UPPSALA
UNIVERSITET

IT 11 087
Examensarbete 15 hp
December 2011

Taking Notes on A Mobile Device; An Unsuccessful Agile Project

Emilio Nyaray Valenzuela

Institutionen för informationsteknologi
Department of Information Technology

UPPSALA
UNIVERSITET

Teknisk- naturvetenskaplig fakultet
UTH-enheten

Besöksadress:
Ångströmlaboratoriet
Lägerhyddsvägen 1
Hus 4, Plan 0

Postadress:
Box 536
751 21 Uppsala

Telefon:
018 – 471 30 03

Telefax:
018 – 471 30 00

Hemsida:
http://www.teknat.uu.se/student

Abstract

Taking Notes on A Mobile Device; An Unsuccessful Agile Project

Emilio Nyaray Valenzuela

This thesis describes the intended methodology, encountered obstacles, final results and insights from developing a mobile application. It started out as part of the work with a web-based, collaborative, graphical, real-time information-management tool, called Canvas (the service).

The initial purpose of the thesis project was to develop an application that was able to gather notes; the specified note types were text, image and sound. These notes would, after creation, be synchronised (possibly in both directions) with a web service, for ubiquity.

Shortly before the work began, an executive decision was made to stop all development on the service due to a change in direction, resulting from the fact that the rate of adoption of the service by new users stagnated and a diminishing amount of funds.
It turned out that issues with the communication and the unforeseen changes to the plan were challenging to handle and, in turn, resulted in the prototype not being fully implemented.

Handledare: Johan Ekhager
Ämnesgranskare: Mats Daniels
Examinator: Anders Jansson
IT 11 087
Tryckt av: Reprocentralen ITC

Contents

Chapter 1

Introduction

This chapter accounts for relevant events that occurred and decisions that were taken before work with the project began, as well as some necessary introductions.

The work behind this thesis was done at Creado AB in Sweden with the intention of implementing a working prototype of a smart phone application that could synchronise with the company's cloud-based service, Canvas, henceforth known as the service.

1.1 Smart phones and Mobile Applications

With current advances in technology, a new breed of consumer-oriented cellular phones, within the smart phone segment, have emerged. These phones have started to ship with large touch-screen displays that serve as the main means of interaction with the phone. Although the term was coined in the end of the 1990s, consumer awareness of it was still lagging behind.

The reason for the introduction of the term 'smart phone' is their relatively advanced hardware which allows them to perform tasks that either would drain the battery in an unacceptably short amount of time or take too long for it to be useful to perform a given task on a conventional cellular phone.

At the time of writing, in the end of the summer of 2011, smart phones, along with service subscriptions that contained data plans had become a viable and even, to some extent, popular alternative for consumers.

1.2 The iPhone, iOS and Applications

One company that has had a leading part in defining how smart phones should perform and look is Apple Inc., with the release of its highly regarded iPhone which uses an operating system called iOS, which currently is at its fourth revision (iOS 4), with iOS 5 set to be released during the fall of 2011.

With the release of the second version of iOS, inline with the suitably named 3G-capable iPhone 3G, it was also announced that a new service would be offered, under the name of the App Store. It was stated in a press release[1] that the "iPhone 3G includes the new App Store, providing iPhone users with native applications in a variety of categories". Applications that run on smart phones are colloquially known as "apps", but the use of the word has started to include applications in general.

An application, in this context, is a modular piece of software that can be "dropped" into a system and used without the need to reconfigure the hosting system much or even at all.

1.3 Application Development

When developing applications for iOS devices, one mainly works with a language called Objective-C; the language is defined as a small set of extensions to the standard ANSI C language. Its additions to C are mostly based on Smalltalk, an object-oriented language[2]

The recommended tool for application development is XCode, which is an integrated development environment (IDE) supplied by Apple, for applications targeted at both OS X (the company's operating system for its lines of stationary computers and laptops) and iOS. Along with providing XCode, Apple also provides two important frameworks that consist of libraries, API's and runtimes that enable developers to get started and productive faster. These frameworks are known as Cocoa and Cocoa Touch, and were used to build OS X as well as iOS.

[1] http://www.apple.com/pr/library/2008/06/09Apple-Introduces-the-New-iPhone-3G.html, retrieved August 26th, 2011.

[2] http://developer.apple.com/library/mac/documentation/Cocoa/Conceptual/ObjectiveC/ObjC.pdf, retrieved August 26th, 2011.

1.4 Canvas, The Service

The service is a web-based[3], collaborative, graphical, real-time information-management tool that allows users to, among other things, see pieces of text, hyperlinks, pictures, sticky notes and lines on a plain, white, canvas. When a canvas is being edited, the changes are forwarded to other users that are viewing it, in near real-time via an internet connected server that is running the back-end software.

The purpose of the mobile application, which would be targeted towards iPhone devices running version 4 or later of the operating system iOS, was not to implement the regular client behaviour, but rather to give users of paid accounts additional value by giving them a way to capture text, pictures or audio recordings with their smart phone in situations where using ones computer is not feasible, for example, when on a short bus ride. It can, in other words, be said that the application at least would be able to perform uni-directional synchronisation.

1.5 Change of Direction

With only a few weeks left before the thesis project was supposed to begin, it was clear to the executives of the company that there was not going to be enough resources available to fund further development of the back-end during the thesis projects duration or even the immediate, foreseeable, future. This meant that it would not be possible to make the mobile application send data to the service, thus making it a moot point. Another relevant issue was that most of the user feedback had shown that while users were impressed with what the service was capable of, it was far from obvious how users could use it to improve their productivity.

These things, when weighed together, were the basis of the executive decision (which was made in agreement with me) to make a few minor changes in the specification of the mobile application. With the changes, the application would instead interact with a slightly different, more generic, back-end.

To summarise; it was still deemed a worthwhile enterprise to have a mobile application developed with many of the originally intended attributes

[3]Which means that it runs within the users web-browser software.

and behaviours, still making the project a suitable thesis project, despite the changes in the specification.

1.6 Alterations of the Specification

As previously mentioned, it was decided that the application be developed with a few modifications; the concept of canvases was to be dropped, being replaced by tags instead, allowing users to group content into several overlapping sets instead of just one (which is how canvases would have been translated to the tag concept) by letting them add several tags per inserted piece of content. Even though these changes (or a new back-end) wouldn't be implemented immediately, it wouldn't be too expensive, in terms of money and time, to implement an API that could handle content in the new way, making a mobile application that would fit the changed requirements a feasible and interesting goal.

The original time budget stated that the first two weeks of the project be spent on learning Objective-C, XCode and the necessary minimum to start developing for the iOS platform on the iPhone. After that, a week would be invested in discussing and designing the synchronisation mechanisms in parallel with the original back-end developer that implemented the service, before finally implementing the different aspects of the application.

Due to the previously mentioned executive decision, it was clear that the synchronisation issues, both with respect to design and implementation, needed to be postponed until it was clear how the relevant details of the new service would take shape or at least until it was decided how communication should occur with the back-end in case the new service would be implemented.

Chapter 2

The Problem and its Structure

The project started out with a learning period of two weeks, where I was supposed to research and learn about application development, primarily for iOS 4 on the iPhone 3GS and iPhone 4, which were the two most recent iterations of the iPhone. On its developer pages[1], Apple states that MVC[2] is central to a good design for a Cocoa application. This was taken into account and quickly became a natural part of the design process.

When it was time to design the architecture of the application, the task was delayed. The delay was partially caused by the tasks dependence on having interface sketches ready, or at the very least, use cases, since the hierarchy by which the different view controllers relate to each other is defined by them, but also due to the uncertainty that rose from the situation.

This chapter takes a top-down approach and states what the design problem of the project is and then describes the following: the architecture of the system that the mobile application is a part of, the MVC-pattern and other design choices that formed the mobile application itself.

2.1 Problem Description

The problem that the design described in this chapter aims to solve is that of structuring a mobile application so that it can be used to record thoughts and ideas in the following ways: in writing; with new or existing pictures

[1] http://developer.apple.com/library/mac/#documentation/General/Conceptual/DevPedia-CocoaCore/MVC.html, retrieved August 26th, 2011.

[2] Model-View-Controller

or; with new or existing audio recordings.

A feature that the application should implement is that it should transmit stored objects that have yet to be synchronised with the service as well as ones that have been edited since they were last uploaded to the service. This is to make notes globally reachable, or ubiquitous, in some sense, since they would be accessible through a web-based interface once they'd been uploaded.

2.2 Architecture

A system that the mobile application communicates with (one has yet to be developed) makes its functionality available via an interface that can be interacted with through HTTP operations, which makes it simple to access and update the stored set of objects. Data is supplied to the service only after being serialised through JSON[3] encoding, that is, data is supplied as a string, describing a collection of objects and their respective attributes through a parameter in a GET or POST operation[4] in the HTTP protocol.

Since the intended user is supposed to record thoughts, pictures and sounds with the application in situations where a computer might not be an appropriate tool, the application must make the assumption that the application won't be running in an internet-connected environment at all times. This shows the need for a synchronisation mechanism and one will be described later in this chapter.

2.3 MVC

The following subsections explains Apples interpretation of the MVC-pattern and how the mobile application's design adheres to it. Models, Views and Controllers are briefly mentioned and thereafter explained in greater detail in their own subsections.

In short, one can claim that the Model, View and Controller in MVC are roles assigned to instances of various classes within an application. These roles differ as the Model is used to represent stored information and whatever

[3]JavaScript Object Notation is a format for plain text that is used for the serialisation of structured data. http://tools.ietf.org/html/rfc4627, retrieved August 26th, 2011.

[4]GET and POST, are two commonly used methods of uploading data via the HTTP

relevant meta data one might have about it; the view is capable of displaying things to the user when told to and notifying its delegate of when the user interacts with it and; the controller manages both the Model and the View, figuratively filling the role of the link between the Model and the View. The controllers are often said to contain the "business logic" of an application.

Application Design

After the initial learning and some minor experimentation with the platform had been done, a design was made where there were controllers for the following responsibilities; listing the items that had been entered; managing preferences and synchronisation; managing text notes, picture notes and audio recordings. Each of the controllers had a corresponding view that it managed. To clarify, there were three views that would be used to let the user manage the three different types of content that would be managed.

There was a 1-to-1 relation between the number of controllers and views since the responsibilities of the controllers were suitable to keep in one object, though in a future revision it would be better to factor out the synchronisation management from the preference controller in order to add more functionality and to make it concurrent. The initial purpose was to store the necessary credentials needed to push data to the system from the mobile device.

In summary, the mobile application needed to handle; listing of current objects; text, pictures and audio (**1** controller/view); creation, displaying, modifying and deletion of objects (**3** controllers/views); managing user credentials and triggering synchronisation. (**1** controller/view).

As mentioned, there is a 1-to-1 relationship between the number of controllers and views, this means that each view, in its entirety, is managed by a single controller. This is good for maintainability, because it restricts (or encapsulates) code that corresponds to a specific area of functionality in the application within objects of a certain class.

Controllers

A controllers role in a given context is usually defined by its base class as there are various classes provided by Apple for many of the most commonly

protocol.

occurring behaviours in application views. One example of a provided controller is UINavigationViewController, which keeps track of a stack of active controllers and manages transitions between them, also keeping track of how the menu-bar at the top of the screen should look by querying the currently visible controller, for most of the content, and possibly the one below, for the back button, which will bring the user back to it.

Each controller has a reference to its corresponding view because it needs to be able to make changes in order to notify the user of changes in the program's state or to show an object that is about to be edited. The controller may also toggle the enabled state of a button or value of a text field, among other things.

The controllers responsible for managing stored objects also need a way of updating and reading values from them that will be displayed in the view, so they keep a reference to what is called a model. A model is merely an object that represents an instance of some entity, abstract or real, that is stored in the system, that exists to simplify the management of said entity.

Views

In the main view, the user will be presented with; a grid view, listing all the objects that have so far been entered into the system through the mobile application; a button that will trigger the displaying of the preferences view and; a button that will present a list of available options for creating new objects.

The views of the application will be inherently simple since the data that will be displayed (or played back in the case of audio objects) merely is two short pieces of text, a name and a body of text or a picture (play/pause button for audio) for views that display stored objects. In the case of the list and preferences view, the application will simply plug into the infrastructure provided by apple for the purpose of application development, rather than create some new way of moving about a view hierarchy.

Views in the application will make their owned and visible components available through their public interfaces, allowing their respective controllers to access them so that values can be changed in order to show the user meaningful information, but this is not enough to handle user interaction. In order to support user interaction, the view must be made aware of which object it should regard as its delegate, that is, which object it should notify

of when an event, for example, button press or other event that requires an action from the views coordinator, occurs.

Models

As it has previously been described, the model objects have the responsibility of managing the stored data and keeping it consistent, so as to offload the controller, but also to keep the applications concerns separated.

The models available in the current design are the following: Thing, TextThing, CameraThing and AudioThing. It should be noted that Thing is not a model in the same sense as the others, since it is an abstract class. It is used to group attributes and behaviours that are common in the other Things into one class to simplify their management.

Here follows a listing of the created models, that is, classes and the information they store.

- As mentioned, the **Thing** class is used to group attributes common to the three other **Thing**s together. A **Thing** is never instantiated, it is impossible to do so as it is an abstract class, but it can be indirectly instantiated through one of its sub classes.
- **TextThings** are used to manage **Things** that store a textual note, the note itself is stored as a string in a single attribute.
- **CameraThings** are used to manage pictures, whether taken specifically for use in the application or if they previously existed. The added attribute that **CameraThings** have is a path that allows the application to locate and load the picture from the phones long-term storage.
- **AudioThings** keep track of audio recordings, they, just as **CameraThings** do, keep track of a resource path which tells the application at which location in the phones memory the associated recording can be found.

2.4 Utilities

A common hurdle when one is developing applications is to maintain application state between executions, which is especially important in environments

with restricted resources, for example, mobile phones with limited battery charges and processing power.

It has been a common occurrence that utilities for serialisation and persistence of data be re-invented for each new project that is undertaken or, in more sustainable environments, re-used and improved each time they're used in a project. This still causes a significant amount of overhead for developers since it takes time away from solving issues directly related to the current project.

Apple has dealt with this issue internally since OS X includes various applications and tools that implement useful functions and allow the user to perform different tasks, storing state between executions and the framework that has been developed for this purpose was named Core Data. Core Data has its roots in the NeXT[5] framework Enterprise Objects Framework[6], which was used for similar purposes.

Core Data

In Apples documentation, it is stated that "[the] Core Data framework provides generalized and automated solutions to common tasks associated with object life-cycle and object graph management, including persistence"[7]. This means that it handles the objects and relations that the developer manages throughout her application, freeing the developer to handle, transport and modify data without having to worry about how it is persisted between launches of the application.

Core Data works by storing instances of entities – objects that belong to the same model class – to the secondary storage, usually a hard disk, flash device or memory card, and later loading them in a lazy[8] manner. This allows for performance gains since the memory won't fill up with objects that aren't going to be used.

[5]NeXT was acquired by Apple in 1996, http://web.archive.org/web/20020208190346/http://product.info.apple.com/pr/press.releases/1997/q1/961220.pr.rel.next.html, retrieved August 26th, 2011.

[6]http://developer.apple.com/legacy/mac/library/documentation/WebObjects/UsingEOModeler/UsingEOModeler.pdf, retrieved August 26th, 2011.

[7]http://developer.apple.com/library/mac/documentation/Cocoa/Conceptual/CoreData/CoreData.pdf, retrieved August 26th, 2011.

[8]Lazy, in this context, in the sense that it only loads objects when they are accessed.

Entities

Core Data uses a concept called entities to model instances of objects that will be managed in an application and the relations between them, this means that Core Data stores the objects – along with their respective attributes – and their relations, allowing application developers to just manage their object graph and have Core Data persist it. The Core Data tools available in XCode aid the developer in creating data models, but in addition to that also are capable of generating Objective-C stubs for the modelled entities, which is what will be done in the case of the current mobile application.

Thing

The models mentioned earlier in the chapter are all modelled with Core Data and **Thing** is the super entity (that is, super class) for the three of them. Its attributes are *creationDate*, *name*, *pushed*, *remoteID* and *syncDate*. The attributes are used to store the following meta data (in the same order as they appear); the date that the **Thing** was created; a name, either chosen by the user or picked automatically by the application; a Boolean value to store whether or not the **Thing** ever has been pushed to the service; a string to keep track of the ID that the **Thing** has been assigned in the service and finally; a date to keep track of when the **Thing** was last pushed to the system, if it has been pushed.

TextThing

The most basic type of **Thing** is the **TextThing**, that is used to store text, it has one additional attribute, namely *body*.

CameraThing

Pictures are represented by **CameraThing**s and they have one additional attribute, *resourcePath*, that contains a string representation of the path to the associated **Thing**s location in the phones memory. The *resourcePath* is, however, not pushed to the service – since it would make no sense to store where on the phone the resource could be found on the phone, in the service

– instead, the picture is base64 encoded[9] and sent with the meta data kept in the parent **Thing**.

AudioThing

The third, and last, sub entity/subclass of **Thing** is the **AudioThing**, which is used to represent audio recordings that the user makes and is handled in the same way as **CameraThing**s are.

Fetch Requests

In order to fetch the persisted object graph and even to do some higher order programming involving it, Apple has provided a few classes that abstract away the task of loading, sorting and filtering among requested objects, but before those can be used, one must create a fetch request. A fetch request is basically just an instance of a class that operates on a given persistent store to retrieve stored data, it may also be supplied with helper objects that can be used for filtering or sorting. Sorting is done by supplying the fetch request with a sort descriptor, which in turn has been configured to aid the fetch request when deciding on how to order the entities that will be returned. By instantiating objects of a class used to represent predicates (Boolean expressions that decide whether or not an entity, or object, is selected) one can filter among the initially returned objects, without the need to know how the filtering is done.

Synchronisation

It would be ideal if the application uploaded stored objects to the service immediately after the user created them, but due to time constraints, the design would only allow for synchronisation to be performed when explicitly requested by the user.

Since Apple has yet to provide a simple API for making simple HTTP-requests to RESTful[10] services, an existing, open sourced, library with a permitting license should be used, so as to not reinvent something that al-

[9] Base encoding is used to store or transfer data in environments that are restricted to US-ASCII data. http://tools.ietf.org/html/rfc4648#page-3, retrieved August 26th, 2011.

[10] REST – Representational State Transfer – means that a representation of state is transferred with the data used in the state, as defined by [Fielding00]

ready exists. In my preparations I have found Seriously[11], by Corey Johnson of GitHub Inc. to be a likely candidate for the task.

[11] https://github.com/probablycorey/seriously

Chapter 3

An Agile Project Work Model

The project was to be managed with a simpler variety of the project management framework SCRUM, which has its theoretical foundation in [Takeuchi-Nonaka86]. SCRUM allows for the project to be handled in a flexible manner if it is discovered that some requirements or assumptions turn out to be irrelevant or harmful for the project, either directly or indirectly. Projects that are managed in a way that allows for flexibility in this manner are said to be agile, since it allows for agility as far as the team members can stretch their abilities.

The main reason for basing the project management on SCRUM is that it divides the work into small tasks (the backlog) and allows for work to be done in iterations, generating an incrementally better (that is, better by any agreed upon criterion) product at the end of each iteration. These iterations are known as *sprints*.

3.1 Roles

Since SCRUM originally was intended for larger projects with more people involved than the one undertaken in this thesis project, the roles and work flow must differ to some extent. In SCRUM, three roles are defined; Product owner, ScrumMaster and Team. The product owner is responsible for representing the stakeholders and prioritising the backlog (that is, the list of tasks to be performed), the ScrumMaster ensures that the SCRUM

methodology is followed correctly and that the team performs the tasks that need completing first.

In this project, the supervisor will act as both the product owner and the ScrumMaster since he is both the stakeholder and person that knows how the various tasks should be prioritised. Since the purpose of the project only is to build a simple mobile application and due to the nature of thesis projects being done by lone students I alone shall take on the part of the team.

3.2 Deliverables

There are usually 2-4 weeks within a single sprint, but due to the project being scheduled for 10 weeks, of which 2 would be spent on the final report, the supervisor and I agreed on that it would be better to let sprints last 1 week instead. This would allow for the work to be conducted in a more modular way, that is, that it would be simpler to re-order steps where their respective interdependence's would allow it.

Each deliverable, the application at the end of a sprint, should be a prototype with the functionality described in the corresponding sprint description in the project specification.

3.3 Working Agile

The SCRUM approach to a project is to have regular contact between the three parties, so as to keep everyone informed of what is going on, but also in order to handle and discuss issues almost as soon as they appear, in an attempt to keep the work flow going.

When managing a project according to the SCRUM model, one should have daily meetings at a set time on each working day, so that everyone involved knows what has been achieved since the last day and what is going to be done during the day. As the backlog would not be of the same magnitude as ones in more common SCRUM projects daily morning meetings were not scheduled. However, the supervisor and I would still be in touch a few times per week, to be sure of that everything was going smoothly and according to the plan.

Without the daily meetings, it was decided that the supervisor and I

would get in touch at least once a week to follow up on the sprints and how the current one was progressing or how the latest one had turned out.

Chapter 4

Results

This chapter describes the results of the project, that is, the product at the end of the last sprint. While the purpose of this chapter is to describe the present state of the application and the main analytic part will be in the next chapter, there will be some interpretations of the state of the application, partly for clarity, but also to allow for a more rewarding analysis.

The application was not implemented to a degree where it could actually be considered to be a "fully functioning prototype", as was specified before the project was undertaken. It did, however, reach a certain degree of maturity and provide both supervisor and me with some very useful insights, which will be covered in the next chapter, after we've looked into exactly what, how – and, to some extent, why – some things went wrong.

4.1 The Application

The application was not completed fully in accordance with the specification at the end of the last sprint, as it was lacking in some aspects. Among other issues, the application can crash at seemingly random points in time shortly after being launched, which is believed to be related to memory management, or perhaps a misunderstanding of central platform concepts, since it was a common reason for most of the early bugs that were dealt with.

In the sections below, we will look into which parts of the application are complete, near completion, far from complete and not implemented.

Persistence of Objects

The persistence of data was achieved using Core Data, as the early research in the project had suggested it should be. Modelling entities (and even generating source code for the classes to be used in the application) with the tool that is built into XCode proved to be a rather simple task, showing the power of advanced modelling tools and frameworks.

Support for handling tags on objects (as properties of **Thing**s) was partially implemented, but proved to be complicated to handle and was finally left out since it wasn't essential for the application.

Multimedia-based **Thing**s – **CameraThing**s and **AudioThing**s – were unfortunately never used, so it can not be verified whether or not they work as intended, but since they were implemented in the same way as **TextThing**s they should work. The reason that they were never used is that the provided application components for capturing media were unexpectedly difficult to handle. The concepts weren't too advanced, but there were issues that couldn't be resolved during the work with the project.

Synchronisation

While the application has the ability to communicate with a prototype back-end, that is, a server that merely responds with some trivial response – which possibly relates to what was first sent –, it does not transmit the specific content of **Thing**s[1]. This is due to a lack of documentation and explanations of how to use the Seriously library, which also couldn't be properly explored due to time constraints in the project and some poor choices made by myself. These issues are discussed in further detail in the next chapter.

A common way of transmitting data in and out of RESTful services is to serialise and encode it with some simple technique. This can sometimes be done for strings that contain a lot of special characters such as ones normally used for escape sequences, string quotations or special letters that don't exist in the English language.

User Interface

While the user interface makes the available functionality accessible to the user, unfortunately, too much time was spent on the hierarchy of the views

[1]Specific content refers to the note body of a **TextThing** or the picture/audio file belonging to a **CameraThing** or **AudioThing**

instead of the functionality that the views would make available. The time was spent due to misdirected efforts in my priorities. The effort spent on the user interface was with the intent of implementing it in accordance with the available user interaction sketches made available by Creado. Just implementing functionality and making it accessible to the user would have been enough.

4.2 On the Final Delivery

The final delivery was made through the source code management service GitHub[2] to the company's private repository. GitHub uses the versioning software Git with some additional services created in-house in order to aid developers and companies with the management, overview and general control of their source code.

Prototype Not Finished

As made clear in the previous sections, the prototype was not finished on time, which is attributed to my choices while carrying out the project.

Even though the prototype is not complete it would not require much work to get it in a functional and useful state, since it can be used to take notes, which at some later point will be synchronised, if the synchronisation code is finished.

No Back-end

At the time of writing, Creado did not have a back-end for the application to communicate with, but was, however, in the initial phase of creating a service with the intention of using a modified version of the application.

[2] http://github.com

Chapter 5

Conclusions

As there were a few issues preventing the specification to be followed verbatim, for example, the change in direction of the project, they also complicated things further by interfering with each other.

While no claims of generality are made for the conclusions reached in the following chapter, the insights provided could still be of use in a meta study in the role of a case study, which is what they should be taken as. It is my firm opinion that more experience with open-ended group projects – as described in [Daniels11] – would have softened or even prevented the impact of some of the issues that appeared during the work with the project.

5.1 The Effort

In the end, this project resulted in a prototype a few features short of what was aimed for, along with some very useful insights for me and the supervisor, rendering the effort worthwhile for both parties.

The project entailed me learning Objective-C, programming for the iOS platform and a few lessons about short, small-scale, projects. This alone is good, as the mentioned skills can be practically applied, immediately, in a professional context. Furthermore, since the project was not a textbook success-story, it offers some useful lessons since analysing our shortcomings equips us to better avoid repeating mistakes.

5.2 User Interface

This section goes into how the work with the user interface caused issues in the projects work flow.

Would It Be Possible. . . ?

A fairly large amount of time was spent on implementing a specific view hierarchy instead of the necessary functionality that was supposed to be made available through said hierarchy. This is attributed to me giving an disproportional amount of attention to details – that were discussed with the supervisor and one of his colleagues – in relation to their respective importance.

An important reason that so much time was spent on the user interface is that the supervisor and colleague were misunderstood to be asking for the user interactions to be implemented in a certain way, when they were merely asking if it would be possible to do it that way.

In the context of this project, it is easy for a student to get the priorities of the project wrong, not having any vocational experience. In this particular case there was previous experience in prototype development over a relatively short period of time, as is with bachelor thesis projects.

A Question Not to Ask

It turned out to be inadvisable to ask whether or not I could perform certain tasks or not since it gave the impression that the priorities could be different than they first seemed or that they had changed. If the project had been undertaken in a corporate context, the situation would likely have been similar if not identical, but hopefully handled differently, since one imagines that seasoned project managers have a better understanding of how junior employees handle inquiries relating to additional functionality and tasks that add to the current overhead.

Student Self-Awareness

To help with alleviating and in some cases even preventing similar situations from affecting students' performances, they could be made aware of common (and often unfavourable) reactions that negatively affect their performance or focus in similar situations. What I needed to do, instead of averting my focus from the task at hand, was to keep focused on the task allotted for the time and not get sidetracked.

A simple way for a student to falsify its incorrect perception of an extraneous task's relative importance (if there is such a perception) is to ask

whether the task is critical for the project or if it should be written down and saved for performing after the bug-fixing has been done. This allows the student to set it aside, mentally, and progress with the important tasks.

Distractions

Another source of distraction for me were the often re-occurring conversations about minute details in the user interface, which, to me, signalled that they were of greater importance than they actually were, causing a slight shift in focus towards the user interface and away from the issues relating to functionality.

Since the specification of the application was adjusted while I still was researching the tools and platform that were going to be used, there were parts of the initial effort in producing the application that needed to be reconsidered and adjusted to the new specification when starting the work on the prototype. There was talk about interaction design sketches[1] that would be preferable to use as models for the user interface, but since these weren't ready when work on the prototype began, they should have just been left out of my considerations since there were more pressing issues than the visual layout of components.

The user-interface design-tool that is built into XCode makes it easy to modify existing interfaces, which is what could have been done, time permitting, at the end of the project, rather than putting effort into implementing a design that wasn't available until a few weeks had passed in the implementation phase of the project.

5.3 Communication

In general, the communication between the supervisor, involved colleague and me was good, it happened at regular intervals and allowed for the small group to be up to date with what was happening. Unfortunately, most likely due to the fact that the supervisor works with front-end[2] and the colleague with designs, the subject of user interface details and plans for what could

[1] Approximate drawings of what clickable elements are shown on the display and how they trigger transfers between different views.

[2] When working with front-end issues, one is mainly concerned with how the visible behaviours are implemented and how users perceive the feedback they are given.

be implemented in the future were often brought up, shifting the focus of the meetings somewhat towards user interface issues in general, leading me to believe that these were of greater importance.

Another cause of confusion for me was when the design-oriented colleague of the supervisor insisted on using the term pixel perfect[3], when the intended meaning was more along the lines of a well thought out design.

5.4 Documentation

When approaching a project in an agile manner, focus lies not on documenting everything, but instead in getting the software being implemented to work as specified, leaving some form of documentation behind, be it in self-describing source code or simple documents explaining patterns and structures used in the application.

The agile manifesto[4] states, "[w]orking software [is chosen] over comprehensive documentation"; this is not to say that documentation shall be completely omitted, but rather that one should pick working software over comprehensive documentation, if forced to choose.

5.5 Technology

Learning to work with a new programming language is generally not a great feat for a student majoring in computer science and mathematics, but doing so at the same time as one is learning an integrated development environment, new concepts and programming patterns, and at the same time working under much freer terms than usual, it can be quite the obstacle, all things considered.

A Resource Constrained Environment

Working in a resource constrained environment, as today's smart phones certainly can be said to be, was a new and interesting experience for me, not having taken any courses that directly covered event-handling and highly resource-constraint aware programming. Programming in a memory man-

[3] An implementation is said to be pixel perfect when its rendered interface matches up with a design perfectly, at the pixel level.

[4] http://agilemanifesto.org/

aged environment was not a new thing for me, having programmed in C/C++ before the project, but using reference counting[5] was sometimes confusing as the environment allows for automatic retention of objects. It was not always obvious if it was necessary to retain or release objects.

Learning Objective-C

For a developer familiar with the programming languages C, Java and Erlang, just learning Objective-C, with its concepts and features should not be more than a matter of learning the syntax and how concepts between the languages relate to each other. Erlang is useful for understanding how higher-level programming is done in Objective-C since it empowers the developer to do reflection and closures[6].

One of the most significant hurdles in the process of learning Objective-C is realising that Objective-C uses a dynamic runtime, which has some interesting consequences; sending a message to a nil pointer (equivalent to calling a method on a null pointer in java) causes no error. This is due to the fact that a message sent to a nil pointer is discarded as there is no recipient and returns the value nil (actually 0) by default.

It can be likened to putting a completely unmarked envelope in a mailbox, one can't reasonably expect it to arrive anywhere but a post office where it will need to be handled according to predetermined procedures. In Objective-C it is merely dropped, which can sometimes cause unexpected errors if one does not check that either *1)* the recipient of a message is not nil **or;** *2)* the response to a sent message is not nil.

Working With iOS

The iOS platform and software development kit (SDK) have some well thought out and implemented patterns and features which, when properly studied, examined and used, allow for rapid development of a prototype if

[5]Reference counting is a way of keeping track of whether or not an object is being used, upon creation, objects are retained and then again for each time a new object wants to ensure that it will not be destroyed while it is using said object. When the holding object is done with the retainee, it releases it, as the other retaining objects do, finally causing it to be destroyed when the last one has released it.

[6]Reflection and closures are ways that allow programs to use each other and to modify themselves during execution, respectively.

one just uses the components provided by the environment. Many times it is sufficient to create sub-classes of Apples provided classes that implement standard behaviours.

5.6 Future Work

For future work relating to the issues covered in this thesis it is recommended that the following topics be investigated;

- Do students need to be given more experience in open-ended group projects, as characterised in [Daniels11], in order learn how to better prioritise sub-tasks and requirements of undertaken projects? This can, for example, easily be tested with courses of both small and large scale.
- Is the importance of continually honing ones skill in project management and group communication (both intra team and inter team) emphasised enough before each essential piece of course work is undertaken?
- Even though valuable experience will be gained from exercises and projects in courses, students should be given an opportunity to practice on communicating within teams, between teams and in other roles that might arise in their future careers.
- Are students aware of what tools and methods exist in order to alleviate the issues of software development? These could be made part of strongly recommended, or even mandatory, courses.

The results of the suggested enterprises should be investigated in studies of an academic nature, so as to judge their respective efficacy and – both by students and teaching staff – perceived value.

References

[*Daniels*11]

Daniels, Mats. *Developing and Assessing Professional Competencies: a Pipe Dream? Experiences from an Open-Ended Group Learning Environment.* Doctoral dissertation, Uppsala University, Uppsala, 2011.

[*Fielding*00]

Fielding, Roy Thomas. *Architectural Styles and the Design of Network-based Software Architectures.* Doctoral dissertation, University of California, Irvine, 2000.

[*TakeuchiNonaka*86]

Takeuchi, Hirotaka; Nonaka Ikujiro. *The new new product development game.* Harvard Business Review, 1986.