



CSC3045 – TEAM REPORT

Team CS16 Up and Scrumpers

Ronan McDaid 40042962
Maeve McLaughlin 40087635
Josh Reynor 40112308
William Kyle Spence 40085249
Gareth Nixon 40041488
Tainbai Peng

Contents

Sprint Artifacts	3
Intro.....	3
Planning	3
Scrum Meetings	4
Pair Programming	4
Hartman Orona Spreadsheet.....	5
Sprint 1 Burndown Snapshots.....	6
Sprint 2 Burndown Snapshots.....	8
Story Point Burndown.....	10
Sprint Reviews.....	12
Project Design Specification (UML).....	13
Class Diagrams	13
Class diagram format:	13
Base Classes (CSC3045AgileDataModels)	13
ClientServerModels (CSC3045AgileDataModels)	14
Entities (CSC3045AgileDataModels)	15
Entity Relationships	16
Code Quality Analysis.....	17
Convention to Coding Standards Document	19
Testing Documentation.....	21
Team Work Attribution.....	22
Known Bugs list.....	25
Critical Reflections	27
Appendix	29
Code/Reference material.....	29
Coding Standards Document	29
Code Layout	29
Indentation.....	29
Spacing	29
Library References	29
Naming Conventions.....	29
Folder Names	30
Class Names	30

Method Names	30
Variable Names	30
Good Coding Standards	31

Sprint Artifacts

Intro

Throughout our project we implemented various agile methodologies that were introduced during the lectures at the beginning of the semester. For most members of the group it was a first experience of working in an agile team so there was a lot to be learned. Also for the members who had used agile during placement or elsewhere it was a good learning experience as every team will use a different combination of agile practices and as a group we had to decide which methods to use and how best to apply them to this project. The following sections show the processes we used in sprint one and two as well as any changes we would make if we were to have another sprint.

Planning

For sprint 1 we held two planning sessions to discuss how to work on the project, the strengths of each team member and to develop a joint understanding of the stories and requirements for the project. As a team we looked at each story to ensure that there was no room for misinterpretation which could lead to complications later on. After this we carried out planning poker by each submitting a copy of the product backlog along with assigned 'planning points' before meeting to discuss and agree on a team figure for each story.

In hindsight a shortcoming of this planning was not setting a formal target of stories to be completed and not allowing time for changes, bugs, SVN issues and other impairing factors. This led to a low output from the first sprint which did not reflect the work put in by the team. Based on team discussion, experience gained from sprint 1 and feedback from our sprint review we made a conscious effort to plan better for the second sprint. Again we conducted a planning poker session, this time face to face so we could allow for better discussion amongst the team. As show in the table below the points we assigned to stories in sprint 2 are much higher than we had originally thought before sprint 1, this realisation allowed us to be more accurate in planning the stories we would be able to deliver.

The improvements in our planning for sprint 2 allowed us to deliver more completed stories and made other processes such as tracking tasks on the Hartman Orona spreadsheet much easier. Despite having two incomplete stories that were planned for sprint 2 it was much more successful. We coped better with obstacles and feel that without the absence of a member of the team we would have completed all our planned stories.

<u>Story</u>	<u>Sprint 1</u>	<u>Sprint 2</u>
Register	3	(completed in sprint 1)
Select Roles	5	(completed in sprint 1)
Log In	3	(completed in sprint 1)
Create Project	2	5
Search/Assign Product Owner	3	3
Search/Assign Scrum Master	3	3
View Projects/Roles	2	5
Product Owner Manage Backlog	3	8
Scrum Master Create Sprint	2	5
Create Sprint Team	1	5
Allocate Stories to Sprint Backlog	3	5
Add Tasks w/ Times to Stories	3	8
Take Ownership of Task	2	3

Scrum Meetings

As a team we were very good at communicating with one another both inside and out of our face to face meetings. One initial obstacle we faced was that English is not Tianbai's first language which made it difficult to convey information in person. We solved this by creating an instant messaging group in order to assure that no details would be lost in translation. We also used this group to discuss our progress and any issues we were having, this helped all the team members to keep up to date with each other and allowed us to keep our scrum meetings concise and to the point. Due to availability of team members and meeting rooms we could not always keep a consistent time and place for our meetings but we felt that being flexible with these was worthwhile in order to get everyone together as much as possible.

Based on feedback from sprint 1 we made changes during sprint 2 to incorporate updating the Hartman Orona spreadsheet after each meeting (where possible) in order to improve awareness of our progress and work remaining and to maintain the spreadsheet more accurately than in sprint 1. We also backed up our instant messaging conversations and face to face meetings by sending update emails.

Pair Programming

In both sprints our team used a lot of pair programming to complete tasks and stories faster and more efficiently as well as using it as a learning tool. We also paired on areas of the project such as table design and user permissions as these were often complex so it was helpful to have two or more team members validate each other's work. In particular Kyle and Gareth often acted as the 'Driver' with one or more observer working with them, this way they were able to share their coding knowledge and experience as they wrote and explained code. As many of our functions had similarities pairing allowed observers to build a better understanding and recognise where code could be reused for other tasks and stories.

Sharing knowledge this way enabled all members to have a better understanding of the project and prevented dependencies on one or two team members which would slow the team down. We used pair programming a lot for bug fixes and found that one member reviewing another's work would find the problem relatively quickly compared to trying to solve the issue individually.

Members of the team meeting often to work in pairs or small groups contributed to our successful second sprint as it kept the focus on a particular area of the project where as in sprint 1 there was work being done on stories when the preceding stories had not been completed. Pair programming was of huge benefit to our group and is something we would continue to make use of if we were to complete another sprint.

Hartman Orona Spreadsheet

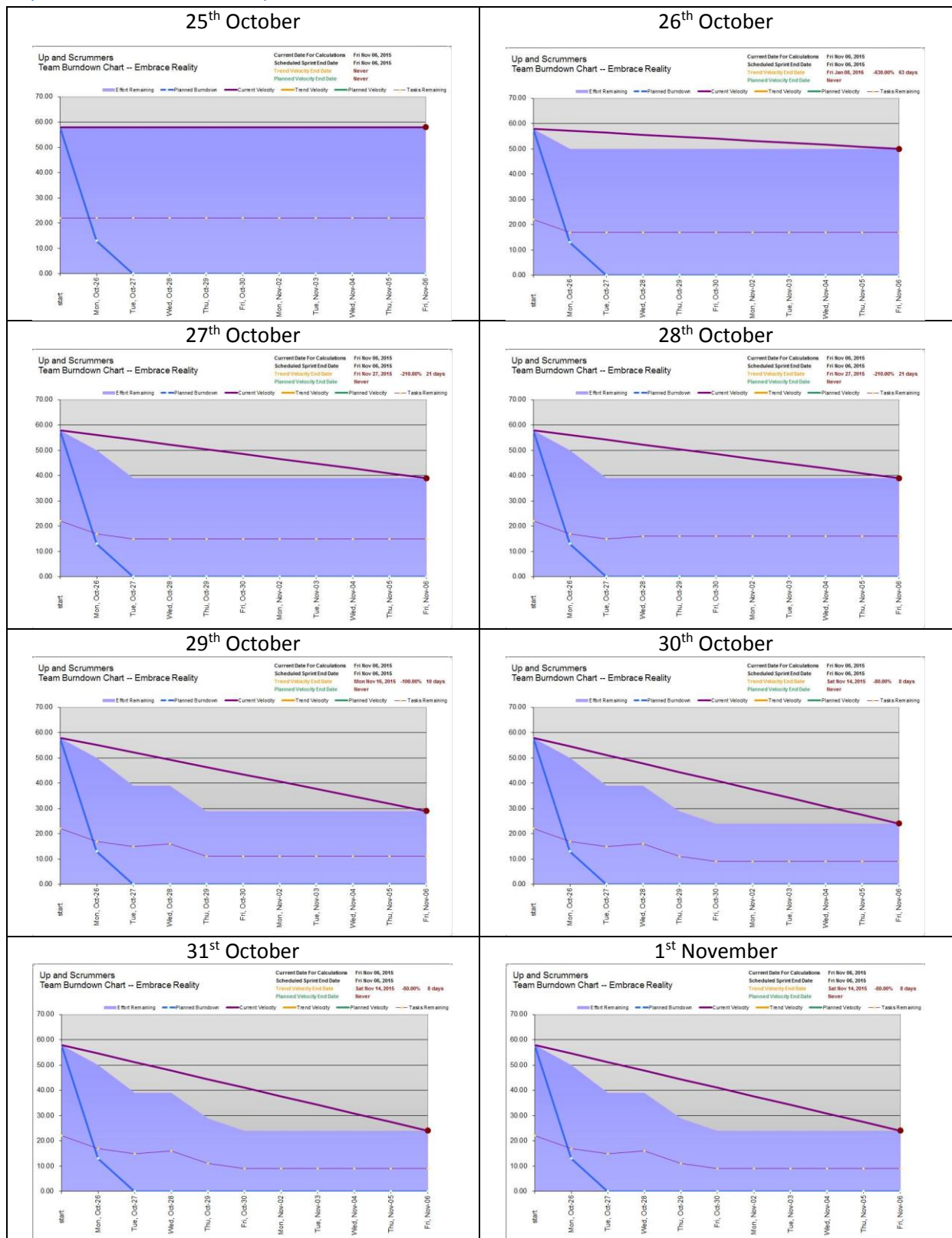
In order to track the progress made on stories, the tasks related to those stories and the hours remaining for those tasks we used the Hartman Orona spreadsheet demonstrated in the lectures.

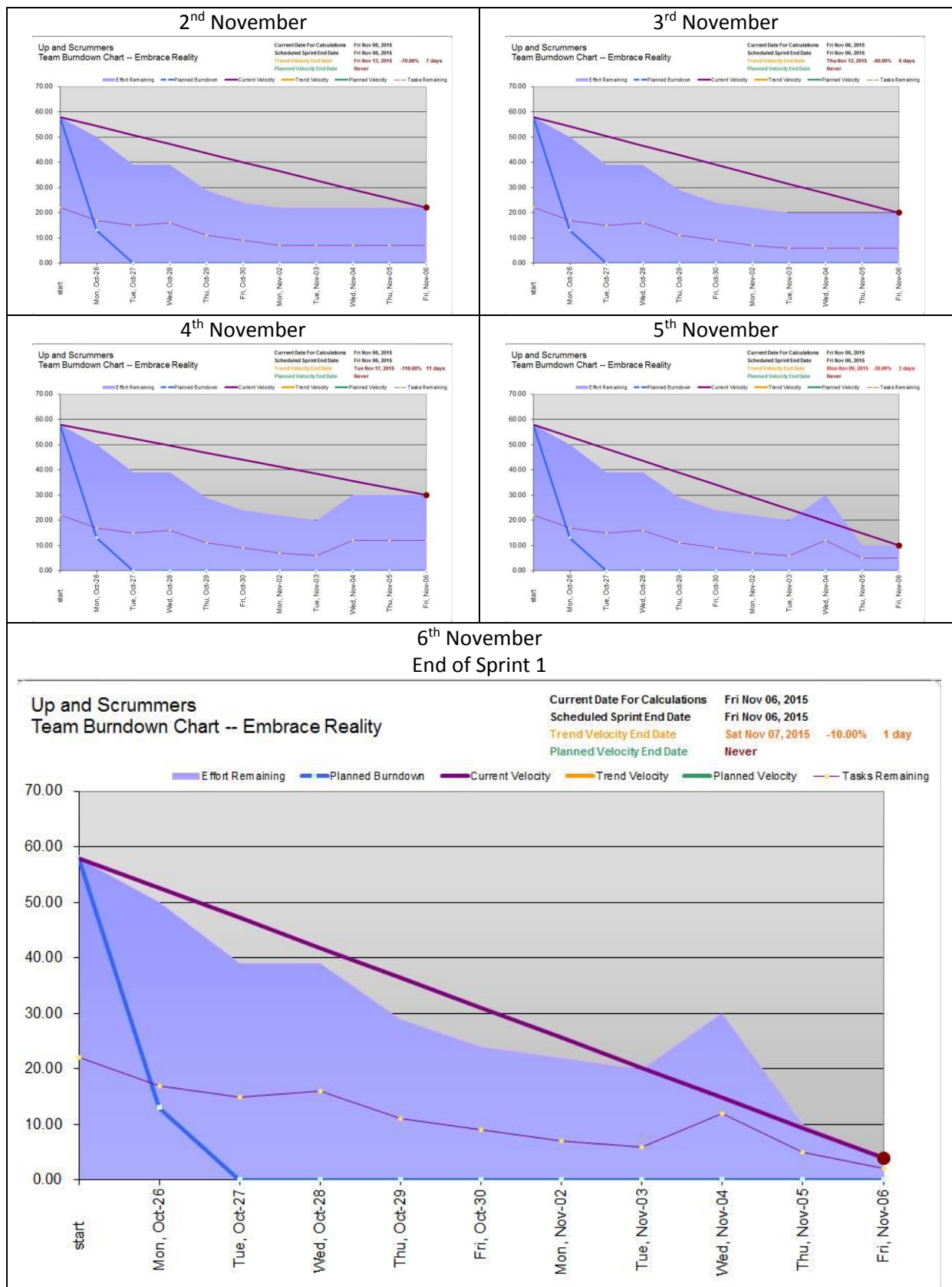
During sprint 1 we saw this as a useful tool in the beginning and attempted to populate it in advance of the sprint but found as the sprint progressed what we were doing was very different from what we had initially thought. As there were so many changes made it was difficult to maintain the spreadsheet efficiently and it was somewhat neglected.

After the first sprint we were aware that we needed to improve our planning and with that be able to organise the spreadsheet better which was also addressed in our sprint review. Having completed the first sprint we had a better understanding of the tasks that would need to be completed in order to deliver each story which allowed us to populate the spreadsheet before the sprint began and add tasks later where necessary. As well as our improved planning we were also more diligent in updating the spreadsheet and made a point of updating it after our scrum meetings if possible, by doing this we accumulated a number of local copies which was slightly cumbersome to combine into one final version but was a huge improvement on the first sprint. Utilising the spreadsheet in the second sprint made us more organised and aware of our progress and what work still needed to be done which was a key factor in our improved performance.

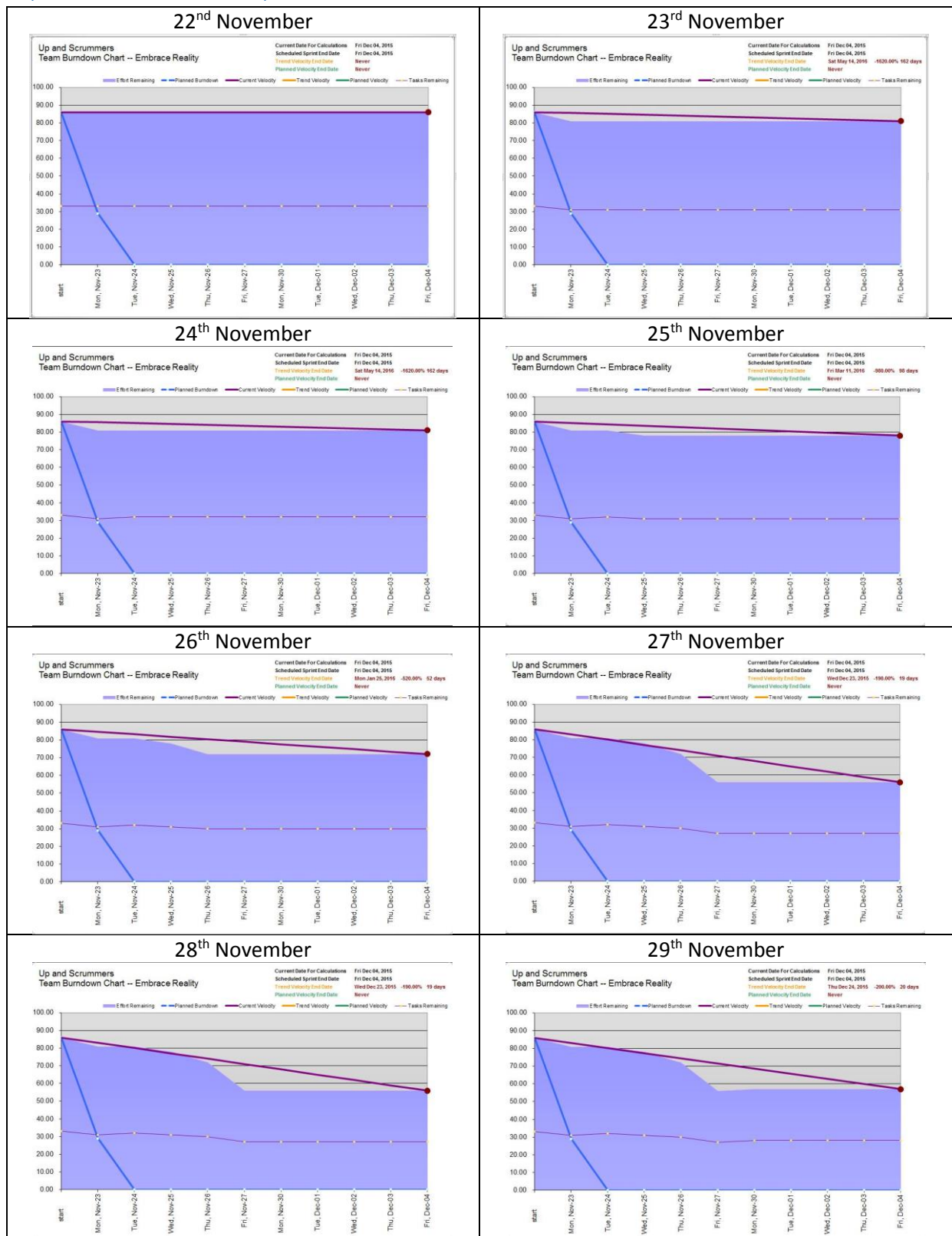
Also in sprint 2 we added another page showing the burndown of our 'planning poker points' to the Hartman Orona spreadsheet. It was implemented showing points burned off each day as opposed to removing all points once a story was complete, a representation of how the later would have looked is included at the end of the next section showing screenshots of our burndown charts each day.

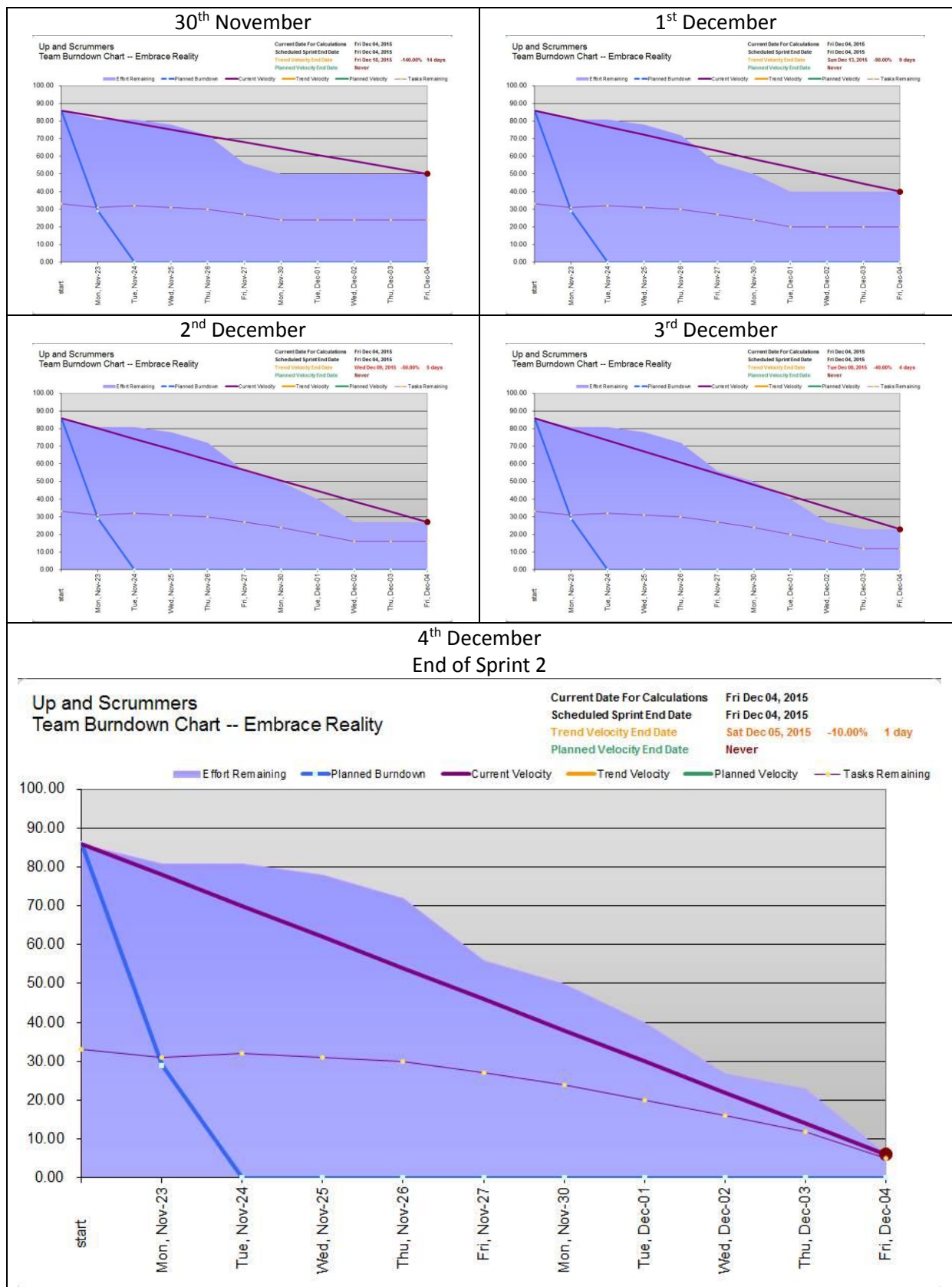
Sprint 1 Burndown Snapshots



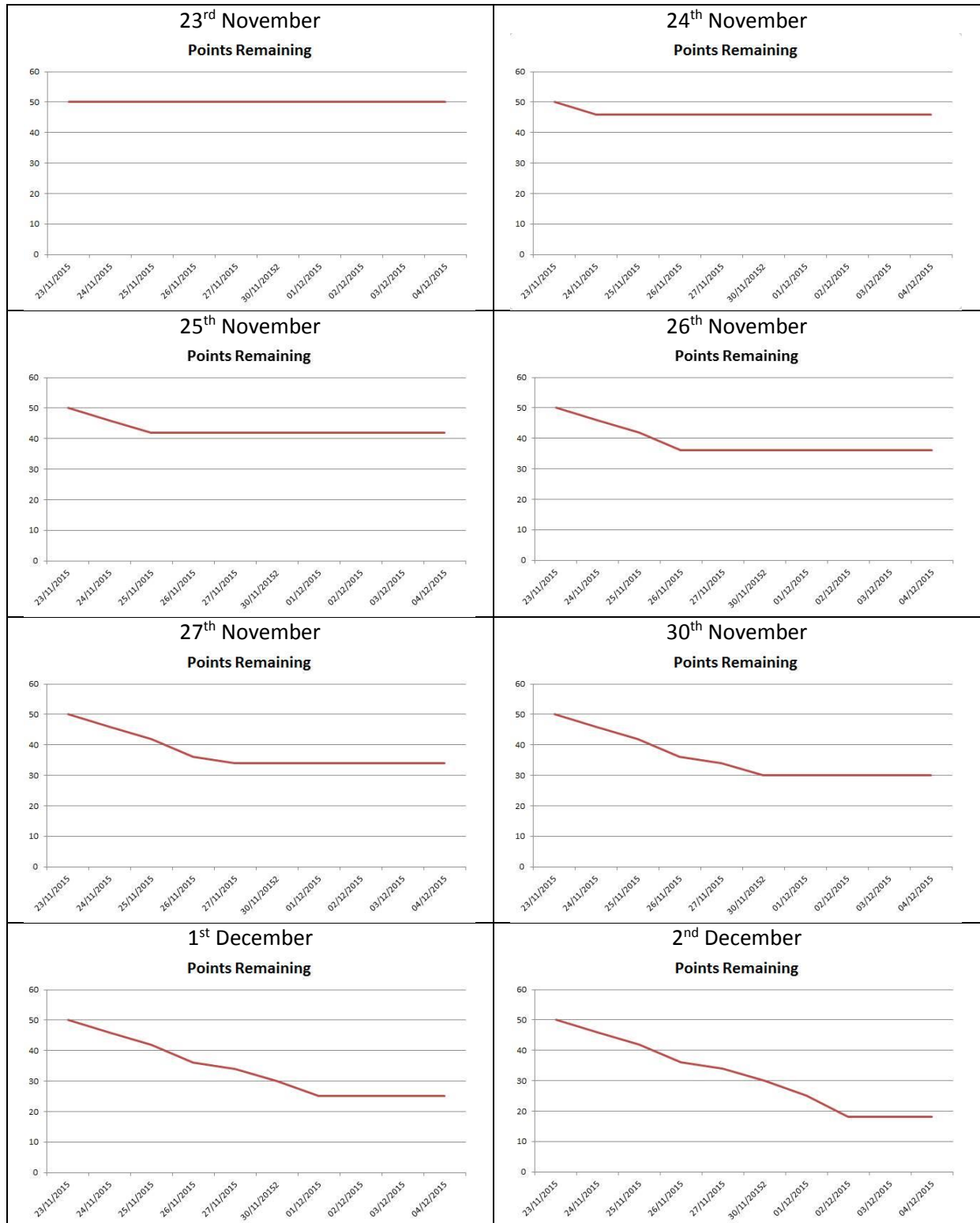


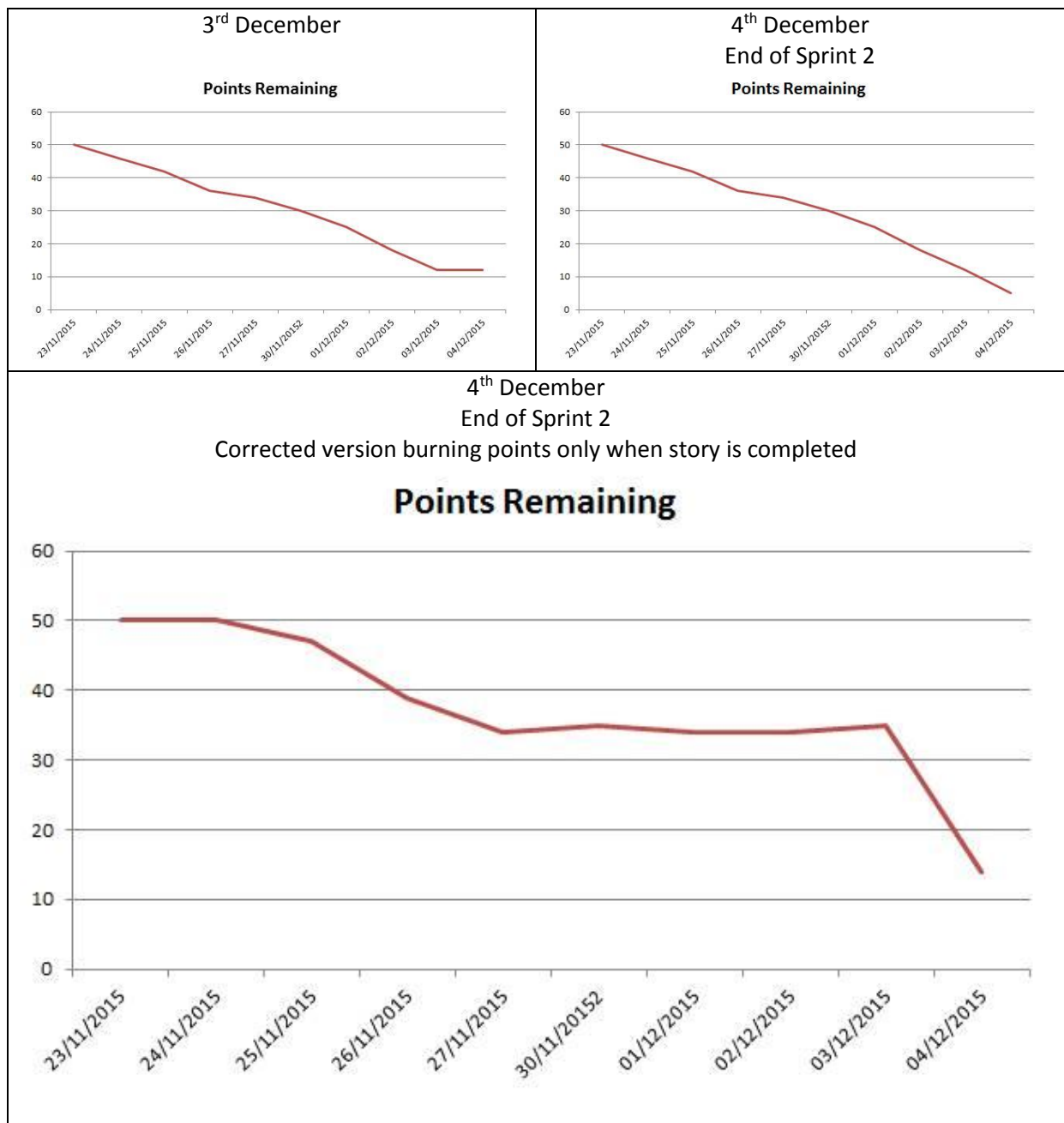
Sprint 2 Burndown Snapshots





Story Point Burndown





Sprint Reviews

As a team at the end of both sprints we met and discussed the aspects of the sprint we were happy with and where we could improve in the next one. At the end of our first sprint we were disappointed by the output in only having three stories completed and sought to find out what caused us to fall short of our expectations. Aside from late changes to the project, bugs and SVN issues we picked out poor planning as a key factor in our low delivery and took steps to improve this as mentioned in previous sections with regards to re-doing our planning poker and better management of our Hartman Orona spreadsheet. We did recognise that the contribution and communication between team members was a strong point of the group as well as being able to pair program and provide assistance very efficiently. Our own conclusions were in line with the discussions in our sprint retrospective and we used that feedback to improve the performance of our team and also to make requested changes to the project itself.

Taking on board the lessons learned from sprint 1 we were able to perform a lot better throughout the second sprint which saw us work more efficiently and allowed us to complete more user stories. We initiated more communication with the product owner to ensure that the features being implemented would be in line with their requirements and avoid assumptions leading to changes needing to be made.

Were we to conduct a third sprint I feel the team would perform better once again as the experience during sprint 2 could be applied to further refine our planning, organisation and also our coding. Taking into consideration that the team planned our work based on having six members and got almost all of this completed and fully implemented with other sections partially working with five members we would adjust our goals based on a team of five. By continuing to communicate and collaborate efficiently we are confident that our team would have a highly successful third sprint.

Project Design Specification (UML)

Class Diagrams

Class diagram format:

Class Name
<i>Variable Name : Data Type</i>

Base Classes (CSC3045AgileDataModels)

User
Id : int Forename : string Surname : string Email : string Password : string Skills : string

Role
Id : int Title : string Selectable : bool

Project
Id : int Title : string

UserStory
Id : int ProjectId : int Title : string Description : string

Sprint
Id : int SprintTitle : string Duration : int InProgress : bool Blocked : bool

Task
Id : int Title : string InProgress : bool Blocked : bool TaskType : string Duration : int TaskDetails : string

SprintTeam
id : int TeamName : string isAssigned : bool

ClientServerModels (CSC3045AgileDataModels)

UserServer : User
RoleIds : List<int>

UserClient : User
Roles : List<RoleClient>

ProjectServer : Project
ManagerId : int OwnerId : int ScrumMasterIds : List<int>

ProjectClient : Project
Manager : UserClient Owner : UserClient ScrumMasters : List<UserClient>

SprintClient : Sprint
Project : ProjectClient ScrumMaster : UserClient TaskList : List<TaskClient> AssignedTeam : SprintTeamClient

SprintServer : Sprint
ProjectID : int ScrumMasterID : int TaskIDs : List<int> DeveloperIds : List<int>

TaskClient : Task
Project : ProjectClient Sprint : SprintClient AssignedUser : UserClient

TaskServer : Task
ProjectID : int SprintID : int UserID : int

SprintTeamClient : SprintTeam
DeveloperList : List<UserClient>

SprintTeamServer : SprintTeam
DeveloperIDs : List<int>

Entities (CSC3045AgileDataModels)

UserEntity : User
Roles : List<RoleEntity>

RoleEntity : Role
Users : List<UserEntity>

ProjectEntity : Project
Manager : UserEntity Owner : UserEntity ScrumMasters : List<UserEntity>

ProjectScrumMaster
Id : int ProjectId : int? Project : ProjectEntity UserId : int? User : UserEntity

SprintEntity : Sprint
Project : ProjectEntity ScrumMaster : UserEntity TaskList : List<TaskEntity> Developers : List<UserEntity>

TaskEntity : Task
Project : ProjectEntity Sprint : SprintEntity AssignedUser : UserEntity

Entity Relationships

With reference to the ScrumMasterUserEntities AgileContext class located in the CSC3045AgileServerSide location, the ProjectEntity_ScrumMasterUserEntities and SprintEntity_DevelopersUserEntities relationship tables are created in the AgileDB database.

For the ProjectEntity_ScrumMasterUserEntities table the program maps the ProjectId property of the ProjectEntity to the UserId property of ScrumMasters. The table will have a many to many relationship meaning that many ProjectId's can be mapped to many UserId's.

ProjectEntity_ScrumMasterUserEntities
ProjectId UserId

The SprintEntity_DevelopersUserEntities table is created by mapping the SprintId property of the SprintEntity to the UserId property of Developers. The table will have a many to many relationship meaning that many SprintId's can be mapped to many UserId's.

SprintEntity_DevelopersUserEntities
SprintId UserId

Code Quality Analysis

Over the course of the project the team used a series of features and techniques available in the .NET Framework in order to deliver our working software. These include:

- Inheritance
- Generics
- Delegates/Lambda expressions
- Event handling
- Exception handling – try-catch-finally
- Asynchronous programming – async and await
- Using statements – disposing objects
- Entity Framework
- LINQ to Entities/Objects
- Json.NET – 3rd party JSON framework
- WPF – client
- Web API – server

The use of these features displays an understanding from the basics right through to some of the more complex topics found in .NET and makes for, what we feel is, an overall good standard of coding and coding practices.

The foundation our software is built on is the database that drives it. As a team, we opted to take the approach of modelling our database in code using Entity Framework Code First Migrations – with some table relationships built using the Fluent API. In doing so, we designed a number of classes – which would represent tables – that could be used throughout the project to transfer and manipulate data. The classes we came up with, however, would need to be used project wide and so it was not always the case that an entity model would be appropriate. It was agreed that the Client, Server and Entity use-cases were slightly different from one another and as such would require unique data representations. Inheritance came in to play when we created abstract base classes with all the shared properties required allowing us to extend these classes as needed and, where appropriate, add properties that a particular representation required also. As an example:

- A client model may need all of a user's properties – to display various details in the UI.
- A server model needs only a user's ID – in order to retrieve that user and operate on the data.
- An entity model may use only a user's ID to form a relation to another table, but might also use a virtual object to retrieve the user's data from the related class/table.

This use of classes allowed us to build our multiple data representations and database tables all from the same foundation with very little code duplication.

This project used asynchronous programming techniques in order to avoid hang ups of the user interface caused by potential delays in business logic and/or during server request/responses. It was important that while the code behind our UI was operating that we don't give the user the impression that the software is unresponsive. Methods are prefixed with the async modifier and we await asynchronous method calls from other classes. There is also use of the .NET Task Factory which allows us to use Json.NET asynchronously as its async methods have been depreciated. This

coupled with extensive exception handling means that the user can be made aware of any issues, via a message area found in the UI, should they occur on either the client or server side.

When querying or manipulating our database or a collection of objects we leaned toward an extensive use of lambda expressions where possible. This useful shorthand provides a nice means of handling data where we might otherwise rely on more verbose techniques such as traditional for loops with logical operators inside. These lambda expressions are used in conjunction with a multitude of extension methods provided by the LINQ library and are passed to these methods in the form of delegate methods – as a side, we also made use of delegate methods when calling customised event handlers on Combo Boxes. This way of handling data made for a succinct and often more pleasing implantation than is otherwise achievable with other techniques.

Throughout the project on both client and server sides we created helper classes in order to cut down on duplicate code. These classes allowed the team to call methods – which quite often made use of generic type parameters – which handle more complex code snippets and allowed them to be reused often. An example of this would be serializing data, making server request, awaiting the response and de-serializing the response data. This method would accept generic type parameters telling it which object types it must send and receive.

The projects front end, built on Windows Presentation Foundation, gave the team an opportunity to work with its robust XAML mark-up language and deal with user interactions via event handling. The UI was put together using a number of the XAML elements available such as Grids, Stack Panels etc. We decided to use a main window which housed common UI elements that were common among the entire application whilst loading individual pages in to the main window as navigation dictated. The main window was backed by a class which allowed each subpage to call and edit individual components – such as updating a page title or displaying an output message. The UI itself was for the most part given percentage (star) size values in order to make it scalable/responsive. In many cases UI components were added dynamically via the code behind based on data gained from the server.

Our server side took the form of an ASP.NET Web API project which replaced our original Windows Communication Foundation implementation. This simple server side framework makes it easy to add and maintain endpoints for our client to call out to via HTTP and provides a great way to make straightforward RESTful services. Web API dramatically cut the amount of code required to implement each endpoint and ultimately comes down to writing a traditional method and providing it with an HTTP method type and URL – rather than the contracts required in WCF. Implicit model binding via JSON was yet another bonus that made the decision to use this framework the right call.

Perhaps an aspect of our code which could have been improved upon was the inconsistent use of commenting, mainly in the form of XML methods comments. Other code implementations previously mentioned may suffer from slight inconsistencies in terms of usage but overall we are happy with the range of techniques we used and how working code operated.

Convention to Coding Standards Document

The coding standards document is attached in the appendix of this document. It outlines the conventions which should have been followed in the development of this project.

For the most part these conventions were followed. The first section of the coding standards document outlines the convention for the code layout. The majority of this section was followed exactly as stated with the exception of the spacing. The document states that there should be one blank space between each of the top level classes and functions as well as between all if, for and while statements. An example of the convention for spacing not being followed is shown below, in the SprintIndex.xaml.cs page. Shown in the code sample below two spaces have been left between two methods.

```
private static List<SprintClient> sprintList { get; set; }
public SprintIndex()
{
    InitializeComponent();
    MainWindow.HFContent.UpdateTitle("Projects You Are ScrumMaster Of ");
}

private void Page_Loaded(object sender, RoutedEventArgs e) ...
private Border GetIndexedComponent(ProjectClient project) ...
```

The next section of the coding convention outlines the naming conventions which were to be followed. The first few of these were always followed such as giving variables meaningful names and not naming variables with single letters. Underscores were also not used in the naming of variables. One particular exception from this is in the SprintCreate.xaml.cs page where a temporary checkbox is named tempCheckBox.

The naming conventions to be used continue to including the type of casing which should be used for things such as folder names, class names and method names. This was the main section within the coding convention which was not followed as closely. The first part of this stated that all folder names should use Pascal casing which was followed an example of a few folder names are ProductOwner and ProjectScrumMaster. The second part of this was for the class names which were again to be Pascal casing and again this was followed closely. A few examples of this are shown below

```
public partial class OwnedCreate : Page
{
    private ProjectClient project;
}

public partial class ManagedCreate : Page
{
    private ProjectClient projectToEdit;
    private bool editMode = false;
    private List<int> scrumMastersSelected;
```

However the convention was not followed as well for the naming of methods. The coding convention states that all method names should use pascal casing. A sample piece of the code for the system is shown below. This code block includes two method names, one of which follows the convention and one of which does not. These are both in the ManagedCreate.xaml.cs page.

```
public ManagedCreate(ProjectClient projectClient)
{
    InitializeComponent();
    projectToEdit = projectClient;
    editMode = true;
    scrumMastersSelected = new List<int>();
    MainWindow.HFContent.UpdateTitle(projectToEdit.Title);
    titleTextBox.Text = projectToEdit.Title;
    ownerSearchTextBox.IsEnabled = false;
    PopulateComboBox(new List<UserClient> { projectToEdit.Owner });
    ownerComboBox.IsEnabled = false;
    PopulateScrumMasters(projectToEdit.ScrumMasters);
    scrumMasterSearchTextBox.IsEnabled = false;
}

private async void ownerSearchTextBox_KeyUp(object sender, KeyEventArgs e)
{
    if (e.Key == Key.Enter && !string.IsNullOrEmpty(ownerSearchTextBox.Text))
    {

```

The coding standard also states that variable names should use camel casing. This again was not followed perfectly. The code above is an example of a time where the convention was followed however the code below is from the SprintCreate.xaml.cs page and shows variables named using pascal casing and not camel casing.

The final section of the coding standards document outlines good coding standards which were to be kept to in the development of this system. Again the most of these conventions were followed with the major exception of commenting. The coding standards document states that if the function of a piece of code is not clear there should be a comment briefly explaining its purpose however there is very little commenting included in the code for this system.

Overall considering that the coding standards were not in place from the start of the first sprint the conventions were followed pretty closely with the main exceptions of commenting in the code and the naming of methods and variables.

Testing Documentation

Unfortunately, we as a team were unable to implement any unit testing, the difficulty being that we were unable to mock the database. There did exist a local implementation of the server solution which included a testing project that successfully called the Web API endpoints but since we could not inject a mock database the unit tests returned invalid queries – usually empty result sets – meaning the tests failed.

During the course of the project we did, however, conduct lots of manual testing using a few different techniques. For GET requests we could simply navigate to an endpoint with the correct parameters via the web browser and would then inspect the loaded JSON data response. We also made use of a Google Chrome extension called Advanced Rest Client Application. This app allows the user to insert a URL and then select the appropriate HTTP method type as well as set the appropriate HTTP headers on the request – such as Accept content-type etc. The app then displays the response sent from the server and in the case of a successful test, the corresponding data.

Example GET request using Advanced Rest Client Application.

http://localhost:22053

/api/Roles/

Query parameters

ADD

name

value

X

ENC

DEC

History hash

HASH

GET

POST

PUT

PATCH

DELETE

HEAD

OPTIONS

Other

Raw

Form

Headers

Clear

Send

SCROLL TO TOP

Status

200 OK Loading time: 59 ms

Request headers

User-Agent: Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/47.0.2526.80 Safari/537.36
Content-Type: text/plain; charset=utf-8
Accept: */*
Accept-Encoding: gzip, deflate, sdch
Accept-Language: en-US,en;q=0.8,en-GB;q=0.6

Response headers

Cache-Control: no-cache
Pragma: no-cache
Content-Length: 238
Content-Type: application/json; charset=utf-8
Expires: -1
Server: Microsoft-IIS/10.0
X-AspNet-Version: 4.0.30319
X-SourceFiles: #?UTF-8?B?
QzpcVXNlcnNcd2lsbGlicRG9jdW1lbmRzcFFvQlwyMDE1LTE2XENTQzMwNDUgLSBBZ2lsZVxQcm9qZWNOXENTQzMwNDVBZ2lsZVNlcnZldNpZGVcQ1NDMzMwNDUwFnaWxlU2VydmVjYUJkZVxhcGlUM9sZXNlc?
=
X-Powered-By: ASP.NET
Date: Wed, 16 Dec 2015 00:32:36 GMT

Raw

JSON

Response

COPY TO CLIPBOARD SAVE AS FILE

[3]
-0: {
 "Id": 1
 "title": "Product Owner"
 "Selectable": true
}
-1: {
 "Id": 3
 "title": "Scrum Master"
 "Selectable": true
}
-2: {
 "Id": 4
 "title": "Developer"
 "Selectable": true
}
}

Team Work Attribution

Gareth

Sprint 1

- Implementation of Front-End WPF Pages and setup initial system navigation structure.
- Created basic data models for first implementation of client-server communication.
- Paired programming with Kyle to create API Route methods for creating, getting and editing Project, Sprint, Task and User objects.
- Skeleton code for UI form object implementation.
- Worked with Kyle/Josh on Login Page
- Bug Fixes and Merge Conflict fixes

Sprint 2

- Began full Implementation of UI form objects for Task/Sprint/User Management until UI overhaul was decided upon.
- Implementation of new Front-End UI System (Displaying existing objects and creating)
- Implementation of UI helpers for managing data objects retrieved from server.
- Implementation of more server side API Route Methods for Task/Sprint/User/Project. Updated and Created new server API Routes and Methods for Post, Get and Put. Updated existing skeleton methods to match with final implementation of objects.
- Implementation of server helpers for converting returned records to send to the client.
- Updates to Data Models, Created new Entities and Base Models with Server/Client specific models.
- Implementation of Sprint Creation and Management to new UI, Paired Programming with Maeve.
- Manual server to client (and vice versa) route testing.
- More Bug Fixes and Merge Conflict fixes

Josh

Sprint 1

- Implementation of Front-End WPF Pages and navigation structure
- Paired programming for basic data models for client-server communication
- Paired programming on initial Login/Registration functionality
- Skeleton code for UI form object implementation
- Worked with Gareth/Kyle on Login/Registration
- Bug fixes

Sprint 2

- Implementation of Task Creation to new UI
- Updated server side API methods for Task Creation
- Updated Data Models for Task Creation
- Paired programming with Maeve on user controllers
- Manual server to client (and vice versa) route testing)
- Bug Fixes

Kyle

Sprint 1

- Initial Server Setup for Queens DB Hosting and Own DB Hosting
- Built our initial Server Side WCF implementation (prior to switch to Web API). Created API Methods and Routes for Client-Server interaction.
- Built out Server to Database communication for storing and retrieving records.
- Created initial basic Data Models
- Worked on Login/Registration page, completed final implementation of both.
- Bug fixes and Merge Conflicts
- Switched server to Web API

Sprint 2

- Implementation of more server side API Route Methods for Task/Sprint/User/Project. Updated and Created new server API Routes and Methods for Post, Get and Put. Updated existing skeleton methods to match with final implementation of objects.
- Expanded Data Models, Built out Entities and Basic Models with Client/Server specific objects.
- Implemented new Client and Server Helpers for Handling and Converting objects for the Server/Client to handle.
- Implemented new UI front-end overhaul, Implemented functionality to track logged in user and permissions.
- Implementation of Project Creation and Management, Tracking functionality implemented into new UI.
- Paired Programming on Task and Sprint creation/management to resolve errors and bugs.
- Manual UI/Server Testing
- Bug Fixes and Fixes of Merge Conflicts

Maeve

Sprint 1

- Worked on Login/Registration UI/Server Implementation.
- Updates to API Route Methods
- Updates to Data Model Objects
- Initial Development of User Controllers
- Initial Database Setup
- Bug Fixes and Merge Conflicts

Sprint 2

- Paired Programming with Josh on user controllers
- Worked on Initial implementation of new UI Overhaul with controller support.
- Paired Programming with Gareth on Sprint Creation and Management
- Updates to new Data Models
- Updates to API Methods for Put, Post and Get routes for Sprint.
- Manual UI Testing
- Maintenance of code to keep in line with coding standards document
- Code Tidy Up and Bug Fixes

Ronan

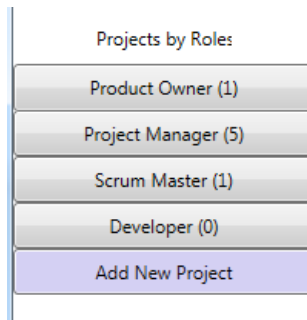
Sprint 1

- Implementation of Front-End WPF Pages and navigation structure
- Initial Database Design and Setup
- Updates to API Route Methods
- Initial Role Search Method
- Updates to Data Model Objects
- Bug Fixes and Merge Conflicts

Sprint 2

- Implementation of new API Route methods for Role Searching
- Implementation of Story and Task creation/management in new UI.
- Updates to API Route Methods for creating and managing Stories and Tasks
- Updates to Data Models for Stories and Tasks.
- Manual UI Testing
- Created Story Point Burndown
- Navigation Fixes
- Bug Fixes

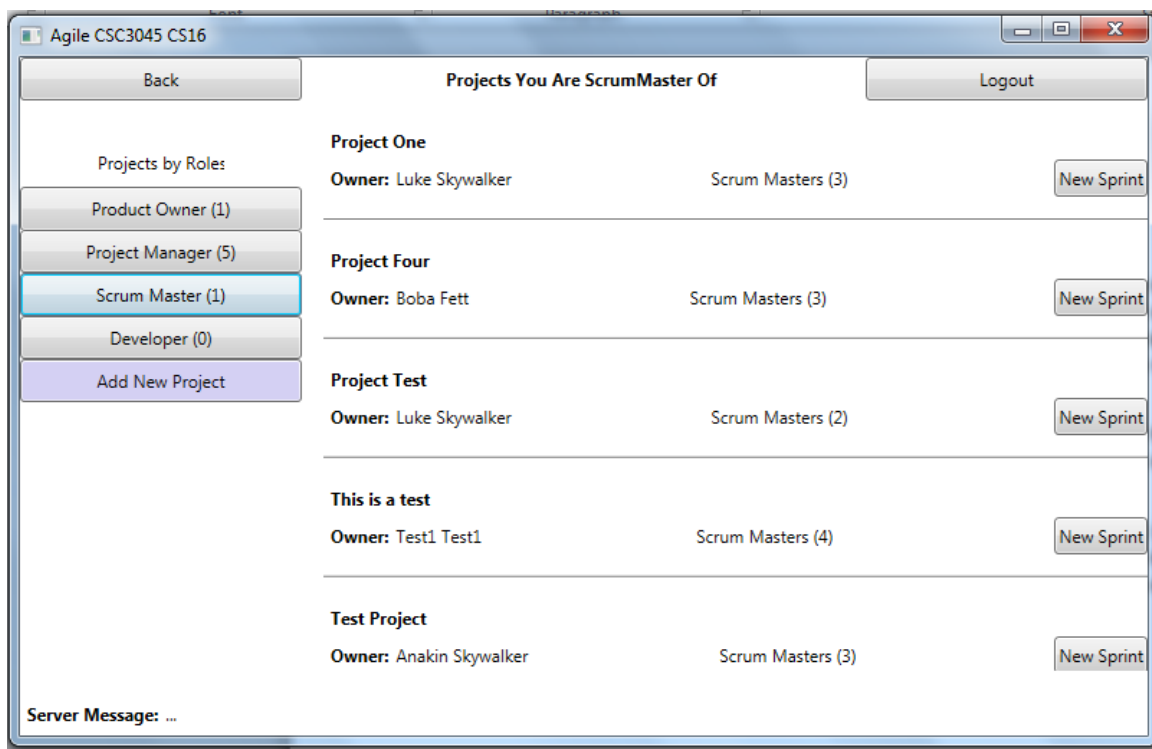
Known Bugs list



There are three known bugs with the current system. The first of these is that in the section of the system where the user should be able to view the projects which they are a scrum master for. In order to view this page the user should click on the 'Scrum Master' section in the menu bar as seen to the left.

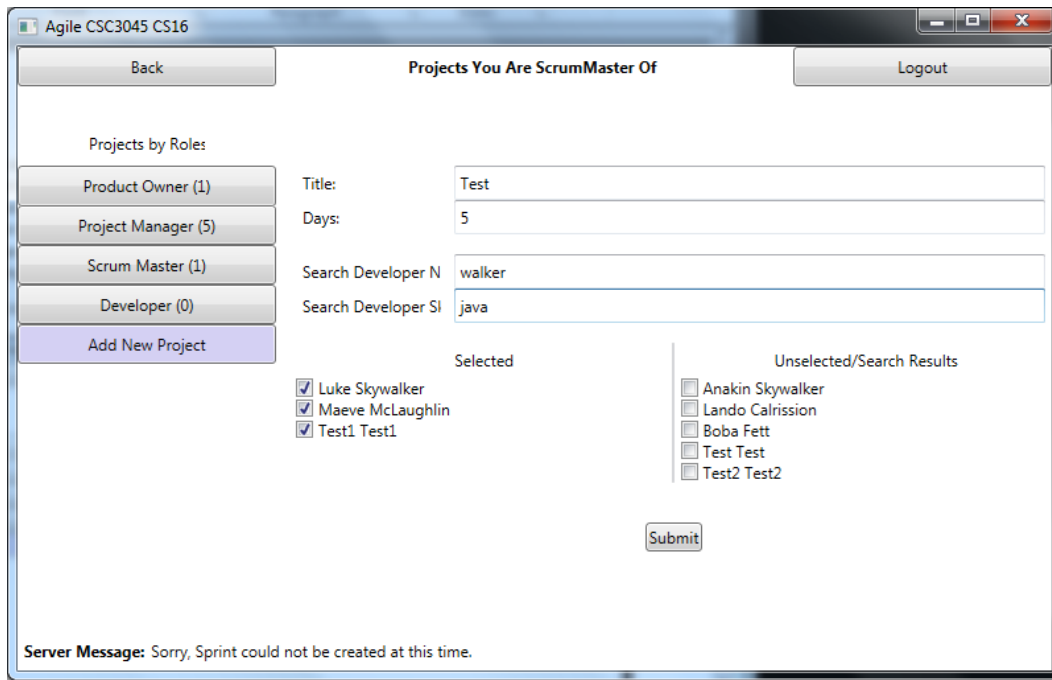
This should then display a list of the projects which they are currently a scrum master in and allow them to create new sprints with user stories, and create tasks from these user stories.

There is however a slight bug in the page in that it loads any projects which the user is a Project Manager in and not a scrum master as seen below. Although the projects which are loaded when the page loads are not the correct project the rest of the page still functions the same in that it allows the user to create a new sprint.



The second of the known bugs in the system is that creating a new sprint page has not been properly coded in that the new sprint page loads and allows the user to input all the data required for creating a new sprint, including searching for developers both by name and by skill however the code behind the submit button below for actually creating the sprint in the database does not work. When the code is run it breaks and an error message is shown at the bottom of the page showing 'Sorry sprint could not be created at this time'.

This is purely because the group ran out of time in which to properly test and write the database call to create a new sprint.



Agile CSC3045 CS16

Back Projects You Are ScrumMaster Of Logout

Projects by Roles:

- Product Owner (1)
- Project Manager (5)
- Scrum Master (1)
- Developer (0)
- Add New Project

Title: Test

Days: 5

Search Developer N: walker

Search Developer SI: java

Selected:

- ☒ Luke Skywalker
- ☒ Maeve McLaughlin
- ☒ Test1 Test1

Unselected/Search Results:

- ☐ Anakin Skywalker
- ☐ Lando Calrission
- ☐ Boba Fett
- ☐ Test Test
- ☐ Test2 Test2

Submit

Server Message: Sorry, Sprint could not be created at this time.

The third and last known sprint bug is that the 'Developer' option on the menu does not have working code behind it so when that option is selected on the menu nothing will happen.

Critical Reflections

As Critical Thinking is a significant and vital part of continual improvement in Agile methodologies, an analysis of our performance during our second sprint and the state of the deliverable system will be a key part in planning a third Sprint and identifying areas of improvement. This section will cover areas where the team performed well as well as areas where there was significant waste and issues.

One of the major successes of our second sprint was the competency of each member of the team in their role as Scrum Master as well as the effectiveness of the communication between all members of the team. On any given day it was clear to the Scrum Master and to the members of the team what the outlined work for the day was, what was expected and how the implementation would be handled. Each team member knew what the other was working on and how it would affect their own work. We were quick to adapt to any issues that arose during prototyping and we were able to adjust our solution to fit any changes that were required. As we each took turns acting as Scrum Masters we were all able to gain additional experience managing the flow of development and we steadily improved at making quick critical decisions to keep the project moving. Moving into a third sprint, we feel that we would be able to make significant progress earlier in the sprint and to be able to reach major milestones earlier.

Another success of our second sprint was our ability to rapidly implement, prototype and test new features. This allowed us to identify areas of improvement, potential issues and breaking changes that we could quickly resolve and react to. During our second sprint where we did not attempt to rapidly prototype and test early we commonly ran into unforeseen issues. Moving into a third sprint we would make significant effort to develop a prototype and test as early as possible, especially if the functionality being implemented was to be reused in other areas of the code (For example in the Helper methods present in the server code).

Waste was a significant issue during our second sprint. Team members spent time working on implementation of features that in the end were entirely cut due to time constraints or due to unforeseen circumstances. For example, we had planned to include a user management system in the second sprint and time had been spent working on this functionality. In the end it was cut due to time constraints with it being completely omitted from the deliverable system. Moving into a third sprint in order to avoid significant waste such as this happening again we feel that better planning would help us manage the workload. We overestimated the amount of work we would be able to achieve and the prioritization of the work could have been better, for a third sprint I believe it would serve us well to correctly prioritize features and underestimate the amount of work to be completed. If there is time towards the end of the sprint to work on additional functionality, then we would add it mid-sprint.

Another issue that arose during our second sprint (as well as the first) was wasted time due to conflict errors. Team members had to spend a significant amount of time on some days fixing broken builds due to conflict errors. We are not sure exactly what caused most of these conflict errors as on numerous occasions (and during testing) we discovered that there were conflicted files being uploaded despite not being included in the commit. This may have been due to our teams inexperience with SVN or other technical issues. In a third sprint we would strive to isolate areas of work so that team members are never working on the same blocks of code at the same time and to tie stronger ownership of features to developers. Another possible improvement is to move our code repository from SVN to Git, all of our team members are experienced with Git and we feel this may cut down on the number of issues we would encounter.

Finally, a significant issue that arose during our second sprint necessitated us to entirely switch our implementation for both UI and Server. We encountered repeated issues that resulted in a lot of time being wasted on bug fixes and resolving recurring errors, despite our best efforts to avoid these happening. Making the UI and Server overhauls resolved the issues we were having and perhaps the decision should have been made sooner. Moving into a third sprint we feel it would be very beneficial to explore our options for implementing new code or features more thoroughly before the sprint begins and whenever issues are encountered, make critical decisions earlier to save time and effort that was previously lost attempting to fix recurring issues.

To summarize some of the key areas of improvement for Sprint 3:

- Faster Prototyping and Testing when implementing new features, particularly if code is a base for other features and highly re-usable.
- Plan and Prioritize implementation of specific features, plan to get less accomplished and focus on getting critical sections of work done first before moving on to additional features.
- Reach major milestones earlier, focus the team on reaching these milestones and make sure the code is sound.
- Take more control and ownership of critical features to cut down on the number of conflicts and breaking changes that are encountered. Will hopefully cut down on the amount of team work devoted to resolving these.
- Switch from an SVN Repository to Git. We encountered far too many issues with SVN, All members are experienced with Git and we can easily branch work and manage merges a lot easier.
- Explore implementation options prior to the start of the sprint, have backup plans incase original decisions cause too many issues or are unworkable.
- Make critical decisions earlier in the sprint, Isolate and resolve areas of repeating issues.

Appendix

Code/Reference material

Entity Framework Tutorial - <http://www.entityframeworktutorial.net/>

WCF Restful Service Tutorial - <http://www.topwcfutorials.net/2013/09/simple-steps-for-restful-service.html>

Coding Standards Document

This document will outline the coding standards which should be followed for the product of this project.

Any text written in red within this document will indicate code

Code Layout

Indentation

Always indent one tab space per indentation level, do not use spaces. We are using visual studio and its standard code format should be followed.

Spacing

There should be one blank space before, after or between each of the top level functions and classes.

There should also be one blank space before and after any if, while and for code blocks.

Library References

All library references used should be at the top of each page.

Each page should only contain references which are needed within that page to avoid unnecessary code.

Naming Conventions

All variables, methods etc. that are created must be named with meaningful descriptive names, for example when creating the variable for a user's forename

use

`string forename`

not

`string variable1`

Single letters should not be used as variable names with the exception of using variables for loop iteration

`for (int i = 0, i < count, i++)`

```
{  
}
```

Underscores should not be used in the naming of variables but may be used in method names for things such as button clicks e.g.

```
private void button_Click(object sender, RoutedEventArgs e)  
  
{  
}
```

There are two naming conventions which will be followed. These are Pascal Casing and Camel casing.

Pascal Casing – The first character of each word with the name are uppercase, all other characters are lowercase e.g. HelloWorld

Camel Casing – The first character of each word should be uppercase with the exception of the first word, all other characters are lowercase e.g. helloWorld

Folder Names

All folder names should use Pascal Casing

e.g. UserManagement

Class Names

All class names should use Pascal Casing

```
public class HelloWorld  
  
{  
}
```

Method Names

All method names should use cascal casing

```
private void button_Click()  
  
{  
}
```

Variable Names

All variable names should use camel casing

e.g. helloWorld

Good Coding Standards

1. Do not hardcode a path in code, get the application path programmatically and use the relative path
2. Methods should not be too large, no method should be long than 50 lines of code.
3. One code files should not be too large. Any files over 1000 lines should be spilt into two classes if possible.
4. More than one class should not be included in the one code file
5. Code files should be organised into folders which should be logically organised by the user roles which access them.
6. Any code which has the possibility of erroring out should be contained within a try catch which should handle the error appropriately
7. If the function of a block of code is not clear there should be a comment briefly explaining it.