IGOS 2.0

Final report

Spring 2013

Gísli Rúnar Guðmundsson

Sigtryggur Óskar Hrafnkelsson

Stefán Brynjólfsson

Þórarinn Magnússon

BSc in Computer Science

Leiðbeinandi:Stefán Freyr Stefánsson T-404-LOKA

Prófdómari: Birgir Kristmannsson Tölvunarfræðideild

Table of Contents

1. Introduction

The project, given by Aegos, was to redesign, rewrite and gamify an in-house cafeteria app for iPad in Unity3D, integrate it with other information systems within the company and make it more fun, visual and independent as a product. Due to separation within Aegos, the project owners split from the company and formed their own named “Nutty Nerds”. The separation ultimately gave us more freedom in designing the project.

The main focus of the project was to provide a fun experience for users when purchasing lunches, snacks and drinks within their company.

2. Vision statement

iGos 2.0 is a self-service in-house app for company cafeterias. iGos seeks to give its users a fun experience by having game elements from role-playing games to it, as well as providing an easy to use inventory and accounting systems. A specific admin interface was not implemented for the back-end and was the focus of the project rather on the user experience of the normal user.

The app is to be able to hold its ground as a simple store managing app but is to be “fun-to-play” in order to make the dull activity of purchasing food a fun experience. Therefore, gamification ideology was integrated into the project.

3. iGos 2.0 and Gamification

"*Gamification is the process of using game thinking and game dynamics to engage audiences and solve problems.*"[1] - Gabe Zimmerman

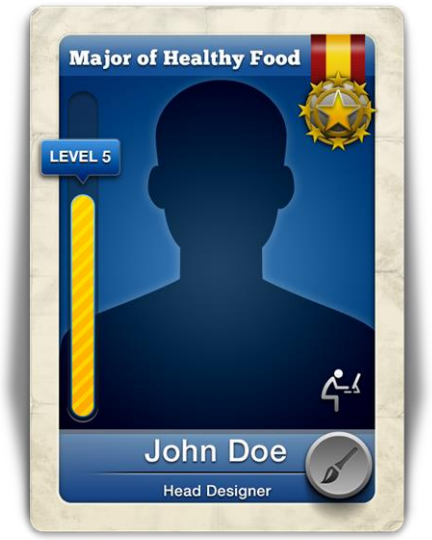
3.1 General gamification

Since the most important goal of the project was to create a fun experience for the user, it was decided to use gamification. In gamified systems users are often rewarded for their actions. iGos 2.0 will use classic game mechanics from roleplaying games. Those game mechanics are player classes, experience points, quests and achievements. Users will receive rewards that can be displayed on their avatar.

3.2 Avatar

A user's avatar will have a trading card look to it, similar to the picture here on the right, consisting of;

- a border
- an achieved title
- an achievement badge
- experience bar
- level of character
- personal 2D picture (even 3D avatar in the future)
- work status (e.g. working from home)
- name of user
- user's workplace title



Each user can choose different skin types for the border, badge and working status of their avatar. The working status in particular will help co-workers to see if the specific user is at work or not.

3.3 Classes

There are two classes in iGos 2.0, the Health food class and the Soul food class. Each user belongs to both of them but the user chooses which one he would like to display on his avatar. He can select which class is currently shown through the profile menu and can swap between them whenever he wants to. The users gain a special title according to which class they chose and the level they have reached in the class.

The health food class is aimed at people that like to eat healthy whereas the soul food class aims at people that like to eat junk food and candy.

[1] "Zichermann: Gamification - game thinking to solve problems." *Vimeo*. N.p., 20 July 2012. Web. 16 May 2013. <vimeo.com/46092961> .

3.4 Achievements

Each purchased item will add up to achievements related to that item, e.g. 100 cola in a single month will reward you with the “Cola King” achievement. Earning achievements will grant the user rewards that can be displayed on their avatar, for example different borders or special badges as well as bragging rights within his/her company.

3.5 Quests

The app also offers users to go on quests (e.g. “buy 10 bags of nuts this month”) that offer various rewards such as extra experience or items to display on your avatar. When a quest is available for an item it will have a visual backlight glow in the store and by holding down the item-button the user can read about the particular quest in the “more info” popup window.

3.6 Experience

Each item in the store is tied to a specific class. When an item is bought, the system gives the user a certain amount of experience points (different for each item) in the class related to that item. The user always gains experience points for a purchase, even though it’s in a class that’s not currently being displayed on the user‘s avatar.

Quests and achievements can also grant the user experience points in a certain class.

4. Preparation

4.1 Scrum

We decided on using scrum as our development framework where the project would be split into several iterations for a total of eight. The first five iterations were two weeks long and the final three were one week each. We wanted the earlier sprints to be longer since during that time we had other classes. Another reason is that early designing would take more work due to less knowledge. This plan left us with four days at the end which would be spent on writing the final reports and polishing the code.

When working on the project we had daily scrum meetings, however they sometimes needed to be conducted online because of other commitments. We reviewed each sprint after it was done, looking over what went well and what could have gone better. After that we planned the next sprint.

A backlog of user stories was set up on Google drive where we first set up several stories and then used planning poker to assign story points to them with a point range of 1,2,3,5,8,13, 20 and 40.

For each iteration we pulled stories from the product backlog and added new as they arose.

Iterations

1. 11.february - 24.february
2. 25.february - 10.march
3. 11.march - 24.march
4. 25.march - 7.april
5. 8.april - 21.april
6. 22.april - 28.april
7. 29.april - 5.may
8. 6.may - 12.may

4.2 Design

The design of iGos 2.0 is based on the older version, iGos 1.0. The mockups for the new system were made in Balsamiq Mockups and sent to a designer, provided by Nutty Nerds, who created the final design of the product.

4.3 Coding Convention

We discussed coding conventions and decided on rules that we all agreed to use throughout the project. (see “iGos Coding Standards”)

5. Implementation

5.1 Coding Environment

The application is written in C# using the Unity3D engine, referred to from here on out simply as Unity. We used Visual Studio 2010 as our main IDE.

For our test-driven development we purchased the Test Star plugin for Unity which is available through the Unity Asset Store.

By purchasing a student version of the Unity Pro license we got access to a version control built into Unity which allowed us to push in changes, update local code and revert to earlier versions. This service was hosted on a remote Amazon server.

5.2 Separation

The application is separated into a logic and a UI layer. The UI layer deals with creating the interface and detecting user input. The logic layer receives the user input and does the appropriate calculations. The application also has a network layer that the logic layer uses to send and receive data from a database. The network layer sends JSON requests to a server and uses the LitJSON plugin for Unity to serialize data.

A service layer which interacts with a database was written in C# using Visual Studio 2012 and is hosted on an Amazon server. That layer is based on the Entity framework which is an object-relational mapper; it uses models to create tables in a database.

5.3 Controllers

Several controllers exist in the system that retrieve and store data from the database using the network controller. When the application first runs every controller retrieves data from the database and stores it in lists. One example is the Item Controller which is used for data relating to the items in the store. It stores all the store items and purchases made in lists. The controller has several functions to retrieve data from those lists, for example one that retrieves all purchases that a specific user has made, using Linq to query the lists. The controllers therefore act for the most part as data manipulators for the logic layer.

5.4 Gamification

An achievement and quest system was written based on the observer pattern. Achievements and quests need to check if certain items have been purchased, all items therefore have a list of achievements and a list of quests which may be empty. Controllers handle inserting achievements and quests into the lists of the items that they are related to. When an item is purchased it notifies all the achievements and quests in its lists of the purchase. One quest in the system requires the user to purchase 10 Coca Cola soda cans at once. Once a Coke is purchased, the quest is notified of the purchase and it polls the ItemController for the user's purchases. It then goes through them until it finds a purchase tied to the identifier of the Coke item and with a quantity of 10 or more. If so, it uses the network controller to insert a new row into the appropriate table that includes the quest identifier, the user identifier and the date and time of when the quest was finished. This is also done for achievements and in this way we can control how many times quests and achievements can be completed per user. If a reward is tied to the achievement or quest it is also inserted into the database through the network controller.

5.5 Admin notification system

We also created a basic mailing system that has connection settings for a certain Gmail account. It also has a list of recipients and we use this mailing system to send e-mails to the recipients to notify them of low stock in the store. This is triggered automatically when the system detects that a certain item has run out. Items in the store all have a variable that tracks that item's quantity which is reduced every time that item is purchased. As this depends on the quantity variable being updated correctly when the store is re-stocked (as well as no one stealing from the store), we also included an option for users to report low stock through the store menu.

5.6 Navigation

There are three Unity game scenes in iGos 2.0; Login scene, Store scene and Profile scene, with each having it's own UI layer and logic layer. When the application is started, the Login scene is loaded and from there the user can get to the Store scene by clicking on his/her avatar and to the Profile scene by holding down the avatar button for about 2 seconds. This we managed by measuring the time from which the touch starts to when it ends, so it is actually the action of lifting the finger that performs the selection. This was quite useful when implementing the scrolling function of the application, whereas if the finger moved before a certain amount of time, no clicking would happen. If a click would occur the logic layers would load up the appropriate scene.

All input from users has to be handled correctly to make the user experience as enjoyable as possible. That's why we created our own input handler that tells the system what to do based on user's input which knows the difference between short touches, long touches and swiping. By having a single input handler, it was easier to fine tune it for a better experience in every scene.

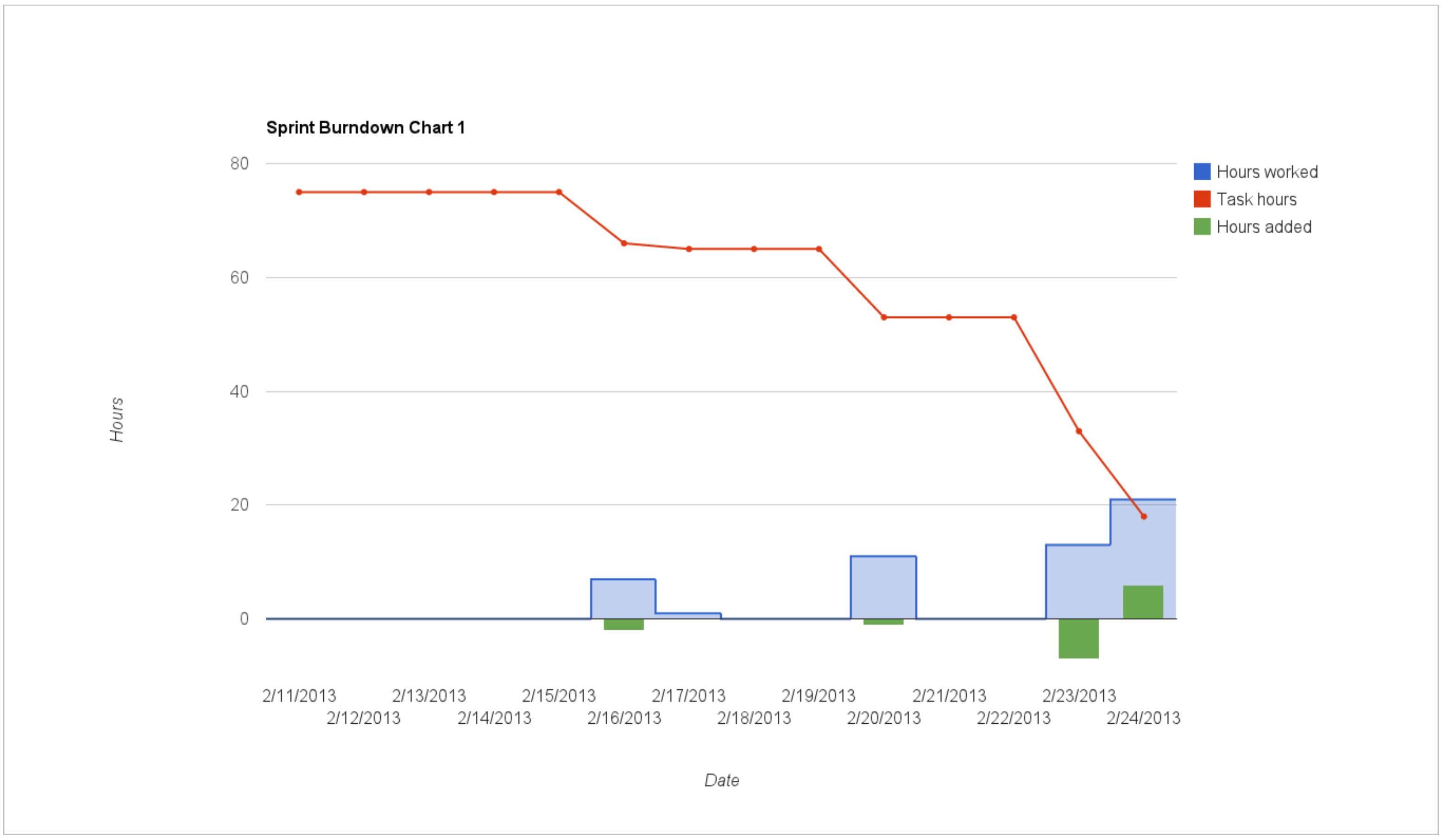
6. Scrum

Sprint 1

It took a while to start because of issues with getting the Unity pro licenses since without them there was no subversion server. It took even longer to acquire the Test Star plugin which delayed unit tests.

We were initially cautious about adding stories and estimated that we would finish roughly half of the basic login and store interfaces. The sprint went better than estimated and most of the basic functions and look for both store and login were finished. Many of the basic classes, like users and items, were also added.

As seen on the burndown chart there was no work done until on the 16th, or when we acquired the Unity licenses. We did not manage to finish the sprint and several lunch menu related stories were moved to sprint 2.

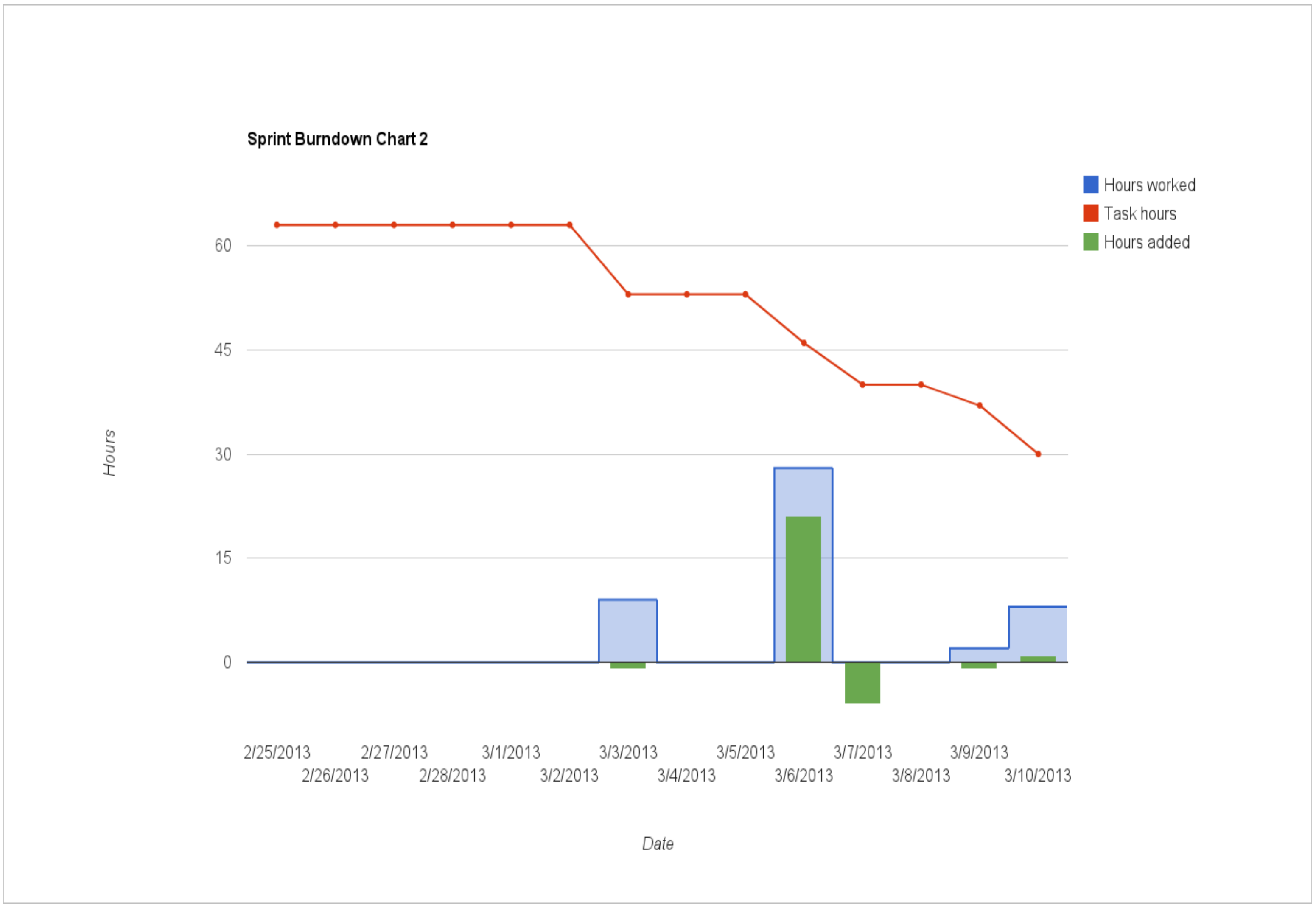


Sprint 2

In this sprint we added an interface for the lunch menu and separated the logic from the UI in the store and login menus. Additionally we started work on a network controller to talk to the server, however it did not go very far. Several stubs were created to act as a database so that we could simulate how the application would run once the network controller was up and running.

Getting Test Star to work took some time but eventually two tests were successfully created.

One issue that arose was when one of the laptops crashed. However, all of the data on it that had not yet been committed was recovered and a replacement laptop was quickly acquired.



Sprint 3

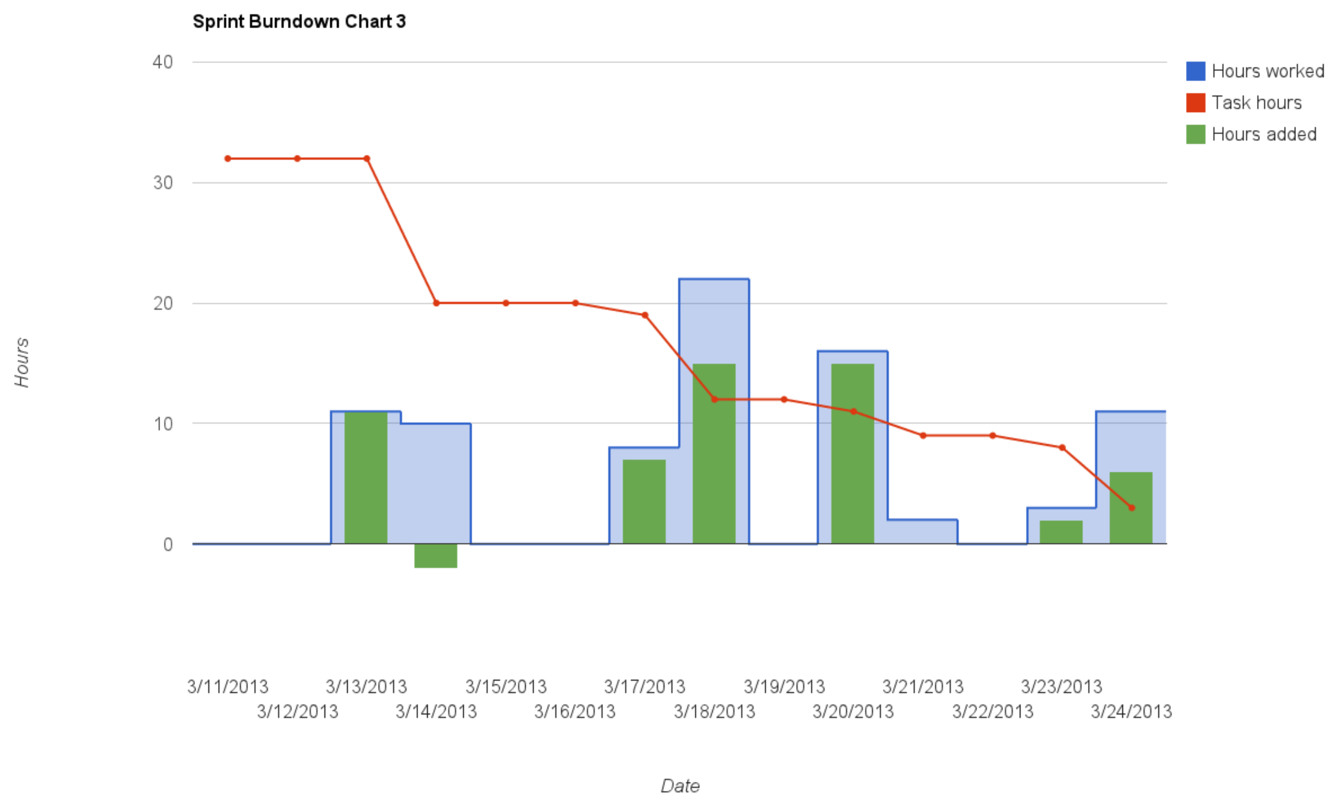
Since we did not have the Amazon server available at this point we experimented with using a locally hosted server using Microsoft IIS 7.5 which was in turn connected to a local database. The idea was to have the network controller connect to this server and then transition the service layer code to the Amazon server once it was available.

We expanded the network controller and made several functions that serialized data to XML files in order to send them to the server who in turn deserialized it.

Most of the original tests were completed and used to test changes to the code.

We originally did not assign enough stories to the sprint log and so we added more and more as the sprint progressed.

Additionally, in this sprint we got images from a designer that enabled us to skin the existing interfaces. However, we badly underestimated the time required for skinning as it was more complicated than we had expected. For both of those reasons we saw several spikes in the burndown chart.



Sprint 4

During sprint 4 all the group members were extremely busy working on final projects in other classes and we therefore decided to simply skip sprint 4 and focus on the other projects.

Sprint 5

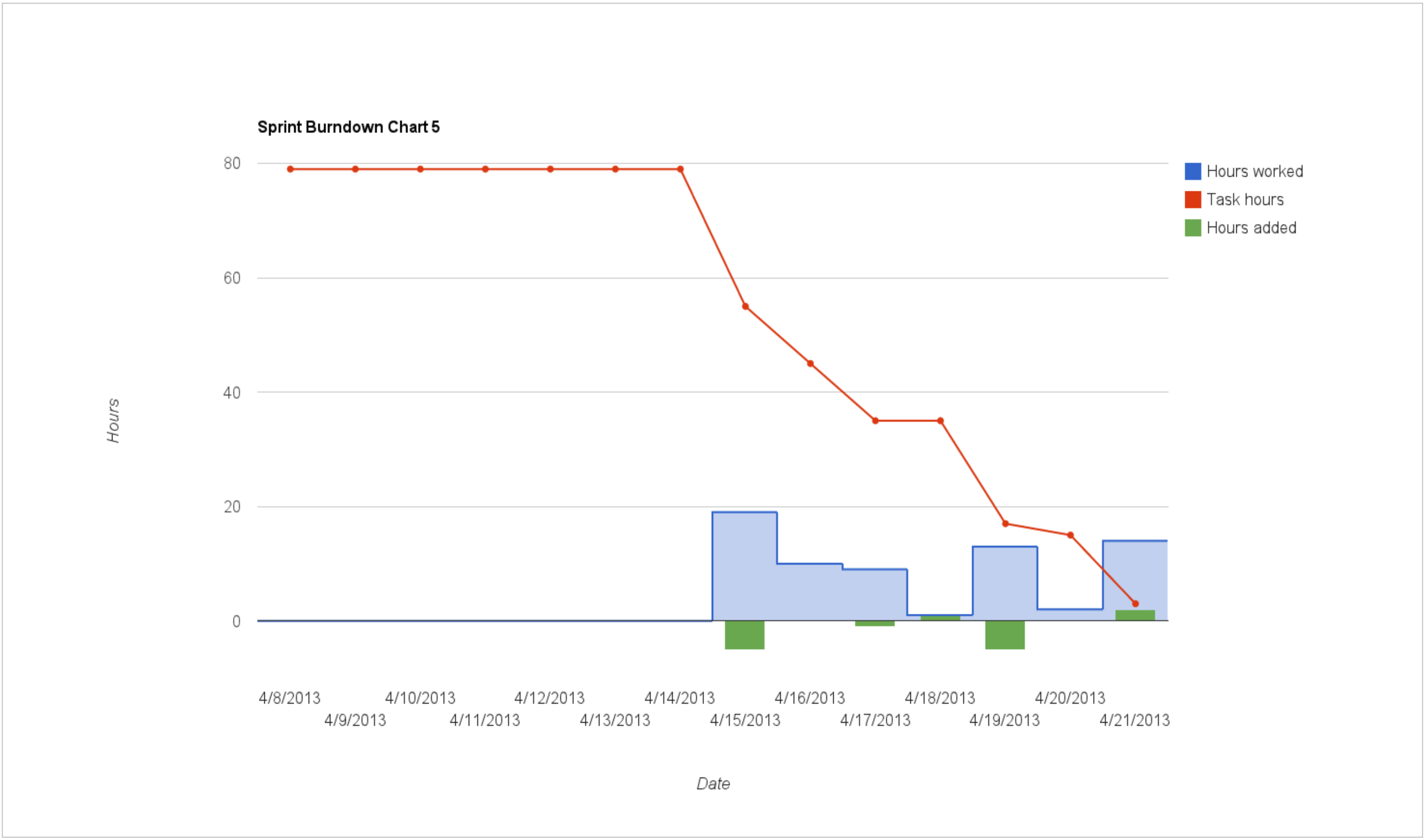
Major changes in sprint 5 included a total rework of the network and service layers. After a discussion with the product owner it was deemed that the current version of the service layer was unsatisfactory. It was therefore rewritten and this time based on the Entity framework. As we now had access to the Amazon server the service layer was built there and connected to a database on the same server. By writing several different models in the service layer that automatically create appropriate tables in the database there was little need for us to directly interact with the database.

The network controller was also changed and could now receive and send data to the service layer using JSon requests. All other controllers now communicated with the network controller instead of the previous stubs.

We changed the stubs so that instead of acting as database stubs they now use the network controller to seed the database when required.

A profile interface was created, meaning that now we had created most of the important interfaces. All users also had a unique avatar saved in the database although it was not yet customizable by the user.

Early in the sprint there was no work done because of the other final projects, however once they were finished we quickly got back on track.



Sprint 6

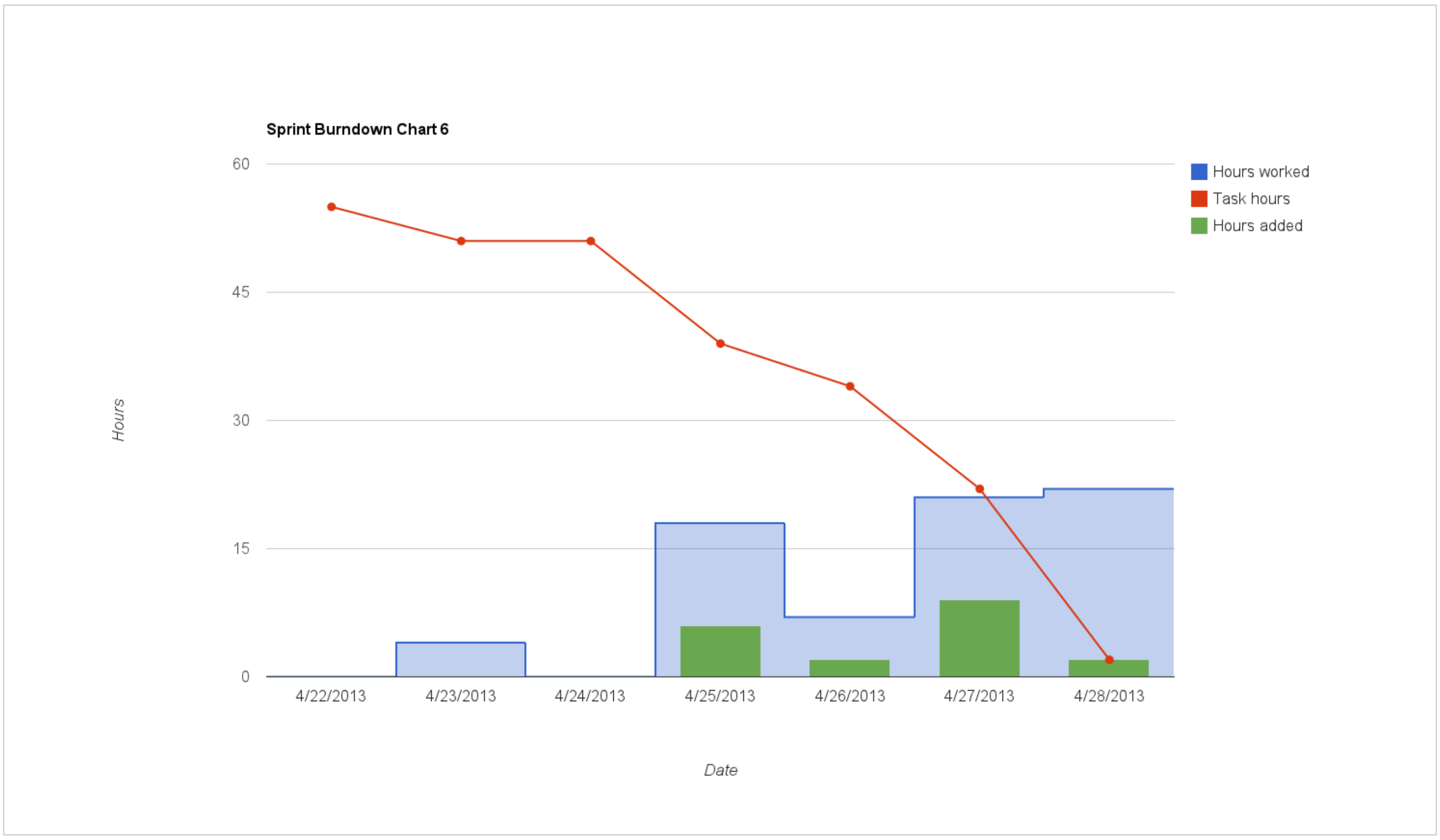
Achievements added based on the observer pattern.

A rough XP system added where the XP and Level variables of a user are updated when purchasing items based on a certain amount tracked by a variable in store items.

Two more avatars were added to the user class since we wanted users to have three different customizable avatars which he could seamlessly switch between. We changed the profile UI so that it displays all three avatars and also menus where you can see the different avatar items and select them. Selected items then get added to the currently active avatar.

A news feed was added to the login screen, showing the latest purchases. We also added a purchase history menu that used a calendar look. When opened it shows the current month, allowing users to see purchases per day and browse through months. This took much longer than originally planned since the designer originally created an unsatisfactory image that did not account for months with 31 days in them. We therefore had to redesign the image ourselves.

As the sprint started in the middle of final exams in other classes there was not much work done initially, however after the 24th the chart approached normal values.



Sprint 7

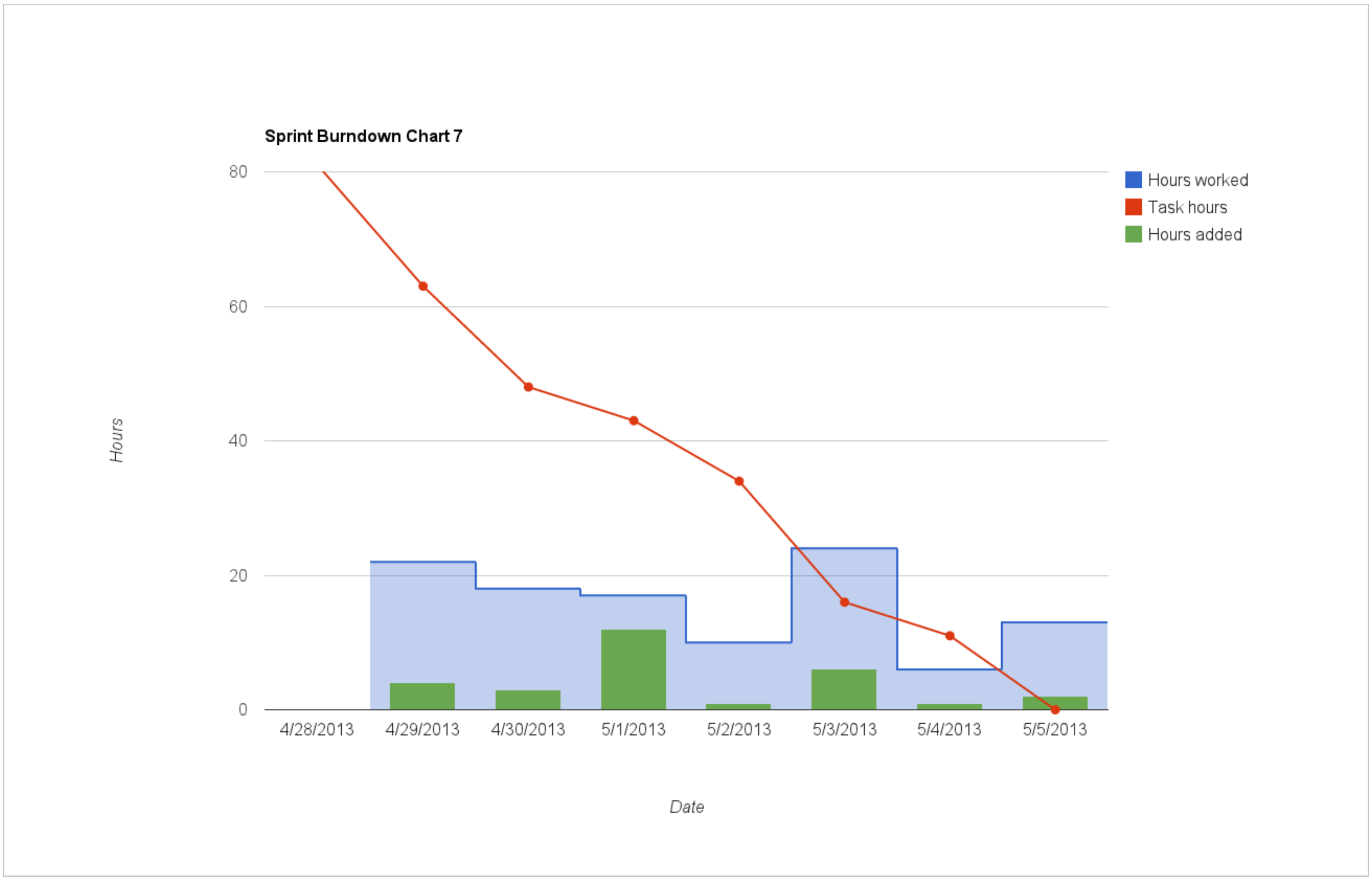
Most of the gamification happened in this sprint. A class system was added so that users can gain experience points in a certain class and increase their level. A table was added to the database to track each user's advances in the classes. When a new user is added to the system through the service layer it automatically creates two new entries in the class table, one for each of the two classes in the system. These entries have the user's id, an experience point variable and a level variable.

A quest system was added that works almost identically to the achievement system.

We changed the avatar items so that they have a boolean variable that denotes whether the item is available for all users or is locked. If an avatar item is marked available it is displayed in the profile of all users. If it's locked it will only be displayed in a profile if it has been unlocked through a quest or a achievement. Information about unlocked items is stored in a separate table.

We decided to change the login screen so that it only displays the users currently seen on the screen. All other users are simply not rendered until the screen is scrolled to them. This ensures that the system does not lag even with a large amount of users.

Due to lack of time it was decided that there was not enough time left to implement leaderboards. We would therefore have to cut it out of the schedule. Other than that the sprint went very well.



Sprint 8

The last sprint before the code freeze. To ensure that no data is lost we added multiple cache files that all changes are saved to before they are sent to the database. If the server returns the correct response the cache changes are discarded, otherwise they are kept.

Every time something is changed the system tries to sync with the database by reading from the cache files and sending any data there.

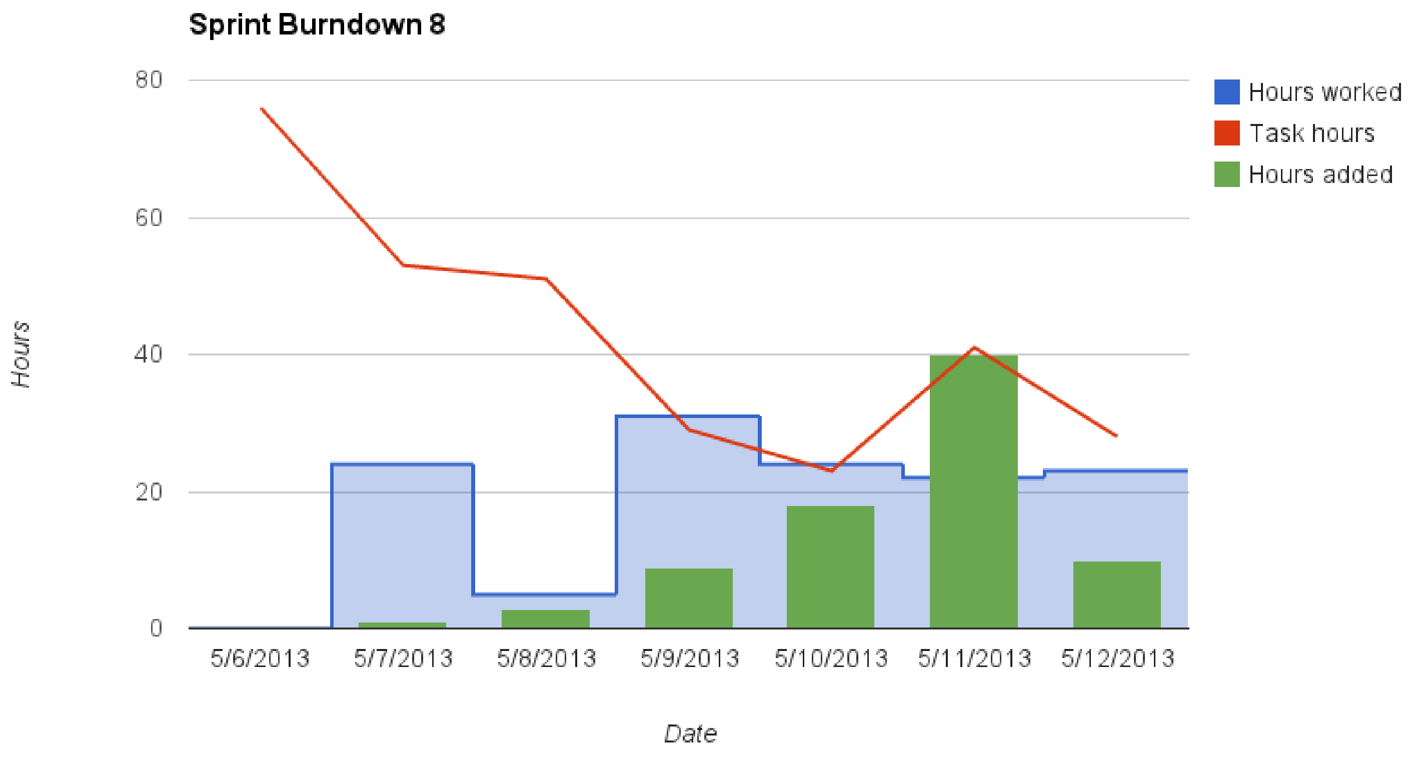
A master controller was added to keep track of all the other controllers and ensure that they retrieve data from the database in the correct order. It also makes the controllers try to retrieve everything again from the database at certain periods. On an hourly basis it makes the network controller sync by reading from the cache files and every three hours it makes other controllers retrieve all the application data again (such as users).

By changing all local variables when changes are pushed on the database we ensure that even though the network connection is down the user will immediately see the changes he has made.

Finally, the lunch menu was once more changed so that the user can now see what lunches he is signed up for and cancel them.

A problem with the ipad arose in that we were using Linq to sort lists. Certain functions do not function on iOS systems so we had to write our own sorting functions to handle that.

The sprint went very well and we decided at the end of it to add every possible polishing story we could think of to it. The most important of these were worked on in this sprint and the rest were handled in the code freeze.



7. Project Retrospective

Unity3D proved to be quite efficient in rendering our project for a smooth experience on the iPad. We got to draw up the mockups ourselves with only a few guidelines from the project owner, which was then sent to a designer and therefore we strongly affected the final outcome of the project.

As we learned more and more in programming the graphical user interface (GUI) in Unity3D we developed a workflow when implementing a GUI based on a design made by the designer.

At first we ripped out all of the assets of the photoshop files and placed them almost by guessing in which position they should be and then adjusting based on if we thought it was on the right place or not.

But as we progressed we learned that we could change the photoshop file to exactly match the amount of pixels on the iPad. So when starting on a new scene we would extract all of the sizes and positions of the resources down, so when we actually started to program we knew exactly how many variables we needed for the positions and textures and were much quicker to program the GUI.

As the project progressed it gradually evolved to better fit our time restraints with regards to other schoolwork and activities.

Originally we planned to have 3D avatars that could be fully customized like changing the eye color, clothes and background. However, after we received the avatar designs from the designer (shown in section 3) we decided that 2D images would work wonderfully with it.

As mentioned earlier we had also planned to have leaderboards where users could see the top users in terms of levels, achievements unlocked and number of purchases of specific items. However, this had to be scrapped as there was not enough time to do it properly.

When users were marked as “sick” it was supposed to show up in the news feed. However, we felt that this info would simply either get lost or clutter up in the news feed. Instead, we felt it would be sufficient to order the users on the login screen by their status so that sick users show up last with a special icon to mark them as sick.

Similarly, we did not implement any special functions to connect RSS feeds to the news database. We felt that this would be optional and company-specific so that companies who wished to connect an RSS feed to their news feed could easily use existing models to do so.

Finally, we did not implement methods to rate a lunch that you were signed up for as it was not deemed a core functionality. However, it could easily be added into the purchase history in the future.

We learned to adapt scrum to our own needs. The first few sprints did not look the way we had wanted them to, but as the team got more involved in the project and more used to sprint planning the sprints started to take a more natural form, especially when other school classes were over and all of our focus was on the final project.

8. Conclusion

The project was ultimately successful; the team was able to make a complete product that is ready for deployment. All the core functionality works while having a good and solid framework which easily allows modifications and additions.

Making the app simple and easy while still making sure that it is fun to use proved to be the most challenging part of the project. This means that the application needed to have a clean interface that allowed users to get what they wanted from the system very quickly. We feel that we have achieved this in that users can easily find themselves from a list, by scrolling down the list of all users or by categorizing them by departments. The store menu sorts items according to each user's most frequent purchases, which allows for easy access to his/her favorite. Checking out is just a single click and therefore the whole process of purchasing an item can be done in only 3 clicks! In this way the application achieved its goal of being simple and convenient for every user.

Each purchase will add up to a user's achievement list in some way and grant him/her experience points. By rewarding users with experience points and potentially achievements and quest rewards the application seeks to enrich the user's experience. We believe that we have succeeded in making an application that will leave users happier than they were prior to using the system.

Each user has the option to customize a few aspects of his/her character with relation to what they have done in the app. Users can gain new items for their avatar by completing achievements and quests. In this way each user can have an avatar customized to their tastes, that by reminding them of completed achievements, can leave the user with feelings of gratification.

We also believe we have created a very stable platform that can be easily customized to suit individual company needs. Basing our service layer on the Entity framework means that modifying tables and working with data in the database is very simple. We have also strived to maintain proper programming standards in that many rendering functions (such as the one that displays a user's avatar) are stored in separate classes.

We ended up spending 1550 hours on this project which is within the scope that was estimated at the start. All in all we feel that the project went very well and we are happy with the result.