project would take longer than we thought, and learning React might cause too much confusion and push our timeline back significantly. In addition, it was found that a member of our team had no experience with web languages or frameworks before. So, to keep focused on the end goal and have a completely functional application, we decided to keep our technologies simple. We decided that using HTML (Hypertext Mark-up Language), JS (JavaScript) with jQuery (a cross platform JavaScript library that helps simplify the client-side scripting of html) & CSS (Cascading Style Sheets) would be the best option.

When deciding what back-end technologies to use, we wanted one main function, which was speed. As dockets come and go very quickly, we needed speed to prevent lag. So, after some researching between SQL databases and NoSQL databases, we concluded that using the Firebase Realtime Database was the best we could use. As the database is real time, we would see results instantly across all displays within our application. It is hosted in the cloud and is fully compatible with JavaScript SDKs, which was very useful for us with syncing up our work.

However, as nobody had used a NoSQL database before, plenty of research was going to be needed to understand how the database ran. Also, we quickly realised that if we used this database, we would need to plan how we wanted our database to look and function as one mistake could cause major disruption later in the project.

For hosting, we decided that if we could go the dynamic route, we would use node.js. However, if we couldn’t implement this, we would remain static and use a static host such as bitballoon.com.

**Testing**

The purpose of testing the application and any functionality added is to detect where any points of failure exist so the problem can be discovered and either corrected or removed and re-done. As the project would continue, we realised that we would need to test the application as we completed the major pieces of functionality. So, we began writing up some tests that we would use to show that the application worked as it was meant to while also fixing any errors that occurred during the process.

The first test was simple but had major implications if any failures arose. This was to show that each page could communicate with each other and transitions could be made from one page to the next and back again. This was the core of everything we would code as no one page should work on its own for a user. So, we specifically decided to test if we could log in or sign up to the application and transition to the navigation page. This was where the heaviest piece of code would be and would be more likely to have a problem.

The second test that was empirical to the success of the application was to show that a member of managerial staff could make a menu and send it to the waiting staffs page. Specifically, we would test that management could set the table count, set the current courses, add items to the menu in the appropriate location and that the item does proceed to the waiting staff display and delete an item once the menu is made and needs alteration. The reason for testing this piece of the application would be because this page is the bridge between waiting and chef displays and, like our previously made test, would be very code heavy. This bulk and complexity in code could cause problems for the page and would even highlight if there was a malfunction on any other linked pages.

On completion of these tests, answers will be formed on whether a function needs to be corrected, changed entirely or forgotten about. In order to test our application, we would use the Selenium software which would monitor and record our clicks and transitions.

**Distribution**

At this stage, the application was understood by all team members. Each person knew how the application worked, what languages would be used, what features existed and how we would test the application. All that was left to complete was to distribute of each task on a per person basis. The easiest way to do this was to give each member one page to complete (five pages over five team members). This would ensure that the load would be evenly distributed across all team members. At the end of the project, we would collectively create a common styling for the application to ensure consistency over each page.

We distributed the tasks according to a “who wants to do it” basis. This was because each person knew their own talents, strengths and weaknesses, and so it seemed logical to allocate tasks based off this.

The breakdown was as follows:

* Tosin Salimon: Log-in Display
* Conor Cohen Farrell: Chef Display
* Eoin Duffy: Waiting Staff Display
* Sean O Grady: Navigation Display
* Connal Lynch: Management Display

All team members agreed to this breakdown as it reflected, in each person’s opinion, where their skills focused more. Each member decided to learn how to use firebase independently and, if necessary, help others in understanding the database workings. Furthermore, as we would collectively create a common stylesheet to use, the CSS per page during creation would be confined to placement of objects on the screen and UX (user experience) rather than how the final application visually appealed to the user.

**Chapter 3-The SCRUM Process**

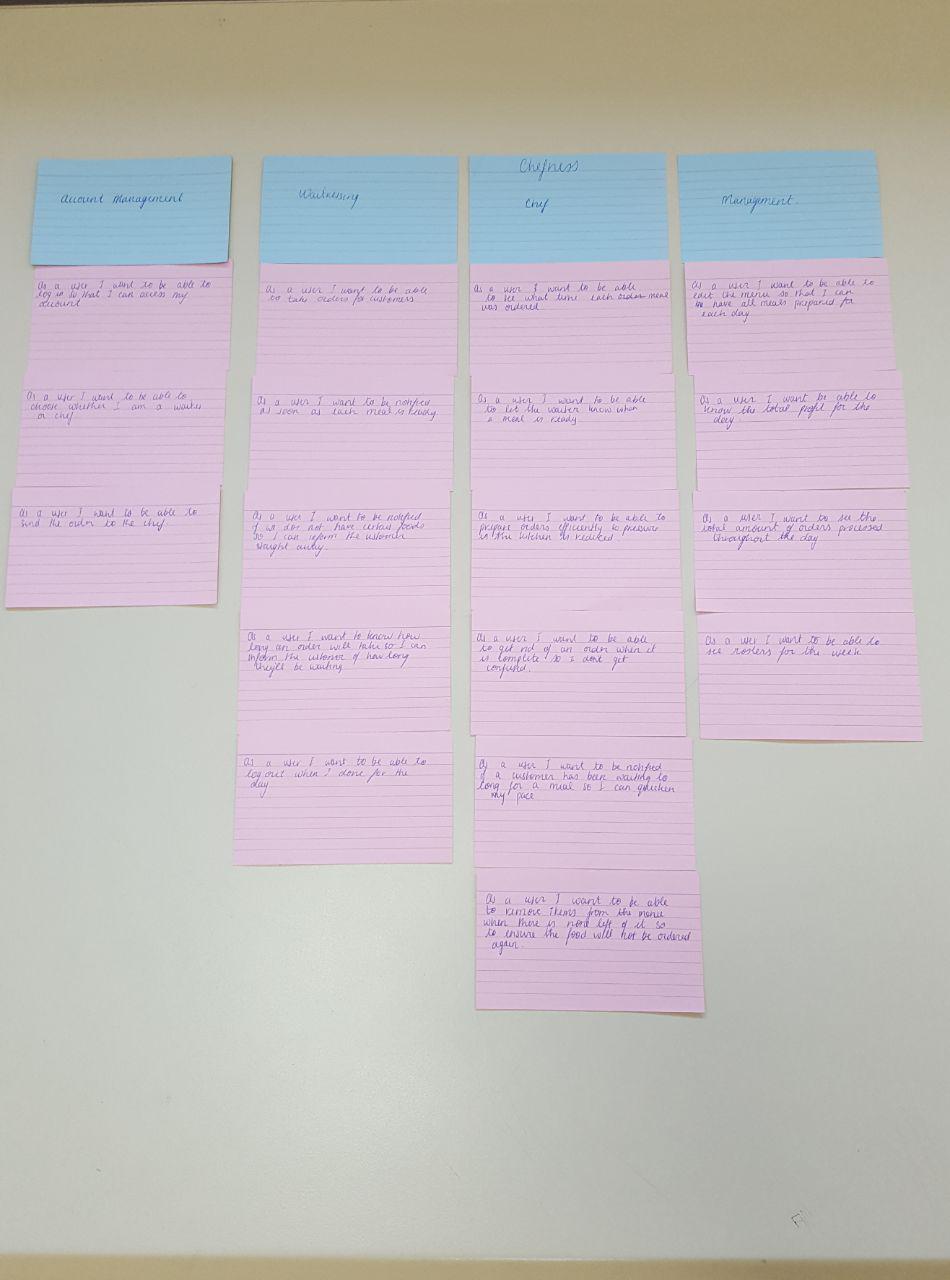
**Initial SCRUM Meeting**

For everyone to stay focused on the tasks that have been completed and what tasks need attention, we participated in having SCRUM meetings each week. SCRUM is an agile process that allows us to focus on delivering the highest business value in the shortest timeframe. As part of this, we divided up everything to be done over a two-week basis, with every fortnight being a sprint. We then held a weekly scrum meeting over the course of each sprint. Meetings were held every Wednesday during lab times and were also done on Thursdays in the same location each week. Each meeting within the lab lasted approximately 15 minutes and each meeting outside of the lab ranged from 30 minutes to 1 hour. This ensured that the discussion remained relevant and we did not fray of topic. Having the weekly SCRUM gave each team member the opportunity to show their progress over the course of the week or explain to the other team members any issues they had during the week. One member of the team was assigned the position of SCRUM Master each week. The job of the SCRUM Master was to gain an insight into how the project was operating, asking each team member three main questions that were broad but to the point. These questions allowed all team members to understand what stage the project was at, and they included:

1. What did you do last week?
2. What do you plan on doing this week?
3. Are there issues or problems in your way?

Asking these questions allowed the SCRUM Master to distribute new tasks to team members, provide help where help was needed and manage time, so the product would arrive on schedule. Thus, each SCRUM meeting made all members of the team aware of the progress of the other team members as well of the progress of themselves. It gave us the opportunity to solve any issues that may have arose for a member during the week.

Once we chose our group project in week 1, the next step was to create a set of user stories. “User stories are short, simple descriptions of a feature told from the perspective of the person who desires the new capability, usually a user or customer of the system” (Mountain Goat, 2016). Each member of the team created these user stories individually. We had our first SCRUM meeting during the second week. As this was our first meeting it lasted for approximately three hours. At our initial scrum meeting each user presented the set of user stories they created over the week. This brought to the table many different ideas that could be included into the project. We discussed each of these functionalities and how it would be possible to implement them into the project. By the end of the first hour we agreed on a set of user stories to which we would follow when implementing our project. Using our set of user stories, we created a shared story map. Story mapping is an approach to organize and prioritize user stories. We then added task-centric stories under each activity which explained what would be typically done under each activity and in order of which one would be done first. Each activity had multiple task-centric stories since each activity could do more than one task. The story map helped us gain a better understanding of the end-to-end use of the system for the end user. Creating the story map gave us a starting point and a guideline of each tasks we had to implement for each activity.

The story map that we created at the beginning was unfortunately unsatisfactory as we found holes in it and realised that we talked too much about the functionalities rather than what the user would want from the application itself. So, we decided to enhance the story map, organising it better and showing it from a user’s perspective.

**Planning Poker**

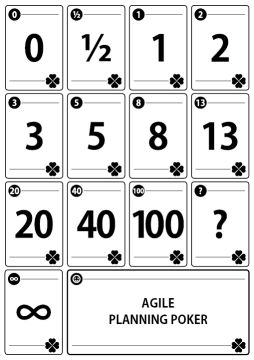
Once we knew the tasks that need be complete we had to estimate how long each task would take. To estimate the length of time each task would take we used planning poker. “Planning poker is an agile estimating and planning technique that is consensus based.” (Mountain Goat, 2016). Each team member has a deck of planning pokers cards like the one shown in Figure 1 below, with values 0, 1, 2, 3, 4, 8, 13, 20, 40, 100. The values represented the number of hours in which each team member thought it would take to complete each task. Tasks from our story map were read aloud and the team privately selected one card to represent roughly how long we believed each task would take. Then all the chosen cards were revealed at the same time. If we all selected the same value, then that value became the estimate. If not, then we discuss our estimate and why we chose it. Once we got an insight as to why everyone chose the value they did we reselected a new value, and again revealed it at the same time. This process was repeated until we came to an agreed estimate for each task. Planning Poker was an easy and fair way to estimate the time required for each task. By using planning poker each member of the team got to have an opinion on each estimate which led to more accurate and realistic estimates.

Figure 1: Planning Poker Cards

**Meeting Minutes**

During each of our weekly meeting we kept a record of what we discussed. During each SCRUM meeting, one member of the team was assigned to keep a record of what each meeting consisted of. This person was rotated every week. It was important to keep a record of each meeting for later reference. At the start of each meeting the minutes and contents of the previous meeting would be quickly mentioned to reminded us of what we talked about the previous week. By keeping account of each meeting, we were able to follow up on any issues that occurred the previous week and to see if they were resolve. We had a total of thirteen meetings over the course of the project. A summary of the minutes for each of these meeting was as follows:

***Meeting 1: 04/10/2017*** This was our first meeting after we chose what project to pursue. The aim of this meeting was to ensure that we all knew the purpose of the project, the functionalities to be included in the project and an estimate of the time it would take us to complete the project. We used user stories, story maps and planning poker for this meeting. By the end of this meeting we all knew the complete objective of our project and an estimate of how long we wanted to spend on each task.

***Meeting 2: 11/10/2017*** We started this meeting by ensuring that we were all up to date on the progress of our project. Since we knew all the functionalities our application required, we began dividing tasks between each team member. Since the application required five user interfaces and there was five team members we decided to give one UI to each member. We aimed to start building each UI for the following week. We also engaged in a meeting with lecturer Kevin Casey and portrayed our story map to him. On further inspection, we realised that we needed to rewrite our story map as we had too much information on functionalities rather than what the user wanted to have at hand on completion of the application. This was then redone in the latter half of the lab that day.

***Meeting 3: 18/10/2017*** At this meeting, each member of the team showed their progress to everyone else. Everyone had started their work on the UI’s, but no functionality had been added. Each student had a guideline of how each page was going to work and where each functionality would be. During this meeting we made sure that everyone was happy with each UI and if any changes were needed before implementing the functionalities for the application.

***Meeting 4: 25/10/2017*** Due to lack of communication over the previous week, there was not much progress. Over the week it was thought that we all had to work on the same UI. So, at this meeting everyone had focused their efforts on a single UI. Due to this we decided that we needed to communicate better both in person and through telegram, so we decided that as well as the weekly meetings we would meet up for an extra hour later in the week. This was to ensure that everyone was up to date and so that if there were any changes or issues we could try to resolve them before the next weekly meeting.

***Meeting 5: 8/11/2017*** For this meeting each member of the team had to present the work they had done over study week to all other members of the team. A good start on implementing the functionalities had been made on most of the UI’s. We discussed the problems that were in our way. One of the major problem was that most team did not understand how firebase worked. “Firebase is Google’s mobile platform that helps you quickly develop high-quality apps and grow your business”(Google, 2017). Four of the UI’s required firebase, one using authentication and the rest using the database for inserting and pulling information. At this meeting we decided that we would all take the week to gain a better understanding of how firebase worked so that we would be able to implement it into our application. As Conor seemed to understand how firebase worked, he offered to help everyone if they required help. Another major problem was that a team member had broken their laptop over the week, therefore losing all the work they had done. This meant that we were behind on one of the UI’s, so to solve this problem some team members met up later in the week and worked on this UI while another team member began working on adding the lost functionality.

***Meeting 6: 15/11/2017*** At this meeting, it was found that most of the team had a better understanding of how firebase worked. Some team members had issues trying to implement firebase, so we worked on trying to fix these issues. We decided that we would continue to focus on completing the functionality of the app before looking at the design of the application.

***Meeting 7: 26/11/2017***

This meeting was done so the group could collectively gain a better understanding of firebase, focusing mainly on where problems occurred in members code. The members that could come followed up by fixing their code and sharing resources with other members of the team to help them complete their database interaction functionality.

***Meeting 8: 22/11/2017*** At this meeting we decide to start looking at how we were going to style our application. We collectively agreed on a set style that each UI would have and planned to include it on each page for when we met up later in the week. At this point we were on track with our project, but we did decide that we needed to start thinking about hosting and looking at the group report in more detail.

***Meeting 9: 23/11/2017***

During this meeting, we continued our progression towards a concurrent application design, ensuring that each member of the team present knew the format and layout of how the final application should look. We began looking at backgrounds for the application as well as what colour schemes we would use.

***Meeting 10: 29/11/2017***

At this meeting all the functionalities had been added and there was a set style on each UI. Each member displayed there working UI’s to the other team members. We made sure each user interface worked as how we expected it to by going back to our story map and looking at the tasks we had planned to include in our final project. We also decided we would meet up later in the week to start looking at hosting.

***Meeting 11: 01/12/2017***

We decided to meet and research what hosting services we could use. We quickly realised that we didn’t use any MEAN architecture, so static hosting was necessary rather than node,js.

***Meeting 12: 6/12/2017*** By this week we had our application hosted. Our application was complete, and we began to practice for our presentation and ensuring that all our tests were completed and still valid. We divided out what part of the application each person would talk about. Whilst preparing for our presentation we decided to also record our screencast while we were all together. We decided that we would meet up at the beginning of the following week just to have a run through the presentation. This was our last weekly meeting.

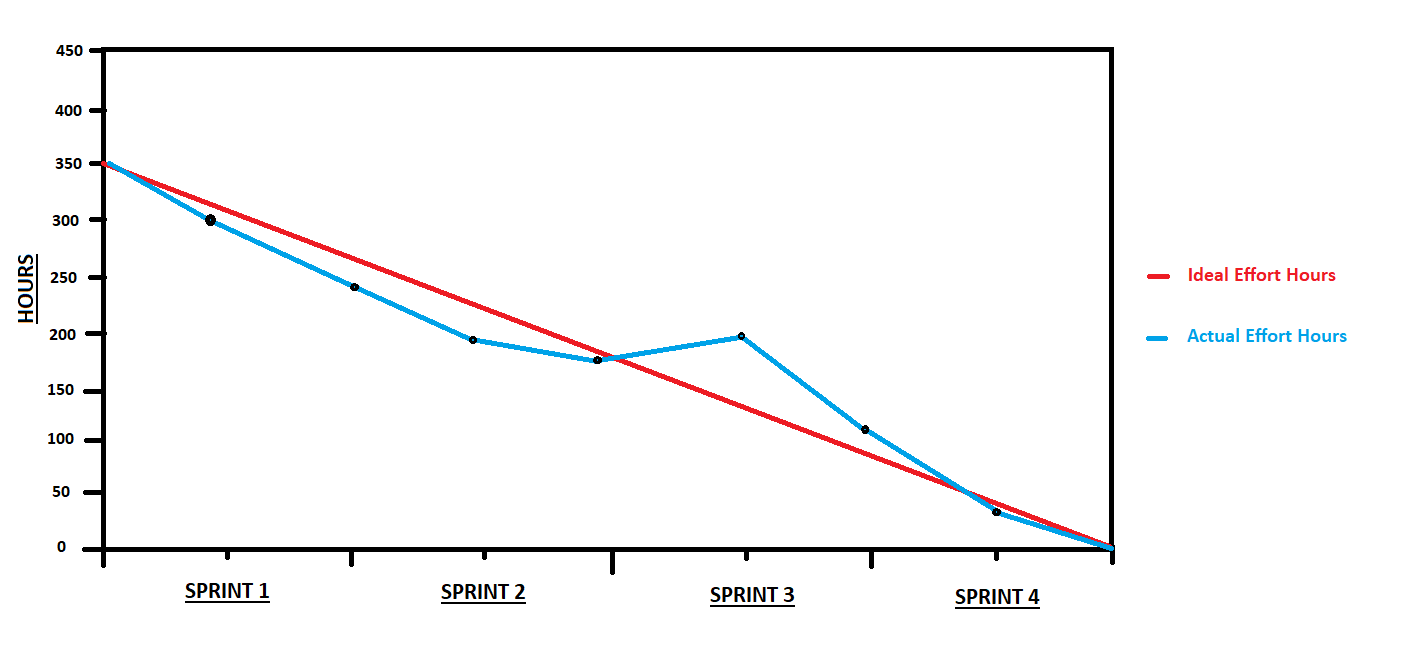
***Meeting 13: 11/12/2017:***

We decided to meet up one last time during the week. For this meeting we focused on talking about the report. As we were going on our midterm break we made sure everyone understood which parts of the group report they were doing. We set a deadline for when to have the group chapters done by so that the other team members could have it included in their project on time. We also ran through our presentation aloud in a demo fashion, ensuring that we knew what we wanted to talk about, when we needed to speak and if the presentation filled the 10-minute time limit allocated.

**Burndown Chart**

The burndown chart allows us to easily visualize how much work is left in our project in a simple way to read and view. We originally projected to get most of the work done in the early sprints, and this came to fruition. Having the burndown chart as a reference and consistently updating it after every sprint, we could see that we were ahead of schedule, so we may have done too much in the first sprint. However, when the third sprint came, we realized that we would no longer be ahead. A member of our team lost all his work due to a computer hard drive failure and thus pushed us back significantly as the whole team came together in order to get back up to speed as quickly as possible. It was quickly noted that we managed to get back on track and mid-way through the fourth sprint, we were slightly ahead of schedule again. This was necessary as we felt that hosting could take longer than anticipated.

However, we managed to complete the project right on time, despite the minor inconvenience thrown our way.



**Story Maps**

In our first week, we collectively decided to go with the idea of a multi-page web application to manage restaurant dockets named Dished-Out. We created a Story Map together to paint a picture of all the use cases that we aimed to get into the application by the end of the development process. We designed it with the intent of changing it as the weeks had gone on and react to change in the development process. However, it was clear that the design we had was ambitious but manageable, so the Story Map did not change as the development cycle went on. We found that we had enough functionality to fit the 10-week period. The story map was used to keep the team on track with tasks that were to be done. Our story map from the looked like this:

| Account Management | Waitressing | Chef | Management |
| --- | --- | --- | --- |
| As a user I want to be able to log in so that I can access my account. | As a user I want to know how long an order will take so I can inform the customer of how long they will be waiting. | As a user I want to be able to prepare orders more efficiently to decrease the pressure in the kitchen. | As a user I want to see the total amount of orders processed throughout the day. |
| As a user I want to be able to send the order to the chef. | As a user I want to be notified as soon as each meal is ready. | As a user I want to be notified if a customer has been waiting too long for a meal, so I can speed up the order (time keeping). | As a user I want to be able to see the rosters for the week. |
|  | As a user I want to be notified if we do not have certain food, so I can let the customers know as soon as possible. | As a user I want to be able to get rid of an order when it is complete, so I don't get confused. | As a user I want to be able to know the total profit for the day. |
|  | As a user I want to be able to take orders for customers. | As a user I want to be able to see what time each order was taken. | As a user I want to be able to edit the menu, so I can have all meals prepared for each day. |
|  | As a user I want to be able to log out when I’m done for the day. | As a user I want to be able to remove items from the menu when there is none left of it to ensure the food won’t be sold. |  |
|  |  | As a user I want to be able to let the waiter know when a meal is ready. |  |

There were a few adjustments to the story maps as the weeks went on, but nothing major was changed. We were fortunate that Conor had a clear vision as to what this application would end up being and how much time we had to finish it. Some changes included not being able to remove items if they are out of stock, having a total profit for the day (although this has been set up, just not completed), having a roster system and having total orders for the day recorded (nearly ready for use, just not completed).

Overall, the main functionality of the application did not change from the first week. We made some adjustments to the management page due to time constraints and other obstacles that came in our way such as loss of code and data, but we are collectively very proud of what we have created with this application.

**Testing**

During the progression of our application, it was necessary to test specific pieces of the code to ensure they worked as per the specification. For this, we had quite a few tools we could use, some of which include Ranorex and TestComplete. However, we decided to use Selenium, a software testing framework in conjunction with Google’s Chrome browser. It is widely used within major software companies for testing, as well as for other features that Selenium supports.

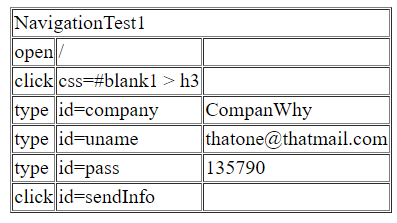
For the first test, we launched our hosted application on Google Chrome and ran the Selenium software. The purpose of this was that Selenium would mimic our clicks and replay them to ensure the application was working as expected. The first test was to see if we could properly transition between pages after authentication. This test passed, as the results in figure 1 show below.

Figure 1: Test 1

However, after running a test like this one where we transitioned to a page and back again, we encountered a problem. As our application utilized the inbuilt back-page button, Selenium couldn’t recognize the transition from one page back to the previous page. As such, when we made changes in management and transitioned back to the navigation page to logout, Selenium showed that the logout button didn’t exist. The software believed it was still on the management page, where no logout button exists. However, this wasn’t seen as an error on our behalf and we moved on.

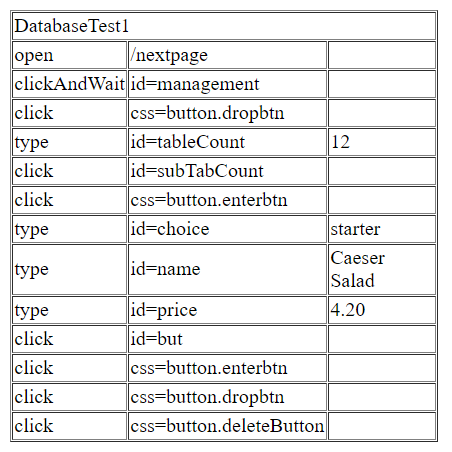
**** The final test was to ensure we could add items on the management page and have that be available for the waiting staff. This test also passed as Figure 2 shows below.

Figure 2: Test 2

**Future Releases/ Sprints**

We have many more plans for Dished Out in the future! Our first objective would be to change the way in which we developed the site. When we first began to develop our site, we did so by dividing out the work among the five of us, each member working on a separate page. Due to this, everything was done differently due to different styles of coding, and different thought processes. This led to us only being able to host our project statically because everything was locally created, and no JavaScript was running from a server. If we were to bring this forward and update it, one of our first steps would be to change the app so that it could be hosted dynamically, allowing our site to work faster and be easier to configure and apply updates to. We also wanted to add a pop up menu to each time a specific course was ordered. This pop up would contain all the possible sides, and toppings, thus allowing the waitress to adjust and to personalise a customer’s order, accounting for allergies as well as “picky” eaters. We would also try to get each individual order to have a timer linked to it, giving the chefs assigned to each course an allotted amount of time to have the course prepared and ready to be served, this would give an alert when the time had run out, letting them know it is time to bring it out to the customer. We would also like to have a register page to act as a cash register for waiting staff and management. This page would have a list of currently sitting tables within the establishment and with the click of a button, their total order cost for their meal would be shown. Finally, we would like to apply all the functions that we didn’t have time to complete during the 10-week period. These include being able to remove items if they are out of stock, having a total profit for the day that management can access, having a roster system for all staff members and having total orders for the day recorded.

**Code Versioning (Git, etc...)**

We handled out our Git in a different way than expected due to each member of the group working on a page entirely by themselves, but in a way that we felt was more efficient. We had all previously used git on personal projects, as well as for other parts of our overall course, and while most people created public repositories and allowed each individual member to push their code to that repository, we created a git where we all knew the password and login details for the account, and we used this to push all our code to, though, we did not do this as often as we should have so our repositories have large gaps where there are major changes from version to version. Each new version was saved in a new repository. We thought the creation of a team git would make the whole situation a lot easier, but it did also lead to some problems. We were unable to see which team members were pushing to git, so were unable to see if everyone was pulling their weight within the project. The only method around this was our communication within labs and meetings, where we understood who contributed to the project and what specifically they contributed.

**Team Communication/ Management**

We had multiple strategies to ensure our communication lines were kept open for the duration of the project. Firstly, we had our weekly CS353 lab session, assigned on Wednesday at 11 o’clock, which lasted for a total of three hours, and was an invaluable resource that dedicated each week to pure development. We spoke about what we had completed, made plans for the next sprint, discussed any issues we may have been having, and then spent the remaining time to progress further on our app.

We created a group Gmail account for the project (group14work@gmail.com) so that we could all access a shared Google Drive, and maintain our report documents together. We used this email to set up our group Git profile, as well as our Firebase. We also set up a group chat on the messaging app Telegram, so we could query and question each other outside of meetings and college. It also gave us a platform on which to update each other about our progress, inform each other of our possible absences, pool resources, and plan future meetings outside of the lab meetings. In addition to the communication online, we made the effort to hold weekly meetings outside of the Labs. These meetings were held on varying days, depending on both our timetables and on gaps in personal time. These meetings allowed us to update one another face to face, also allowing us to explain issues we were having more easily than could be done over text, as well as teaching each other new skills as we went along.

**Chapter 4 – My Contribution**

**Chapter 5 – Summary**

**References**

* Carey Wodehouse. (2015). What is a Software Stack? Choose the right stack for your project. Retrieved December 21, 2017, from https://www.upwork.com/hiring/development/choosing-the-right-software-stack-for-your-website/
* Google. (2017). Firebase. Retrieved December 21, 2017, from https://firebase.google.com/
* Mountain Goat. (2016). Agile Topics | Mountain Goat Software. Retrieved December 21, 2017, from https://www.mountaingoatsoftware.com/agile