Kamili

a Platform-Independent Framework for Application Development for Smart Phones

Lars Maaløe
Martin Wiboe

July 2011

DTU Informatics
Department of Informatics and Mathematical Modeling

Kamili

a Platform-Independent Framework for
Application Development for Smart Phones

IMM-B.Sc-2011-23

Kongens Lyngby, 2011

By
Lars Maaløe
Martin Wiboe

Published by: Department of Informatics and Mathematical Modelling
Richard Petersens Plads, Building 321
DK-2800 Kgs. Lyngby, Denmark

Request report from: www.imm.dtu.dk

Summary

With smartphone ownership increasing every year, developers are creating mobile applications, or *apps*, at a quicker pace than ever before. Smartphone manufacturers are competing for market share, causing a number of different mobile platforms to coexist. As the number of platforms increases, developers are looking for cross-platform development solutions that allow the same application to run on multiple devices.

In this thesis, we examine the currently available solutions for cross-platform development in the context of implementing a realistic example application. We then develop a tool, Kamili, for rapid prototyping and development of applications that run on several platforms using the Naked Objects method of automatic user interface generation.

We implement the example application and discuss the implications of working with multiple platforms and of using Kamili. We then describe the development process and the testing methods employed, as well as the possibilities for extensions of the tool.

Resumé

Antallet af smartphone-ejere er stigende, og i takt med dette udvikles der flere mobile applikationer, *apps*, end nogensinde før. Smartphone-producenternes konkurrence om markedsandele forårsager, at antallet af forskellige mobilplatforme øges. En effekt af denne udvikling er, at softwareudviklere efterspørger værktøjer, som muliggør udviklingen af mobile applikationer der fungerer på flere platforme.

I denne opgave undersøger vi de eksisterende værktøjer af denne type. Vores analyse tager udgangspunkt i udviklingen af en realistisk eksempelapplikation. Derefter udvikler vi et værktøj, Kamili, som kan anvendes til hurtig udvikling af prototyper og applikationer som kan køre på flere platforme. I dette værktøj anvender vi Naked Objects-metoden til automatisk generering af brugerflader.

Vi implementerer eksempelapplikationen og diskuterer implikationerne af at arbejde med flere mobile platforme og ved anvendelse af Kamili. Endelig redegør vi for den anvendte udviklingsproces og for test-metoden samt for mulighederne der ligger i en fremtidig videreudvikling af Kamili.

Preface

This thesis was prepared at the Department of Informatics and Mathematical Modelling at The Technical University of Denmark in partial fulfillment of the requirements for acquiring a B.Sc. degree in engineering.

Kongens Lyngby, July 2011

Lars Maaløe

Martin Wiboe

Content

1 Introduction

The rate of smartphone adoption is accelerating worldwide. In 2010, Apple had sold more than 60 million iPhones worldwide since its release in 2007. [1] Other smartphone vendors all profit from the expanding demand on the market. [2] This development has had a great impact on the market for mobile apps.[1] [3]

Apple introduced Apps to the iPhone in 2008, bringing increased revenue for smartphone companies. [4] Apps have become a great showcase for companies to share their advertisements, ideas and products to a broad audience. [5] Developers use many of the hardware features that enable a great variety of apps, such as geo-location and camera. In 2010, the total app market reached 450,000 released apps with Google's *Android Market* and Apple's *App Store* holding the largest proportion. [6]

The market for app development services has grown even larger than the app market. Developers have realized that it is more profitable to start an app development business than it is to publish ideas themselves. This has resulted in a high increase of interest in the app creation services. The need for additional app creation services is growing. [7]

According to [8], the term "cross-platform" can be defined as "*the ability of software to operate on more than one platform with identical (or almost identical) functionality.*" While the term "platform" does not have a universally accepted definition, we will use it in the way we have most commonly seen it used: a "mobile platform" means a set of devices that use the same operating system and have similar form factors. Google Android is a platform. So is Apple iOS.

Several mobile platforms exist today, with the most recent major addition, Windows Phone 7 (WP7), launched in October 2010. [9] These new platforms bring both opportunities and challenges to software developers, who must stay on top of recent developments. While it is indeed possible for software developers to gain proficiency in every platform, many are increasingly looking towards cross-platform development tools, as witnessed by the sheer number of such tools available. [10]

[1] Application

1.1 Project Objectives

The following question was the basis of this thesis:

How can one create a mobile application that works on several platforms?

This led us to four sub-question. The structure of this thesis is built around the sub-questions, leading to the conclusion that is an answer to the main question.

What technologies can be used to achieve this?

We address this question in chapter 3 and elaborate on the mobile OS's used in this project. Furthermore, we give a short introduction to the hardware setup and features in the smartphone market today. The end of chapter 3 gives an introduction to the existing cross-platform solutions on the market.

What set of functionality exists across platforms and should be exposed by a cross-platform framework?

Answered in chapter 4. We describe challenges in cross-platform development. The chapter also contains an analysis of approaches to building the UI framework.

What are the advantages and disadvantages of the cross-platform approach?

Chapter 4 provides an examination of this question. We analyze cross-platform advantages and disadvantages.

Design and implement a tool to facilitate cross-platform UI development.

We supply this implementation in chapter 5. A further elaboration on the project goals and requirements is included as well as a conclusion about the technologies used in order to reach the goal. Furthermore we explain the development process which has led to a working implementation of a cross-platform tool. At the end, we test the extensibility of Kamili, and describe our unit test method.

Chapter 2 introduces a case that gives a real life scenario that our tool will hopefully be able to solve. In chapter 6 we solve the case and describe the advantages and disadvantages of the result.

1.2 Project Result & Name

The result of this project should be a fully functional cross-platform tool that is able to facilitate UI generation for at least two mobile platforms. The rest of the underlying code is made portable through the use of third party tools, so that our focus can remain on the UI alone.

The name of our project is *Kamili*, meaning ‘complete’ or ‘exact’ in Swahili, reflecting our vision for it to make mobile applications run on all platforms.

2 Case Introduction

The following is the description of the case from Danish news: [11]

> *Troublemakers in the Danish nightlife are an increasing phenomenon. The Danish government has made a decision to allow pubs and nightclubs to keep a register on troublemakers. This allows the bouncers to keep the troublemakers out of the specific bar or nightclub, and helps keeping a proper behavior throughout the Danish nightlife.*

We add further interpretation of the case:

It has been decided that the implementation of the register shall be digital. This allows the register to keep updated at a frequent rate, and it allows the bouncers to access it easily. It is decided that the implementation should be tested as a mobile application. This application will make it possible for the bouncer to access the register. The committee responsible for the implementation is uncertain about whether a mobile application is the smart choice. Therefore a prototype will be created.

The requirements for the prototype are as follows:

1. The application should be available on at least two mobile platforms.
2. It must be possible for the bouncer to look up the status of a potential troublemaker by entering his civil registration number.
3. Internet access to transfer the personal details back and fourth from the register is required.
4. Due to the fact that this is a prototype, it should be possible to change the features easily. The development should be as quick as possible.

3 Background

3.1 Focus Platforms

The design of Kamili emphasizes reusability and the ability to relatively easily add new target platforms. For this report, we have decided to implement the ability to target 2 platforms: iOS and WP7.

The following subchapters briefly describe each platform and the advantages and disadvantages for developers targeting the platform.

3.1.1 iOS

First released in early 2007, iOS is an operating system in the Mac OS X family which targets mobile devices. iOS runs on recent versions of the iPhone, iPad and iPod Apple devices. [12]

iOS was significant in popularizing smartphones among consumers, introducing a touch-screen, gesture-based user interface that other companies have been quick to adopt. [13]
With the App Store, introduced in 2008, Apple was among the first companies to offer mobile developers a digital distribution network to sell their software to iOS users. As of 2011, more than 15 billion apps have been downloaded from the App Store, netting developers revenues in excess of $2.5 billion. [14]

For developers, the large iOS community and the high revenue from the App Store are definitely an advantage of targeting this platform. Users of iOS on average are more likely to buy apps than users of other major mobile operating systems such as Google Android or RIM BlackBerry. [15]

Another advantage is the low number of device models that run iOS, meaning that developers do not have to perform testing on many different devices compared to broader operating systems like Android.

To develop for iOS, one must use the Objective-C language, which is far less popular than the .NET family of languages. [16] The iOS development tools can be only used on a Mac – including the Interface Builder tool for drawing a UI, the XCode tool for compiling, debugging and signing and the iPhone emulator. This means that the learning curve for new developers is higher, as is the initial cost of starting development.

Finally, apps for distribution through the App Store are subject to an approval process administered by Apple. There have been reports of apps getting rejected with little justification [17] and of app approval rules being enforced in a seemingly inconsistent manner. [18]
The risk of getting an app rejected from the App Store is a disadvantage of targeting the iOS platform.

3.1.2 Windows Phone 7

Windows Phone 7 is a recently created mobile operating system, released by Microsoft in October 2010. The system, which is based on the Windows CE embedded operating system, represent Microsoft's latest foray into the mobile market, having seen its previous offerings overtaken in the marketplace mainly by iOS and Android. [19]

WP7 is licensed to several phone manufacturers and runs on smartphones. Microsoft does impose some minimum system requirements that licensees must fulfill before being allowed to use the system. [20]

Application developers can distribute their apps in an app store that is similar to Apple's. This store obviously has not yet obtained the same revenue and customer base as the iOS one, which might be seen as a disadvantage. However, the smaller amount of apps in the WP7 app store, along with a continuous flow of new users coming to the platform, could also be seen as an advantage.

The tools for developing for WP7 are Visual Studio 2010 on Windows or MonoDevelop on Mac or Linux. Visual Studio is an effective and pleasant development environment. [21,22]
The version of Visual Studio used for WP7 development can be downloaded free of charge. [23] Developers can utilize the popular .NET framework while programming in the C# language, making the learning curve flatter for many developers. The familiar and affordable development environment is a clear advantage of WP7.

Theoretically, apps run equally well across WP7 devices, but in practice developers will have to test on more devices to ensure a positive experience across different screen sizes, processor speeds etc. The increased device fragmentation is thus a disadvantage of WP7.

3.2 Devices

A mobile platform will typically consist of several devices using the same operating system. While the device does not affect the programming language used to target the platform or the general UI guidelines, there are differences between devices that developers need to be aware of.

3.2.1 Screen Size & Resolution

Screen size and resolution will often vary between devices, and this has caused problems in the desktop world where older programs had the size of common screen elements hardcoded in pixels. If the user wanted to increase text size in such a system, graphics and screen elements of incompatible programs would not scale properly. [24]
The major mobile operating systems all implement some kind of device-independent pixel addressing scheme to assist developers in creating apps that work on different screen sizes, but it is imperative that care is taken to provide graphics in a scalable format etc.

Another example would be the platforms where different form factor devices, such as tablets and smartphones, co-exist, like Android and iOS. Even though applications may run on either form factor, some planning is typically needed to maintain an optimal user experience across form factors.

3.2.2 Hardware features

The hardware capabilities of devices vary in general. Some devices have camera, compass or GPS antenna. Taking Android as an example, some devices will be tablets, some will be smartphones and some will even be TV boxes. [25]

It is up to the developer to let the application check that the required features are present and fail gracefully if they are not there.

3.2.3 Connectivity

Most smartphones have an Internet connection, and many apps will rely on this connectivity to work. However, the connection may be disrupted due to poor signal strength, roaming etc. Mobile applications should be prepared for when no connectivity is available.

3.3 Existing solutions

In the app development industry, there are several cross-compilation and cross-platform tools available. In this section we introduce three solutions that we find interesting.

3.3.1 jQuery Mobile

The purpose of the jQuery Mobile project is to create a cross-platform web user interface. It is sponsored by powerful organizations, such as Nokia, Palm, BlackBerry, Adobe. [26] The technology is built on JavaScript, CSS3 and HTML [27]. This gives the technology the freedom to span over a wide selection of mobile devices. The main vision of the project is:

> *"Delivering top-of-the-line JavaScript in a unified User Interface that works across the most-used smartphone web browsers and tablet form factors." [28]*

jQuery Mobile is built on the most commonly used web standards. All developers of modern browsers, with at least a nominal amount of market share, implement these technologies in their product. The HTML code is the foundation of web documents, and gives the web design structure. CSS defines the layout, color and font setup. The jQuery JavaScript library enables dynamic features to the implementation. The bottom line is that jQuery Mobile has developed a comprehensive platform that enables developers to more easily implement complicated features and usable designs. [29]

The disadvantage of the jQuery Mobile project is that developers are restricted to the features in the libraries. This results in uniform products, which probably will not satisfy the customers that are interested in design or innovate UI. Furthermore, the cross platform capability and the adherence to web standards limit device-specific features.

3.3.2 Titanium Mobile

Titanium Mobile is developed by the company Appcelerator. [30] The idea behind Titanium Mobile is very similar to jQuery Mobile. It gives the developer the freedom to build an application using JavaScript. Thereby the developers do not need to learn the platform-specific languages like Objective-C and Java in order to develop a fully functioning mobile application. Despite the apparent similarity with jQuery Mobile, Titanium Mobile uses a much different technology to deploy the application to a specific mobile device. Where jQuery Mobile uses the browser technologies within the devices, Titanium Mobile compiles parts of the implementation into the native platform independent languages. [31]

Titanium analyzes and preprocesses the JavaScript, CSS and HTML code. This is followed by a pre-compilation, which separates the code into a hierarchy of symbols. The symbols are decided upon the applications use of Titanium APIs. The symbol hierarchy is the foundation of the generation of a symbol dependency matrix, which maps to the underlying Titanium library. This is done in order to understand which APIs the application needs. The files compiled to platform-specific object files. When the compilation of the dependency matrix is complete, the native SDK compilers are invoked in order to compile the application to the final native library [32].

Where jQuery Mobile uses the browser engine for the application, Titanium Mobile uses the hardware in the mobile device. Using hardware acceleration from native platform independent code gives a much faster feel to the application. Titanium Mobile has some disadvantages, namely that it is not possible to compile all of the JavaScript code into native code. Some of the dynamic code runs through an interpreter, which slows down the application compared to an application implemented in pure native code. [31]

3.3.3 MonoTouch and Mono for Android

Mono is an open source project that enables developers to create C# and .NET development on non-Windows operating systems. Mono is available for Mac OS X, Linux, BSD etc. [33].

Novell developed MonoTouch, which is similar to the Mono framework. It allows developers to develop iPhone applications using C# and .NET. The MonoTouch applications are compiled to machine code on the iPhone. [33]

Novell also developed Mono for Android, built on the Mono framework. This enables developers to develop C# and .NET applications for the Android platform. Contrary to MonoTouch, Mono for Android does not compile to machine code. It deploys the Mono runtime to the specific device.

MonoTouch and Mono for Android do not support compilation of the platform specific UI. In MonoTouch the UI is developed in Apple’s own software which creates a XML file in the format .xib. In Mono for Android the UI is developed in a XML file in the *axml* format. This fact prevents the same Mono application from running on both platforms.

3.4 Introduction to Naked Objects

This section gives a general introduction to the concept of Naked Objects. This theory will be used later in the thesis.

Naked Objects was first presented on the annual ACM[2] conference OOPSLA[3] in Seattle 2001. The designers Richard Pawson and Robert Mathews of CSC’s Research Services proclaimed that they had invented a new approach to designing a UI. [34] The main idea was quite simple. They eliminated the implementation of user interface design altogether, by creating objects that corresponds to the business objects. A Naked Objects framework then creates the UI from the implementation of software objects. The developers’ OOP layout will be “naked” to the users, hence the two entities will be in a strong correlation, because they see the same programmatic structure. [34]

Naked Objects is a counterpart to the Model-View-Controller pattern that is very common within the software development society. The main idea of the MVC pattern is that it separates the modules of the program. This makes the program more agile. For example if new views are to be implemented, then there are only two modules that need to be modified, View and Controller, and so forth. Richard Pawson says that the MVC pattern is so common, that it never is questioned or challenged. [35] Furthermore, he says that the MVC pattern holds some shortcomings that Naked Objects can handle [35] .

In Figure 1 the difference between a regular MVC pattern and a Naked Object pattern is presented. First of all, the controller layer is removed

[2] Association for Computing Machines

[3] Object-Oriented Programming, Systems, Languages and Applications

in the Naked Objects implementation. This result in a big decrease in the developers workload because of one less layer to implement. Furthermore, the figure shows that the presentation layer in the Naked Objects implementation gives a direct view of the domain objects, whereas the presentation layer in the MVC pattern has a fully functional individual presentation layer. Again the developer's workload decreases substantially by not having to implement the individual presentation layer.

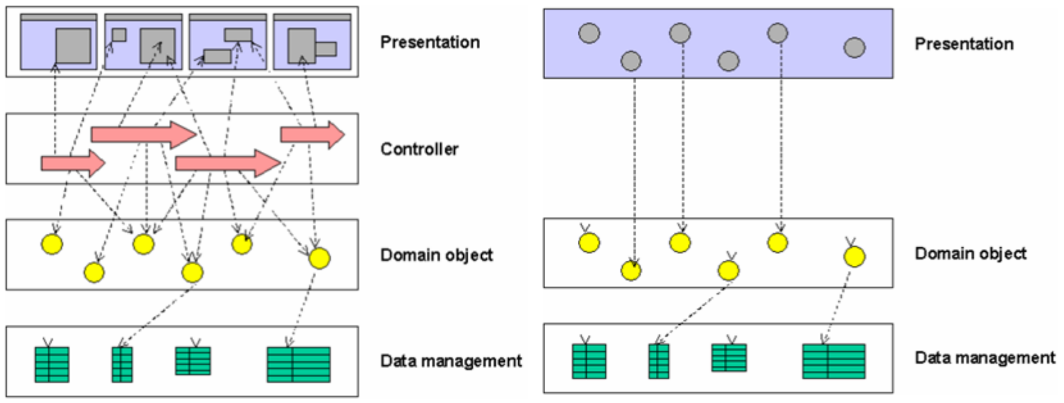


Figure 1: The MVC pattern with data management vs. The Naked Object pattern with data management [35]

Using Naked Objects has advantages and disadvantages. From a development perspective, it decreases the development cycle, because the domain objects carry the entire implementation. Furthermore it generates a common language between developers and users, which results in a better understanding amongst the two entities [35]. It also minimizes the amount of maintenance, and there is no risk that the user or developers level run out of sync [34]. The disadvantages of Naked Objects are mostly on the usability constraints. The logic of a software developer does not always respond to the logic of an end-user. Furthermore the one-size-fits-all premise of Naked Objects does not always implement the unique aspects of a UI [34].

4 Analysis

There are several aspects of creating apps that work on multiple platforms. Some are of a purely technical character, like how to use native APIs in portable code, and some are of a “softer” kind, like how to maintain the proper look and feel across platforms.

In this chapter, we address the most important aspects and try to relate these to some of the cross-platform solutions that exist today. We will also be examining different approaches to, and the advantages of, constructing our own cross-platform solution. Our analysis is based on the example case from [reference til kapitel om case].

4.1 Techniques for Cross-Platform Development

To develop native smartphone apps for a platform, it is necessary to know the supported programming language or instruction set. If the programming language is known, one can use the "official" developer tools to produce the application. If the lower-level instruction set is known, one can write in any programming language and create a custom compiler that targets the platform. However, the binary instruction set for a platform is not necessarily publicly documented. Additionally, creating a production-level compiler will take effort that is beyond most developers.

Another problem when developing a native app is the use of the platform-specific Application Programming Interface (API). The API is the set of methods that the operating system exposes to applications. To perform any non-trivial action, for example manipulating the UI or using the geolocation hardware, one must invoke methods in the APIs. APIs vary wildly across platforms, so it is necessary to create some kind of wrapper before it is possible to write portable code.

There are several techniques for writing cross-platform applications. Cross-compilation, interpretation and web technologies are some of the options. The following subsections list some common techniques along with their advantages and disadvantages.

4.1.1 Cross-compilation

Cross-compilation is the process of generating native code to run on multiple platforms. An application created using this technique is indistinguishable from one written specifically for the platform.

Cross-compilation is difficult since it involves compiler design and low-level optimization. On the other hand, this technique offers the best performance when implemented correctly. Cross-compilation is typically used with statically linking wrapper libraries, which may increase application size. It is necessary to re-compile the application for each new platform. Likewise, one must re-compile to update the linked library.

This technique has the advantage of being unencumbered by legal issues, since it produces "real" native applications. Apple, for example, does not currently allow interpreted applications to be distributed on iOS devices [36]
The MonoTouch framework is an example of pure cross-compilation and static linking. [37,38]

4.1.2 Interpretation

To achieve cross-platform compatibility using interpretation, one compiles the application to an intermediate format, which is then executed by a dedicated program, the interpreter, on the target platform. Interpretation is deployed in cross-platform desktop application frameworks, for example Java [39] and most Android applications are developed and distributed in an intermediate form for running in the Dalvik virtual machine on the device.

Interpreted applications have the advantage of not requiring recompilation for new platforms. To add a new platform, one must simply develop an interpreter for that platform. In addition to not requiring recompilation of applications, it is also easier to develop a new interpreter than it is develop a new native compiler, since the interpreter does not have to perform code analysis and optimization. Since the interpreter will necessarily include the API wrapper library, this can also be updated without recompilation.

Additionally. interpreted applications will be smaller than statically linked native applications because they do not have to include the wrapper library.

Compared to native applications, interpreted programs generally have worse performance, though this is highly dependent on the interpreter design. [40]

Performance can be improved with on-platform compilation, so that parts of the code are not interpreted but rather executed natively. The decision about what parts of the program to compile can be made before beginning execution by performing static code analysis, or it can be done at run-time by compiling the most frequently executed parts of the code. The latter technique is referred to as Just-In-Time (JIT) compilation. A combination of both techniques is implemented in major frameworks. However, implementing these features takes significant effort — the JIT compiler for Java, called HotSpot, consists of almost 250,000 lines of code. [41]

The Titanium Mobile framework combines cross-compilation and interpretation. [32]

4.1.3 Web Technologies

A rather new option is that applications can be developed using only open web standards. Applications developed in this way are executed by the Internet browser on the device. Web technologies represent a serious development towards having a common, standardized feature set available across devices in both desktop and mobile platforms. Additionally, the current web technologies have been designed for use on different screen sizes and include accessibility features for use by disabled people.

In practice, the implementation of web standards has been varying across platforms - forcing developers to incorporate work-arounds in their code or to target the lowest common denominator. However, most mobile devices today contain reasonably advanced browsers with generally good support for standards. Web frameworks such as jQuery aim to provide a completely uniform set of features across browsers. [42]

Most of the points about interpreted programs apply to web applications. In addition, the web is inherently based on a client-server paradigm which may not fit all application types. With mobile devices, a further problem arises: connectivity is often sporadic, and cannot be relied upon. This makes client-server applications more difficult to use on such devices.

As mentioned previous, to use the native features of the mobile platform (such as geolocation, compass, proximity sensors, camera etc.), this functionality must be exposed through the interpreter API. With web applications, the interpreter is the browser, so the functionality will have to be included in a web standard. This makes it difficult to use the latest functionality.

The jQuery Mobile framework allows for the creation of pure mobile web applications. PhoneGap allows one to create a web application and bundle it with a native application stub that includes an embedded browser. This allows the application to behave more like a "real" native application.

4.2 User Experience

Along with low-level technical difficulties within cross-platform development tools, there are several other challenges. One is the "look & feel,” which has to comply with the target platform.

This includes using the native widgets (such as buttons, menus etc.), adhering to UI guidelines in general and communicating with the user through platform-specific mechanisms. For example, an application might provide detailed notifications for iOS and a "home screen widget" for Android.

4.3 Implementing the Case Using an Existing Solution

4.3.1 jQuery Mobile

jQuery Mobile can be used to develop the mobile user interface of a web application. To use this solution for implementing the case application, one would implement the business logic in some web application language like PHP or ASP.NET. Then a user interface must be built using HTML and CSS with jQuery Mobile. The finished solution would be served from a web server.

For applications developed in this way, there is a clear distinction between server-side code and client-side code, which is run on the device. Since the product would run in a browser, the client-side code will be JavaScript. The server-side code would typically be PHP or C#. To implement an operation in the application, server-side code for serving the request and presenting the result must be implemented as well as client-side code to launch the request. In summary, the jQuery Mobile approach does require the use of several distinct languages and technologies.

As stated in the case description, it is anticipated that the application changes regularly during development. A web application is well-suited to address this requirement, since any updates deployed to the web server will automatically take effect for all users and devices – there is no need to publish an update through an app store.

Web applications are dependent on Internet connectivity for their use, making them unsuited for applications that should be used offline like text editors, dictionaries, calculators or games.

4.3.2 Titanium Mobile

Titanium Mobile provides the ability to write an application using JavaScript and compile it to run on several mobile platforms. To use this for implementing the case application, one would design the desired user interface and implement it using the Titanium Mobile API.

As with jQuery Mobile, it would be necessary to create a server application for servicing the “Bully look-up” requests from the application. However, contrary to the jQuery approach, the server need not be concerned with generating markup for presenting the result. Thus the server can be a simple service which communicates in JSON, XML or the like. As in the case of jQuery Mobile, the server will probably be implemented using a language other than JavaScript.

4.3.3 Disadvantages of Existing Solutions

Even though both tools are well-suited for cross-platform mobile development, they might not be the best possible choices for implementing the case application, which is a prototype that has a need for rapidly adapting to change. In particular, both tools place emphasis on separating the logic from the user interface, necessitating the use of more than one programming language and requiring more effort when implementing changes in the early phase of the test application development.

It would be desirable to have a tool which allowed for rapidly implementing changes across the business logic and the user interface, without requiring separate effort for those areas.

Additionally, it is desirable to be able to use the same programming language for server and client code. While JavaScript servers do exist, they are not commonly used. [43]

4.4 Creating our own UI Framework

This section examines the options available for creating a framework that allows for easier, more agile user interface development.

4.4.1 Cross-Platform Markup Translated to Device UI

To make it possible to target multiple platforms, an abstraction markup language could be allowed. This language should then allow developers to specify the desired interface in a platform-independent manner. Such markup languages already exist, most notably for use with Java [44,45]

A defining feature of our markup language would be simplicity, to facilitate easy changes.

When compiling for a specific platform, the UI markup would be translated to platform-specific UI code. We believe the benefit of this system to be that it allows the developer a relatively high degree of freedom while still providing cross-platform functionality. However, designing such a system would require significant design effort because a common set of functionality must be identified and wrapped into an intuitive abstraction. Additionally, we expect that the risk of breaking compatibility would quickly increase with the flexibility of the language.

4.4.2 Auto-generated UI: Naked Objects

Rather than provide a comfortable abstraction across platforms, one can simply decide to do away with UI design and have the UI generated automatically based on the domain objects in the application. When using this approach, developers simply develop their application in a strictly object-oriented fashion (and possibly subject to some restrictions) and have a code-generator output platform-specific UI code that allows for manipulating the business objects. [46]

The benefit of this approach is a truly rapid development process, since UI design does not factor into the process at all. [46]

We estimate that adapting Naked Objects for the framework would result in increased flexibility and less resources needed to adapt the application to changing requirements.

5 Discussion

5.1 Project Goals

The main objective of this project is to implement a code generation tool named Kamili. Kamili takes C# code as input and outputs an auto-generated UI code that is compatible on a specific mobile device. The satisfactory result is for Kamili to be able to generate code for two incomparable mobile platforms – WP7 and iOS. Kamili will also generate plumbing code to enable the UI to manipulate the domain objects. Kamili only generates device-specific UI and plumbing code – it is assumed that the target platform can execute C# code.

5.2 Technologies

This section describes the technological choices made during the project. We will discuss the reason for the decision as well as the implications of a particular choice.

5.2.1 Using Naked Objects

After performing the analysis of possible approaches to our own UI framework, we see that the approach of using a declarative, markup-based user interface definition (see 4.4.1) is a real alternative to the existing solutions that were examined. But we also believe that the benefits of using this approach are small compared to using the existing solutions today. We find the declarative approach to be useful in platform-specific solutions, because the markup elements can correspond directly to on-screen controls. This is in fact the design mechanism for the three major platforms. However, this approach means that UI and business logic code are still separate and that both require changes whenever the application is modified.

On the other hand, we have come to see Naked Objects as an interesting and potentially powerful new way of constructing user interfaces. As alluded to in the analysis, we especially see the use of Naked Objects as an improvement of the prototyping abilities of Kamili, making it possible for developers to create functional applications rapidly.

Another benefit is that using the Naked Objects method will hopefully encourage developers to use object-oriented techniques more consistently than before, since the user interface will now reflect the actual design of the application.

There are, of course, also weaknesses to using the method. We estimate that Naked Objects will never be a replacement to the custom-made user interfaces of today, since consumers have come to expect beautiful applications with innovative user interfaces and clever design. The automatically generated user interfaces of Naked Objects might be acceptable for prototypes, business applications or custom-built functionality where aesthetics are not a primary concern.

Ultimately, we have decided to use Naked Objects as the user interface design method in Kamili.

5.2.2 Using C#

The choice of programming language is an ever ongoing discussion, with proponents of each language endlessly asserting its superiority over other languages. Our choice of language was made after the decision to use the Naked Objects method, and as such we sought a language that was object-oriented and statically typed. Both Java and C# conform to these requirements, and both group members had used either language.

While we must both admit to having a personal preference of C#, mainly due its elegant syntax for lambda expressions and the pace of language updates and additions, we felt it was important to consider the suitability of the language in mobile development.

The Java language is already used for developing native Android applications, making it an obvious choice for that platform. However, the requirement was for the language to be usable on multiple platforms, which meant that we would have to devise a way to get Java code running on another major platform. This can be done, but it would be an enormous task when coupled together with implementing Naked Objects for mobile development. As such, we saw Java as being a suboptimal choice for the framework.

The C# language, too, is used natively only on a single platform, WP7, for development with a subset of the .NET Framework. However, Novell Inc. is the owner of the MonoTouch and Mono for Android projects (as described in 3.3.3), meaning that it would at least be possible to use the same syntax across the platforms. The Mono projects allows for the sharing of business logic, but not UI code, meaning that our project would be relevant here. Finally, Mono does implement a subset of the .NET Framework, meaning that not only language syntax, but also platform features, could be used across the three major operating systems.

By using C#, it will also be possible for developers to focus on a single language for both the client and server implementations. We believe that this will improve the development process and encourage code reuse by allowing sharing of types between the server and the client.

Based on the discussion above, we decided to use C# as the framework language for Kamili. We also use C# to implement the Kamili generators.

5.2.3 Using MonoTouch and Windows Phone 7

To demonstrate the feasibility of our approach, we will implement functionality to target two different platforms in Kamili. The immediate options for this is iOS (with MonoTouch), Android (with Mono for Android) and Windows Phone 7 (with its native support for C#.)

We regarded it as most interesting to have target platforms of entirely different families, meaning that we decided early on to include WP7 as a target.

After that we had to decide whether to target MonoTouch or Mono for Android. There would be little difference for developers, since the two systems permit reuse of non-UI code between them. However, despite the similar names, the two system are technically quite different with MonoTouch compiling C# code into a native iOS binary and Mono for Android using interpretation and requiring that the Mono runtime be present on the target device. While both approaches are interesting and viable, we found MonoTouch to be a more mature product, with better support for debugging. We also like the idea of being able to produce native iOS binaries.

In conclusion, Kamili will allow developers to target WP7 and iOS using MonoTouch.

5.2.4 Using the NRefactory Parser

Seeing as the Naked Objects method generates user interfaces from source code, it quickly became apparent that we would need a parser in order to process the source code programmatically.

Parsing is an important subject in computer science, both because parsers and grammar have a solid mathematical background and because parsing is the foundation of compilation. We had been using ANTLR to generate a simple parser before, so we naturally considered implementing a C# parser ourselves.

In the end, we decided that the subject of creating a parser was beyond the scope of this report, and that it would add little value. Therefore, we have used a parser called NRefactory, part of the SharpDevelop open source project. The parser is implemented in C# and it will generate an object graph from a source code file.

5.3 Development Process

This chapter explains our development process and the software design. We explain the reason behind any significant changes made.

Being just two group members, our communication and coordination needs did not call for a very formal development methodology - these needs could be addressed on an ad hoc basis. We have had good experiences with following the agile approach to software development before, and so we settled on a method with emphasis on **working software** and ease of **responding to change**. These two key parameters are pillars of the agile set of development methods, as outlined in the Agile Manifesto. [47]

To maintain momentum, we followed a loose iterative process with the end of each iteration coinciding with a status meeting - some of these with attendance of the project advisor.

The free version control system Subversion was used for keeping code history and for synchronizing any work done separately.

5.3.1 Implementation Order

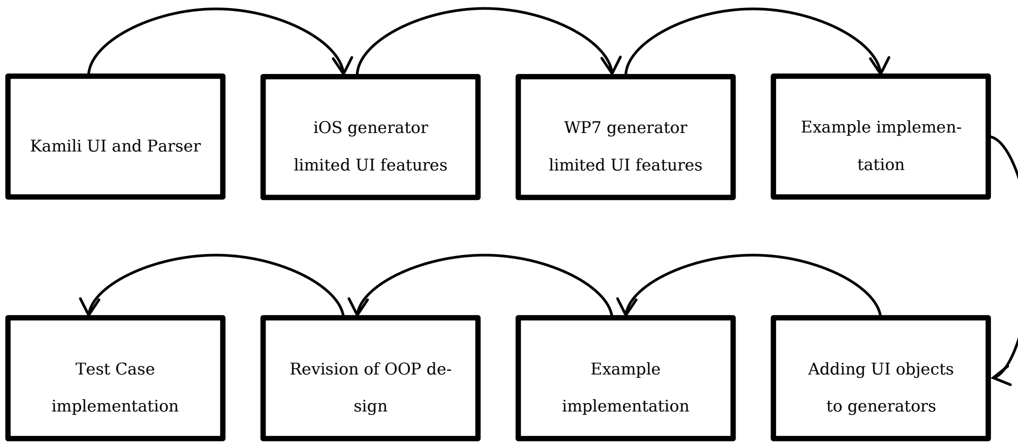
We decided to target iOS for our first working prototype. The goal was to have a working generator for iOS and a sample application demonstrating trivial functionality. In order to achieve this goal, quite a bit of reverse engineering had to be done since we were unable to find any detailed documentation of Apple's Interface Builder file format. The files are XML, but the meaning of the different elements had to be discovered by using Interface Builder to manipulate the file and then examining changes in the file.

We completed this task by joint effort through using the pair programming technique. [48] After a few days, we had a functional program that could generate an iOS user interface from a C# class.

After completing the iOS prototype, we started work on the WP7 generator. It was important to demonstrate that our technique would actually achieve cross-platform compatibility. Developing for WP7 is somewhat similar to developing for iOS since the UI specification done in XML on both platforms. When working with WP7, the markup language used is called XAML. The format is intuitively structured, and the tags are well-documented. [49]

Thus, no reverse-engineering was required, and we were able to create a functioning WP7 generator in a short time. Like with the iOS code, our focus here was to create a working product, and we wrote code with little regard to proper object-oriented design techniques.

After demonstrating that our approach is feasible, we turned our attention to constructing a better design that would serve as the basis for the final Kamili generator. The design goals and considerations are listed in section 5.4, and we used a combination of manual refactoring, automatic refactoring with Visual Studio 2010 and rewriting to implement the final design.



In the figure above is an outline of the phases in the development of Kamili. In the phase where we revised the OOP design, we implemented testing (see 5.6.)

5.4 Design & Implementation

5.4.1 Rationale on Design Requirements

The specification of the design was based on a wish to implement a piece of software that enables developers to input an implementation and let the software generate platform-specific apps. During the research process we realized that the amount of work that was needed to implement a fully functional cross-platform compilation tool by far exceeded the time window given for this assignment. This was one of the reasons why we decided upon the use of the MonoTouch framework (see 5.2.3.) With the help of this tool we had the ability, from C# code to generate platform-specific apps for iOS. The MonoTouch framework did not implement the UI, here the developer needed to use the native tools given by the platform provider – in this case Interface Builder. Furthermore we realized that the C# plumbing code[4] used by MonoTouch as opposed to the one used by the WP7 platform were radically different. This led us to the general requirement:

> *Kamili is a cross-platform tool, which is able to facilitate auto-generated UI for the iOS and WP7 platforms.*
> *The output from Kamili, including plumbing code, can be turned into a native application by the platform-specific SDK.*

The MonoTouch framework is used to compile the C# plumbing code to native Objective-C code. WP7 is based on C#, so in this case there is no reason to use third part compilation software.

Now that we had stated the general objective, we needed to define the requirements to the auto-generated UI. Here we decided upon Naked Objects, which is based on an ideology to not let the developer interfere with UI (see 3.4.) This fits our solution perfectly, due to the fact that auto-generated UI can get much more complicated if the developer has no constraints.

The functional design requirements were now stated, which led to the next step in the process – the design requirements of the actual design.

[4] Refers to the model code that controls the behavior of the UI.

The first requirement was that the design should be object-oriented. Kamili is a cross-platform tool. Therefor the extensibility requirements are comprehensive. It must be simple to extend Kamili to support more mobile platforms than two. An Object-Oriented design enables easy expansion and maintenance of the software product. Further extensibility requirements were stated within the UI feature-set. The goal of this project is not to implement a fully commercial cross-platform tool - it is a proof of concept. Therefor the UI features are restricted to the basics. The design of Kamili must be able to easily handle additions to the UI feature set.

The UI of Kamili should enable users to choose the source code and to perform the generation.

5.4.2 System Design

The basic system requirements are fulfilled. There have been lots of challenges in this process and some are still to be solved. In this section we outline the basic components of Kamili; we describe the challenges that we have solved; last but not least we describe the challenges that have not been solved.

5.4.2.1 Basic Components

The basic components of Kamili cover the ability to parse a C# class and process the parsed output into a corresponding XML and C# code representing the UI and the plumbing code. The inputs obtained by Kamili are properties and methods. C# has a concept called property accessors, which is a piece of code containing the executable statements associated with getting or setting the property. [50] Kamili only interprets methods and properties where the access modifiers are stated as public; hence the public methods and properties are the only ones implemented in the UI. The reason for this specific interpretation of the access modifiers are that it must be possible to implement code that is not visible in the UI. Furthermore the UI would not be able to depend on methods or properties - declared different than public - due to the access being denied. In the below code snippet is a C# class from which Kamili can generate a simple app which enables you to throw dice. In this example there is only one property that is not supposed to show in the UI – the random generator. The remaining properties and methods are supposed to obtain different objects in the resulting UI depending on their accessors.

```
public class DiceGame
  {
    public string numberOfDice { get; set; }
    public string die1 { get; private set; }
    public string die2 { get; private set; }
    private Random dice = new Random();
    public void Throw()
    {
      if (numberOfDice.Equals("1"))
      {
        die1 = dice.Next(1, 7).ToString();
        die2 = "";
      }
      else if (numberOfDice.Equals("2"))
      {
        die1 = dice.Next(1, 7).ToString();
        die2 = dice.Next(1, 7).ToString();
      }
      else
      {
        die1 = "Please enter '1' or '2' in  textbox.";
      }
    }
  }
```

When the source file of the Dice Game is applied to Kamili, the parser will run through the code and identify properties and methods. In this example are three properties and one method. Kamili holds an intermediate representation that divides the method into a method-object and the properties into property-objects. The property-objects are distinguished as read-only or read-write, depending on whether the property accessor can be written publicly or not.

The interpretation of the objects - created in the intermediate representation - has many outcomes. First of all we decided that the methods having the *void* return type should be implemented as buttons in the UI. The rationale supporting this decision is that a method is implemented to perform a task. The task needs to be started somehow. We found that a button would be able to start the execution of a task without changing state and without any input. Second, the read-only properties were decided to be a text representation in the UI - therefor the use of labels was applied. Labels can only be written to by the internal code; hence no user can interfere with the value of a label. Last, we defined the read-write properties to be text representations that allow user interference –

therefor textbox-objects were applied. Users will be able to change the value of a textbox, and thereby change the value of the corresponding property.

When the intermediate representation of Kamili has obtained the methods and properties from the source code, the platform generators retrieve the information needed to implement the UI and plumbing code. On the iOS and WP7 platform the UI is interpreted from two different formats of extensible markup language - *.xib* and *.xaml*. The generator loads a platform specific UI template and initializes new markup documents. The markup of the iOS platform requires a definition and type of the objects that are to be represented. Furthermore it requires connections and outlets, defining the action attached to the specific object. These definitions are implemented into the respective positions in the newly created markup document. The document is closed and saved. The markup of the WP7 platform is simpler. It only requires the definitions and types of the objects in order to create the UI. Like the iOS platform, the objects are implemented to the appropriate positions of the document. Afterwards it is closed and saved.

Both platforms needs separate C# plumbing code, which states the actions of the UI. The plumbing code is divided into an instantiation of the source code class, an initialization method, an update method and methods for each UI entity. The initialization method initializes eventhandlers for the buttons and textboxes in the UI. Each time an eventhandler is invoked, the corresponding UI-method is called. This method links to the source code of the application. The update method updates the UI each time a property is changed. Below is a simplified version of the auto-generated plumbing code for the dice game, targeting the WP7 platform. The structure of the plumbing code for the WP7 platform is very similar to the iOS platform. There is essential difference within the C# language that is used, which is the reason for two different implementations of the plumbing code.

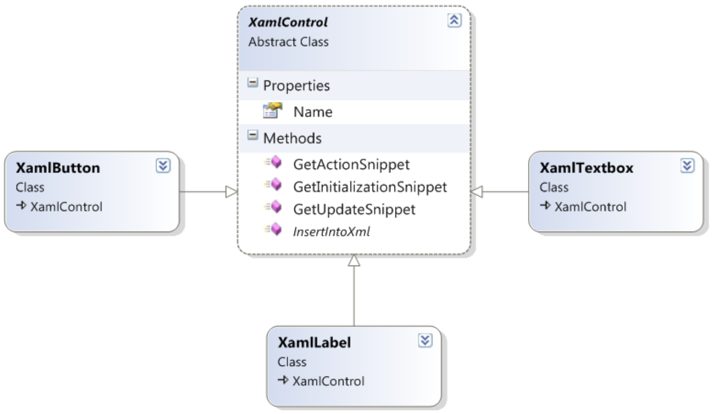
```
public partial class MainPage : PhoneApplicationPage
  {
    // Constructor that initialize and updates
    ...
    private Kamili.Example2.DiceGame theInstance =
                 new Kamili.Example2.DiceGame();
    public void InitializeEventHandlers()
    {
      btnInvokeThrow.Click +=
          new RoutedEventHandler(btnInvokeThrow_Click);
      txtboxnumberOfDice.TextChanged += new

TextChangedEventHandler(txtboxnumberOfDice_TextChanged);
      //Code that implements watermarks
      ...
    }
    public void UpdateUI()
    {
      //Code that implements all UI objects in app
      ...
    }
    void btnInvokeThrow_Click(object sender,
RoutedEventArgs e)
    {
      // Invoke method
      theInstance.Throw();
      // Update UI
      UpdateUI();
    }
    public void txtboxnumberOfDice_TextChanged
                  (object sender, TextChangedEventArgs e)
    {
      // Invoke method
      theInstance.numberOfDice = txtboxnumberOfDice.Text;
      // Update UI
      UpdateUI();
    }
  }
```

5.4.2.2 Solved Challenges

The challenges in developing Kamili have been comprehensive. What objects did the output of the parser represent? How should Kamili handle new generators? How should the dependencies to the parser be implemented? -and so forth. We have handled the challenges, and in this section we will discuss the rationale behind the decisions that comprehends them.

The design of the generators fostered serious consideration. We realized that the implementation of three objects – buttons, labels, textboxes – were far from enough if Kamili should be used for the implementation of commercial apps. This resulted in a vision to implement generators with high extensibility compatibilities. We stated that one entity of the generator was the controls, referring to the objects. Each object should inherit the abstract control definition. This enables a common implementation of the objects specified by the generator. The figure below depicts the architecture of the WP7 generator.



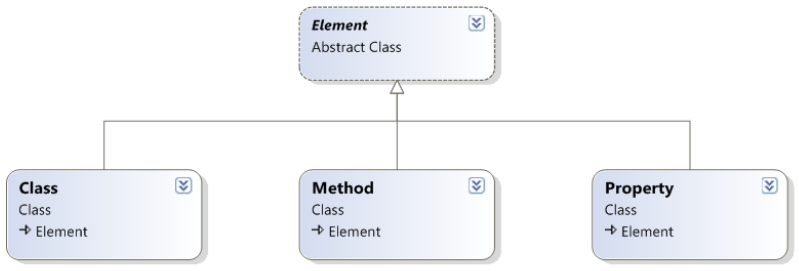
All the methods of the control apply to the object implementation. This specific architecture implements the markup elements and the plumbing code. If a new object is to be implemented, a new class is generated and inheritance of the control-class is enabled. This is a simple way to ensure that the structure of the architecture is kept and code redundancy is reduced. Even more important is that this structure simplifies the implementation of a new object.

The WP7 generator holds three more classes; one to save the platform specific XML file; one to collect and save plumbing code and a class that collects all parts of the generator and defines the objects to be created from the methods and properties given by the intermediate representation.

Besides the creation of an inherited class, the addition of a new type of UI control would also include altering the `WinPhoneGenerator` class, since this class has knowledge of types and corresponding UI controls. The object-oriented architecture could be improved substantially if the delegation of UI elements were moved from the `WinPhoneGenerator` to the corresponding object classes. This would decrease the workload when a new UI control was to be implemented in the generator. Furthermore, it would enforce a stronger object-oriented design.

The overall architecture of the correspondence between the generator and the parser was a big issue. The first version of Kamili included a great deal of dependencies amongst the two elements. We realized that this would result in an issue if the third-party NRefactory parser was updated or replaced by a different improved parser. In this case all the generators would need a comprehensive revision, which would be very time consuming.

The solution to this problem was to implement an intermediate representation which handled the output of the parser. If a new parser was added, the intermediate representation would be the only entity that needed corrections. In the figure below is the structure of the intermediate representation. It enables a categorization of the dependencies amongst the methods, properties and classes. Furthermore the structure is very object-oriented, so if one needs to apply corrections to the interpretation of methods, the only part to revise is the class representing methods.



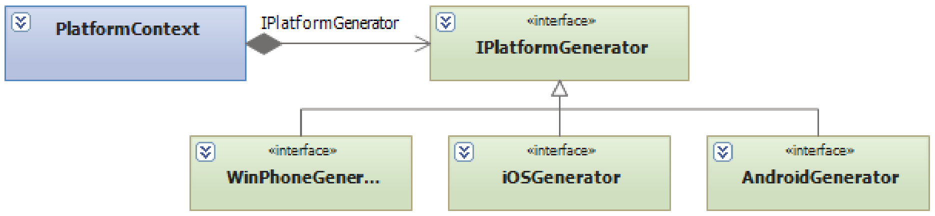
In this project the two target platforms – WP7 and iOS – are a proof of concept. Kamili must have the ability to expose apps to more platforms in order to be a fully functional cross-platform tool. This resulted in design consideration regarding Kamili's ability to target additional platforms. We specified the main features of the design pattern:

1. Generators perform the same task.
2. The algorithms are similar, but implement different actions.
3. No dependencies must exist between the main application and the generators.

The rationale on the requirements was that the generators all implemented a representation of the UI and the underlying plumbing code. Furthermore the algorithms used were assumed to be similar. Last but not least, dependencies between the view and the generators would result in mutual revision when one of the entities was to be changed. To overcome the requirements we decided on using *strategy patterns*. [51] The intent of strategy patterns is:

> *"Define a family of algorithms, encapsulate each one, and make them interchangeable. Strategy lets the algorithm vary independently from clients that use it." [52]*

We copied the structure of the theoretical presentation of strategy patterns into the design of Kamili. See the figure below for a representation with three platforms. The view will only have access to the `PlatformContext` and thereby the dependencies are none. Furthermore the interface structure of the generators declares the overall structure and the similarities.



The two generators implemented in this project are very similar. They implement the UI into markup and the logic into C# code. Therefor the above structure could easily be optimized by collecting similarities between the two generators. This would be a plausible solution to the design if all platforms were generated this way. This is not the case. Some platforms do not separate UI markup and logic, and this would result in a completely different implementation of the generator; hence we cannot enforce a structure on generator internals.

5.4.2.3 Future Options

Kamili is far from ready for commercial use. This section details some possible future improvements and enhancements to the program.

The present layout in Kamili does not reflect real life app-scenarios. Objects like multiple windows, tabs, date-pickers etc. are not supported. Supporting those will require extending the Naked Objects implementation.

To improve the user experience, the object model could be richer. For example, a namespace could be converted into a tabcontrol with each tab representing a class in that namespace. Doing this would allow the user to switch between objects being manipulated. Each namespace could go in a separate view.

The support for property types should be improved, so that for example a `DateTime` property would result in a timepicker control in the UI. Adding this addition requires extending the intermediate representation, so that generators can generate can generate the appropriate code.

At the present state, public methods – represented as buttons – are not able to take parameters in a satisfactory way. If the execution of a meth-od requires a parameter, properties need to be implemented in order for the user to provide input. This solution makes the app unnecessarily complex. To handle the parameters of a method using the Naked Objects terminology a dialog should be prompted, so that the input values could be given. Implementing this feature in Kamili would require a revision of the methods representation in the intermediate layer. Furthermore, each generator would need to support the new feature.

In many apps, controls can be enabled conditionally. E.g. if a form needs to be filled in order to push a button that leads you to the next window, the button should be disabled until the form has been filled. To allow for this in Kamili, it might be possible to let developers decorate a method with a predicate. This predicate should then be true before the method can be executed.

The properties in Kamili only support the object type of `String`. When the user enters a value, there is no way for the framework to know whether this acceptable to the application. For example, the user might be required to enter a valid e-mail address for a string property. One way to implement this would be for the class to validate the e-mail address in the property accessor and raise an exception, causing Kamili to show an error message. A better way would be for the framework to generate better controls based on the desired input type, for example if a property was set as type *int* with read-write privileges, it should be interpreted to a textbox that would only allow integers as input.
However, this approach cannot address custom validation requirements such as that of the e-mail address above. To indicate which values are acceptable to the object, an Exception could be thrown as suggested, but

there might be better for the object to state what input is valid. One such option would be for developers to decorate their properties with *attributes* containing predicates that define a set of acceptable values of the type.

Code Contracts is another approach to solve this problem, because code contracts allow developers to put pre- and post-condition predicates in methods. This information could be used by Kamili to ensure that methods are only called when allowed. [53]

The use of an *abstract class* is common in the .NET framework. A simple example is when a user needs to choose source of payment. In this example the properties of the multiple credit-cards and debit-cards are similar. Therefor an *abstract class* could provide the similarities and inherit them to each payment method. To support this functionality in Kamili the Naked Objects framework needs consideration. A solution is to implement the *abstract class* as a tab and in this tab implement the ability to choose each member class. In the example on payment methods, the tab would be represented and in this tab each payment option should be represented by buttons. They would lead to a new view consisting of the representation of properties and methods in the payment method. Ideally, this functionality should be automatically added whenever a class has a property of abstract type.

5.5 Project File Layout

The project root is the `Kamili` folder. This folder contains the .`sln` solution which can be opened using Visual Studio 2010.

The solution contains several projects, the most important of which is the *Kamili.Converter* project that contains the generator application.

There are also several examples included. The *Kamili.Example4* project is the example case application from this thesis, and the *Kamili.Example4.Server* contains the server implementation.

The automated tests can be found in the *KamiliTest* project.

5.6 Testing

5.6.1 Methods

In a software development process it is of great importance to implement testing. If tests are implemented correctly, they provide knowledge about errors and weaknesses of the implementation. There are many methods and theories that can be used in order to test a system. Several test levels has been processed while implementing Kamili. The test approaches used are divided into white-box testing and black-box testing. First we will explain the processes undertaken by white-box testing.

> *"White-box testing: Testing that takes into account the internal mechanism of a system or component..." [54]*

The white-box testing processes have been our main testing priority in the development process. Each time new entities and classes has been implemented, they have been thoroughly investigated. We have made some decisions regarding the code coverage[5] trade-offs though, which we will explain later. The testing methods used in the white-box testing process are unit tests and integration tests.

Unit testing is a common procedure to verify the validity of the smallest units in an implementation. Each unit is isolated and proven valid or invalid. [55] Unit testing also gives an understanding of the functionality given by a specific unit. This gives the programmer a powerful tool to retrieve understanding of functionality quickly. If a programmer were to understand the specific code without the use of unit tests, it can be quite time consuming due to object-oriented software having large levels of dependencies. This also concludes that the proper way of unit testing, is to avoid dependencies to other classes than the one tested.

Integration testing widens the perspective. Modules that has been undergoing unit testing are combined, and the dependencies amongst them are tested. [54] The result is giving the programmer an evaluation that elaborates on the validity of the interaction between components. Due to its foundation, integration testing is very dependent on the unit tests being implemented properly.

> *"Black-box testing: Testing that ignores the internal mechanism of a system or component..." [54]*

[5] Refers to the degree of white-box testing on the specific implementation.

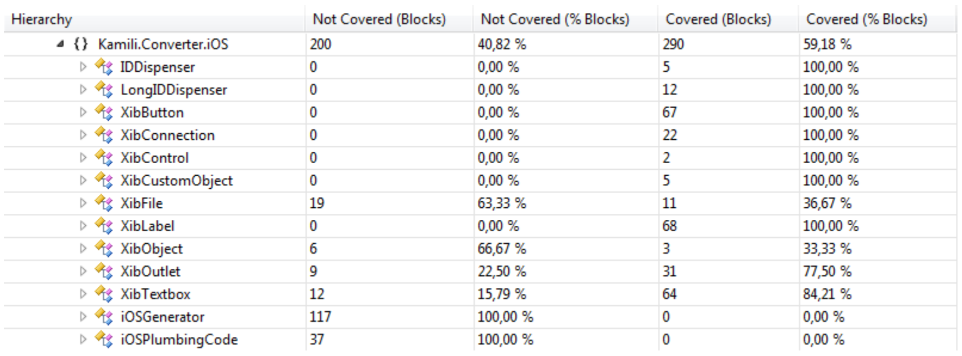
The black-box testing processes has been used solely to conclude whether a given functionality lives up to the functional- and system requirement specifications. The testing method used to comprehend the requirements is system tests. Without any understanding of the underlying dependencies and implementation limitations, the testing process tries out all requirements specifications. If the implementation passes all the tests, a good approach of the system test, is to test beyond the requirements in order to acquire an understanding of the limitations.

5.6.2 Process

The unit testing process has been implemented using Microsoft Visual Studio’s unit testing framework. This allows the unit test classes to be auto-generated and each method to be initialized. With the frameworks assertion method, it is possible to state the expected output, and compare it to the actual output.

It has been a major concern, not to implement unit tests that have dependencies to other entities in the implementation. This concern has been difficult to follow, due to object-oriented design having many links throughout the solution. The unit tests with many dependencies have been implemented including new instantiations of the needed parameters. This has been very time consuming, and it has led to a trade-off decision.

The decision was regarding the amount of code coverage in the project. We decided that the major testing concerns were the platform generators. These are by far the most complex objects of the solution and therefor also the most likely foundation of errors. The first object that was implemented was the iOS generator. We tried to follow the test driven development recipe by creating the unit tests before the implementation of methods and classes. This we found to be extremely difficult because of sudden new additions of modifications to the pre-defined methods in the development process. This ended in modifications of the unit tests in almost every development cycle. Furthermore the need for unit tests in specific cases was re-considered. Because of the time consumption we had to pick our battles. Thereby the code coverage of the iOS and the WP7 generator has been limited to the extent that we find essential to test. In the following figure are the code coverage results from the iOS generator. At the bottom of the figure, two classes have not had any unit testing. This decision is based on the two classes’ use of the other classes. They do not provide any generation by them self. Therefore we did not find it essential to test them, because the methods they use have been tested.



Hierarchy	Not Covered (Blocks)	Not Covered (% Blocks)	Covered (Blocks)	Covered (% Blocks)
{} Kamili.Converter.iOS	200	40,82 %	290	59,18 %
IDDispenser	0	0,00 %	5	100,00 %
LongIDDispenser	0	0,00 %	12	100,00 %
XibButton	0	0,00 %	67	100,00 %
XibConnection	0	0,00 %	22	100,00 %
XibControl	0	0,00 %	2	100,00 %
XibCustomObject	0	0,00 %	5	100,00 %
XibFile	19	63,33 %	11	36,67 %
XibLabel	0	0,00 %	68	100,00 %
XibObject	6	66,67 %	3	33,33 %
XibOutlet	9	22,50 %	31	77,50 %
XibTextbox	12	15,79 %	64	84,21 %
iOSGenerator	117	100,00 %	0	0,00 %
iOSPlumbingCode	37	100,00 %	0	0,00 %

The integration testing has been implemented using manual testing approaches to the project. Each time a development cycle ended we checked the functionality of the independent paradigm. There has been a lack of documentation of the individual functional tests, which eliminates re-use and the possibility for other programmers to get a quick understanding of dependencies in the implementation. Despite the lacking documentation the integration tests gave us an impression, whether functionality was valid or not. We decided to make integration test only on complex matters. Again this was a trade-off on the time consumption.

The system tests have been used to verify the output files of the generators. We implemented different example implementations and ran them through the generator. This enabled us to see the output files generated on different scenarios. We used the output files to generate iOS and WP7 projects, and tested the functionalities of the resulting apps. The functionality requirements of the individual app were pre-defined and at the end of a system test, we concluded whether the outcome justified the requirements. System tests have also been used to test the UI of the generator; to see whether it acted as expected in different contexts.

5.6.3 Results

The resulting code coverage could have been much more extensive. We made some interesting findings in the generated testing environment though. This definitely resulted in the implementation to be of much higher quality than if no testing were executed. If more testing was implemented it is very likely that the resulting software would be even more bullet proof. The trade-off has its justifications though. This project is a proof of concept; hence the purpose is not to sell the software as a commercial product. If this was the case, the necessity of testing would be much more essential. It is important to underline though, that the implementation of unit tests are much easier to do during the development process. If this project was to be commercialized, it might be worthwhile to implement the rest of the testing.

6 Implementing the Case Application

We have been evaluating development tools and techniques from the perspective of having to quickly implement a simple, but non-trivial, mobile application. As described in [reference: intro to case], the purpose of the application is to allow employees at night clubs to quickly check guests against a central "troublemaker registry." By having this ability, club owners can decrease the number of problematic guests in their establishments.

This chapter describes the design and implementation of the case application which we will refer to as *BullyReg*. We will explore the design decisions and tradeoffs that were made in order to create the application using Kamili.

Throughout this chapter, we will only focus on the client application. It is assumed that a server implementation exists which can be used by our application. We have indeed created such a sample server implementation, and it is included in the project source code.

6.1 Design Requirements

The most basic functionality is that BullyReg should take a civil registration (CPR) number as input and return information on

1. whether the number is valid
2. if so, the name of the person it belongs to.
3. the “bully status” of the person: a Boolean value indicating whether this person has been known to cause trouble before.

6.1.1 Server Communication

The database is centralized, meaning that BullyReg will communicate with a central server to retrieve the information. Thus, some data exchange format and protocol must be decided upon.

One option is to use Microsoft Windows Communication Foundation (WCF), which is an API for inter-process communication and building service-oriented applications that can be used through various protocols and data formats. [56]
With WCF, it is possible to share types (known as Data Contracts) be-

tween the server and client applications, and automatically generate code to call methods and pass objects across the wire. [57]
The disadvantage of WCF is the additional complexity introduced by including it in the solution, and the fact that the automatically generated code has to be generated manually for use in MonoTouch on iOS. [58]

To avoid these disadvantages, we will implement the functionality ourselves rather than use WCF.

The .NET framework has extensive native support for processing XML data [59], and XML is a standardized method for data transfer, making it a good choice for implementing the communication in BullyReg.

One issue makes XML less than ideal for use on a mobile device, namely its rather large overhead and inefficient encoding. The W3C has recommended a compact XML alternative designed for use on mobile devices, the EXI Format. [60]

We will only be exchanging small amounts of data with the server, so the efficiency issue is less important. Due to the ease of working with XML, we will use this format in BullyReg.

Our application will be using HTTP requests for exchanging the XML data with the server since this is both suitable for our purposes and the only protocol supported by Windows Phone. [53]

6.1.1.1 Format Specification

Assuming that the server resides at example.com, the application will perform the look-up by executing a HTTP GET request for

```
http://example.com/DoLookup.ashx?cpr={cpr}
```

where `{cpr}` denotes the civil registration number for which to perform the look-up. The number must be a string of 10 numerical digits, optionally with a hyphen (-) inserted after the 6th digit.

After retrieving the data, the server will return the result as an XML document similar to the following

```
<?xml version="1.0" ?>
<result>
      <name>John Smith</name>
      <cpr>1234567890</cpr>
      <knownbully>no</knownbully>
```

```
        <cprvalid>yes</cprvalid>
</result>
```

Note that if `cprvalid` is “no” then name and `knownbully` will be empty. `cprvalid` is “no” if the provided civil registration number is not valid.

6.1.2 Application Design

Due to our prototype implementation of the Naked Objects technique, the entire application must be contained in a single class. This class will contain the functionality to accept civil registration number input, perform the server query and display the result of the query.

We create a `BullyReg` class, which has a read/write string property `CPR` for entering the civil registration number for the current request.

The `BullyReg` class also has properties `Name` and `BullyStatus`, both of type `string`, to display the results of the latest query. These two properties have private setters, meaning that they cannot be altered through the UI but only from a method in the class itself.
Finally, there is also a void method `CheckStatus` for performing the server request.

It will be possible to encapsulate the desired functionality in this class. The auto-generated UI will allow the user to enter the civil registration number and then press a button to perform the request.

6.2 Implementation

6.2.1 Using a Shared Subset of Types

The implementation of BullyReg is relatively forward, as the functionality will to a large extent be provided by the built-in types of the framework. However, a general problem in creating an application with Kamili is to make sure to only use the subset of the framework classes that are available on every platform targeted.

We first implemented BullyReg using the `XmlDocument` class for working with the XML, but this class is not available on Windows Phone 7, meaning that we had to use `XDocument` instead. This class is supported on both iOS and Windows Phone 7.
It is fortunate that `XDocument` can be used on both platforms — had this not been the case, it would have been necessary to implement the XML parsing ourselves or use a different format.

To perform the HTTP requests, we used the `WebClient` class, which is a simplified wrapper around the low-level `WebRequest` class. [24] This class is available on all platforms, and it exposes a method `DownloadString` which retrieves a text string over HTTP.

The `DownloadString` method is blocking, meaning that calls to it block the application thread until the method has completed running. Since network operations are slow and unpredictable, this is a problem if the method is called from the application UI thread. If this is the case, the user will experience a complete "freeze" of the application until the method completes.

Presumably to avoid this issue, Microsoft has not made the `DownloadString` method available on WP7. Developers must use the `OpenReadAsync` method instead. This method launches a background download thread and then returns immediately. When the download is complete, an event handler is executed.

6.2.2 Multi-threaded Execution

Using the `OpenReadAsync` method makes the application multi-threaded. Multi-threading has numerous advantages, especially on modern hardware where multiple CPUs can execute code in parallel, but developers also need to be aware of a number of issues when writing multi-threaded applications. [61]
We will not describe these issues thoroughly here, but one is that of "synchronizing" threads; when different bits of application code run in parallel, the execution order of instructions cannot be guaranteed. For example, an application may have one thread that computes the result of some calculation, and another thread to display the result to the user. Since these operations run in parallel, synchronization is needed here to make sure that the display thread does not try to display the result before the computation is finished. Such synchronization can be implemented in a variety of ways, e.g. by the use of semaphores. [62]

For applications generated with Kamili, a button is shown for every method in the source class. When the user presses this button, the method is executed on the class instance and the UI is updated. In this case, the `CheckStatus` method launches the download thread and executes immediately, causing the UI to be updated prematurely. When the download thread has retrieved the look-up result, no UI update is performed.

The following subsections describe possible solutions for this problem along with the benefits and disadvantages of each solution.

6.2.2.1 Solution: Adding a “Refresh UI” button

Application developers could simply implement an empty `RefreshUI` method in their class. When users click this button, the framework will execute the empty method and update the UI. This is similar to web pages in a browser: the user must press "Refresh" in the browser to retrieve the latest version of the web page.

This solution has the advantage of being simple to implement and of not requiring changes to the framework. However, there are also significant disadvantages: the user experience will suffer, and the user might not be aware that the Refresh button has to be pressed continuously. Additionally, this solution breaks the Naked Objects principle by having application developers implement code that controls the UI.

6.2.2.2 Solution: Force single-threading

Even though Microsoft has not supplied the blocking `DownloadString` method, one could relatively easily make the `CheckStatus` method wait for the download to complete by using a semaphore. The `CheckStatus` method would create the semaphore with initial value 0, launch the download thread and then `wait` on the semaphore (referred to as the `P` operation [62]). The download thread would perform the download, update the class variables and finally `increment` (or `V`) the semaphore, permitting the `CheckStatus` method to complete.

The advantages of this solution include that no changes have to be implemented in the framework to accommodate multi-threaded applications. But this solution is clumsy and requires extra work of the programmer. Having to include manual thread synchronization will make the program more susceptible to deadlocks and race conditions if not implemented properly. [62]

This solution would also cause the application to be unresponsive while the download is performed, which is what the multi-threaded design was created to avoid. Finally, it can also be argued that this solution too breaks with the Naked Objects principles, since it requires the programmer consider the UI when creating his application.

6.2.2.3 Solution: Make the source class raise an event whenever a property value changes

The underlying problem is that the auto-generated code does not have any knowledge about when a value is actually changed; it only knows when the user performs some operation on the object by using the UI. To make this knowledge available to the framework, the Kamili generator could automatically alter the source class to define a `PropertyChanged` *event*. In every property, the *set* block would be appended to raise this event whenever the property is changed. The auto-generated code would then include a handler for that event and make sure to update the UI whenever a property value changes. [63]

The main advantage of this approach is that it can be implemented in the Kamili generator and thus does not impose any requirements on the programmer. There is also a theoretical performance benefit since it is no longer necessary to update the UI after each method is executed. A disadvantage of this technique is that it requires altering the source class. Even though this can be done automatically, it introduces a new layer of complexity and thus a new source of error.
Our current design (see 5.4.2) does not allow for a generator to alter the class code, making this solution more difficult to implement.

6.2.2.4 Solution: Update the UI regularly by using a timer

A timer could run in a separate thread and periodically trigger a UI update.

This solution is fairly simple to implement, which is an advantage. It is also the only solution which takes into account properties that are computed and change without being assigned a new value. For example, an object might have a property that indicates the number of seconds since some event. This computation would be done in the property *get* block.

However, it is a wasteful solution, constantly updating the UI when no changes have occurred. There is also a problem with the user experience: text boxes correspond directly to string properties, and when the user has entered a new value in the text box, the property is updated. If the UI were to be updated by a timer at the same time that the user was entering a new value, the changes would be lost.

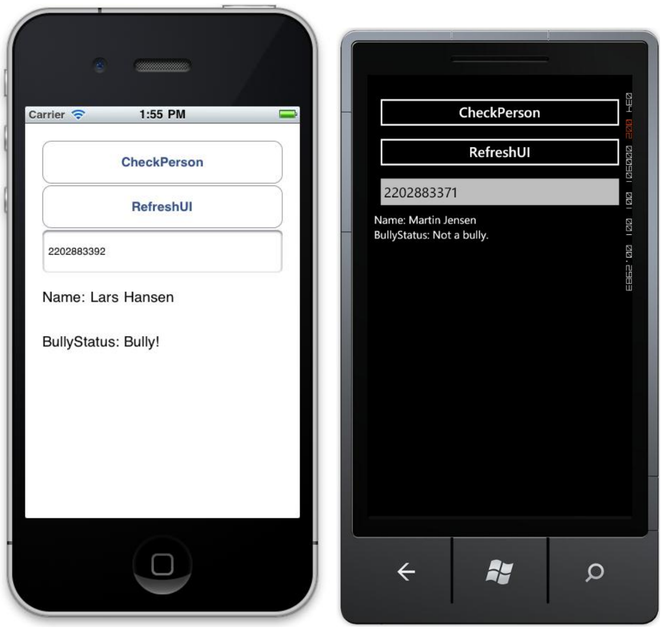
6.2.2.5 Conclusion

For the BullyReg application, we have implemented the “Refresh button” solution, since it is a correct solution that does not require changes to the framework.

We believe that the best long-term solution would be to raise an event whenever a property is updated. This solution is best because it adheres to the Naked Objects principles and it is an efficient way of keeping the UI updated.

It would be beneficial to combine this with the timer approach to address the issue of automatically changing properties. For example, the timer could update the UI with values of only the read-only properties[6], since these will never trigger an update by being the target of an assignment.

The figure below shows the app running on WP7 and iOS emulators.



2202883392

Name: Lars Hansen

BullyStatus: Bully!

2202883371

Name: Martin Jensen

BullyStatus: Not a bully.

[6] That is, properties that have **no** setter at all, not those that have a private setter (like `BullyReg.Name`)

7 Conclusion

In this thesis, we have argued that the market for mobile applications is expanding. The appearance of several new and competitive platforms is a part of this development, and developers are increasingly seeking ways to target several platforms with a single application.

We have succeeded in answering the questions from the Project Objectives through our examination of the currently available solutions for cross-platform development. Our findings indicate that the available tools provide the ability to develop for more than one platform, but that these tools are not very well-suited for rapid prototyping and that some of them require skills in several technologies before they can be used efficiently.

We have achieved our aim in designing and implementing a proof-of-concept tool for creating apps that work on several platforms. Our solution is not revolutionary in that it does build on existing technology, and it is far from ready for production use. However, it is clear that user interface development using the Naked Objects technique is beneficial in the mobile world. We have demonstrated that the technique is applicable to the mobile situation where no common user interface can be found across platforms. The method of completely avoiding any UI considerations during development can help to free developers from worrying about device specifics.

Our results suggest that framework authors should incorporate automatically generated interfaces into their products to a larger extent, since this could increase developer productivity. We suggest that more research is needed to establish whether productivity gains hold for creating larger and more complex projects rather than simple prototypes.

During the writing of this thesis, we have become aware of the multitude of approaches to mobile development, and it seems readily apparent that our solution is neither the first nor the last of its kind. We had no previous knowledge of auto-generated user interfaces, but we find the concept interesting and we believe that it would be valuable to explore more sophisticated applications of the technique in a future paper.

References

[1] Philip Elmer-DeWitt. (2010, September) CNN. [Online]. http://tech.fortune.cnn.com/2010/09/06/apples-ios-pie-chart/

[2] Canalys. (2011, January) Canalys. [Online]. http://www.canalys.com/pr/2011/r2011013.html

[3] Electronista Staff. (2010, March) electronista. [Online]. http://www.electronista.com/articles/10/03/05/mobile.app.revenues.would.top.15b.in.3.years/

[4] Apple. (2008, June) Apple. [Online]. http://www.apple.com/pr/library/2008/06/09Apple-Introduces-the-New-iPhone-3G.html

[5] Douglas MacMillan, Peter Burrows, and Spencer E. Ante. (2009, October) businessweek. [Online]. Douglas MacMillan, Peter Burrows and Spencer E. Ante

[6] Paulina Tourn and Ralf-Gordon Jahns. (2011, January) Research2Guidance. [Online]. http://www.research2guidance.com/in-2010-around-450000-smartphone-apps-have-been-published-guiding-the-app-development-market-to-become-a-multi-billion-dollar-market./

[7] Egle Mikalajunaite. (2011, July) Research2Guidance. [Online]. http://www.research2guidance.com/the-application-development-market-will-grow-to-us100bn-in-2015/

[8] The Linux Information Project. (2005, December) The Linux Information Project (LINFO). [Online]. http://www.linfo.org/cross-platform.html

[9] Joshua Topolsky. (2010, October) Engadget. [Online]. http://www.engadget.com/2010/10/11/live-from-microsofts-windows-phone-7-launch-event/

[10] Sudheer Raju. (2011, March) ToolsJournal. [Online]. http://www.toolsjournal.com/tools-world/item/157-10-of-best-cross-platform-mobile-development-tools

[11] Berlingske Tidende. (2011, June) Berlingske Tidende online. [Online]. http://www.b.dk/nationalt/diskoteker-faar-oplysninger-om-boeller-0

[12] Mathew Honan. (2007, January) MacWorld. [Online]. http://www.macworld.com/article/54769/2007/01/iphone.html

[13] Dan Frommer. (2011, June) BusinessInsider.com. [Online]. http://www.businessinsider.com/iphone-android-smartphones-2011-6

[14] Apple. (2011, July) Apple.com. [Online]. http://www.apple.com/pr/library/2011/07/07Apples-App-Store-Downloads-Top-15-Billion.html

[15] Amy Thomson. (2010, September) Bloomberg.com. [Online]. http://www.bloomberg.com/news/2010-09-13/iphone-app-downloads-outpace-android-blackberry-nielsen-says.html

[16] Bryan Costanich, *Developing C# Apps for iPhone and iPad Using MonoTouch: iOS Apps Development for.NET Developers*.: Apress, 2011.

[17] Jack Schofield. (2010, May) The Guardian Online. [Online]. http://www.guardian.co.uk/technology/2010/may/10/ipad-apple

[18] MG Siegler. (2010, February) TechCrunch.com. [Online]. http://techcrunch.com/2010/02/23/apple-iphone-pornography-ban/

[19] Gartner. (2011, February) Gartner Newsroom. [Online]. http://www.gartner.com/it/page.jsp?id=1543014

[20] Thomas Newton. (2010, October) recombu.com. [Online]. http://recombu.com/news/what-is-windows-phone-7_M12576.html

[21] Martin Heller. (2010, April) InfoWorld. [Online]. http://www.infoworld.com/d/developer-world/infoworld-review-visual-studio-2010-delivers-182

[22] Huw Collingbourne. (2010, April) PCPro.co.uk. [Online]. http://www.pcpro.co.uk/reviews/software/357286/microsoft-visual-studio-2010-professional

[23] Microsoft. (2011, June) Microsoft Developer Network. [Online].

http://msdn.microsoft.com/en-us/library/ff402530(v=vs.92).aspx

[24] Microsoft. (2010, July) Microsoft Developer Network. [Online]. http://msdn.microsoft.com/en-us/library/dd464660(v=vs.85).aspx

[25] Google. (2010) Google TV. [Online]. http://www.google.com/tv/

[26] The jQuery Project. (2010) jQuery Mobile Framework. [Online]. http://jquerymobile.com/

[27] The jQuery Project. (2011) jQueryMobile. [Online]. http://jquerymobile.com/demos/1.0a4.1/#docs/about/intro.html

[28] The jQuery Project. (2010) jQuery Mobile Overview. [Online]. http://jquerymobile.com/demos/1.0a4.1/#docs/about/intro.html

[29] Matt Doyle. (2010, November) elated. [Online]. http://www.elated.com/articles/jquery-mobile-what-can-it-do-for-you/

[30] Apple. Apple Developer. [Online]. http://developer.apple.com/technologies/ios/cocoa-touch.html

[31] Jeff Haynie. (2011, June) Re: Questions regarding Titanium Mobile. E-mail.

[32] CEO of Appcelerator Jeff Haynie. (2010, March) How Does Appcelerator Titanium Mobile Work? [Online]. http://stackoverflow.com/questions/2444001/how-does-appcelerator-titanium-mobile-work

[33] Wallace B. McClure, Martin Bowling, Craig Dunn, Chris Hardy, and Rory Blyth, *Professional iPhone Programming with MonoTouch and.Net/C#.*: Wrox, 2010.

[34] Larry Constantine. (2002) Foruse. [Online]. http://foruse.com/articles/nakedobjects.pdf

[35] Richard Pawson. (2004) Naked Objects. [Online]. http://downloads.nakedobjects.net/resources/Pawson%20thesis.pdf

[36] Apple. (2010) Wired Magazine. [Online]. http://www.wired.com/images_blogs/gadgetlab/files/iphone-sdk-agreement.pdf

[37] Novell. (2009, November) MonoTouch Documentation. [Online].

http://monotouch.net/Documentation/Linker

[38] Novell Inc. (2010, April) Mono-Project. [Online]. http://www.mono-project.com/newstouch/archive/2010/Apr-09.html

[39] ORACLE. (1997) ORACLE Sun Developer Network. [Online]. http://java.sun.com/docs/white/langenv/Security.doc3.html

[40] Theodore H. Romer, Dennis Lee, Geoffrey M Voelker, and Alec Wolman, "The Structure and Performance of Interpreters," in *Proceedings of the Seventh International Conference on Architectural Support for Programming Languages and Operating Systems*, 1996.

[41] OpenJDK. (2011) OpenJDK Web site. [Online]. http://openjdk.java.net/groups/hotspot/

[42] Wikipedia Contributors. (2011) Wikipedia, the free encyclopedia. [Online]. http://en.wikipedia.org/wiki/Mobile_browser#Popular_mobile_browsers

[43] Assaf Arkin. (2010, December) Labnotes blog. [Online]. http://labnotes.org/2010/12/29/2011-is-year-of-the-server-side-javascript/

[44] Xoetrope. XUI Zone. [Online]. http://www.xoetrope.com/zone/intro.php?zone=XUI

[45] OpenLaszlo. (2010, February) OpenLaszlo.org. [Online]. http://www.openlaszlo.org/lps4.9/docs/developers/language-preliminaries.html

[46] Naked Objects Group. (2009, December) Naked Objects. [Online]. http://www.nakedobjects.org/introduction.html

[47] Kent Beck. (2001) Manifesto for Agile Software Development. [Online]. http://agilemanifesto.org/

[48] Laurie Williams and Alistair Cockburn. (2000) NCSU. [Online]. http://collaboration.csc.ncsu.edu/laurie/Papers/XPSardinia.PDF

[49] Microsoft Corporation. (2011) Microsoft Developer Network. [Online]. http://msdn.microsoft.com/en-us/library/cc189036(v=vs.95).aspx

[50] Microsoft. (2003) MSDN. [Online]. http://msdn.microsoft.com/en-us/library/aa287786(v=vs.71).aspx

[51] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides, *Design Patterns*.: Addison-Wesley Professional, 1994.

[52] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides, *Design Patterns*.: Addison-Wesley Professional, 1994.

[53] Microsoft Coporation. (2011) Microsoft. [Online]. http://research.microsoft.com/en-us/projects/contracts/

[54] The Institute of Electrical and Electronics Engineers. (1990, September) idi. [Online]. http://www.idi.ntnu.no/grupper/su/publ/ese/ieee-se-glossary-610.12-1990.pdf

[55] Adam Kolawa and Dorota Huizinga, *Automated Defect Prevention: Best Practices in Software Management*.: Wiley-IEEE Computer Society Press, 2007.

[56] Microsoft. (2010, March) Microsoft Developer Network. [Online]. http://msdn.microsoft.com/library/ee958158.aspx

[57] Microsoft. (2011) Microsoft. [Online]. http://www.microsoft.com/windowsembedded/en-us/windows-embedded.aspx

[58] Novell. (2009, October) MonoTouchWiki. [Online]. http://wiki.monotouch.net/HowTo/WebServices/Using_WCF

[59] Aaron Skonnard. (2001, January) MSDN Magazine. [Online]. http://msdn.microsoft.com/en-us/magazine/cc302158.aspx

[60] World Wide Web Consortium. (2011, March) W3.org. [Online]. http://www.w3.org/TR/exi/

[61] Edward A. Lee, "The Problem With Threads," Berkeley, CA 94720, U.S.A., 2006.

[62] Edsgar W. Dijkstra, "Cooperating sequential processes," Eindhoven, The Netherlands, 1965.

[63] Microsoft. (2009) Microsoft Developer Network. [Online]. http://msdn.microsoft.com/en-us/library/8627sbea(v=VS.100).aspx

[64] Steve McConnell, *Code Complete, 2nd. Ed.* Redmond, USA: Microsoft Publishing, 2009.

[65] Tom Sullivan. (2003, May) InfoWorld. [Online]. http://www.infoworld.com/d/developer-world/sun-seeks-grow-java-developer-base-10-million-299

[66] Eric Palto. (2008, January) ezinearticles. [Online]. http://ezinearticles.com/?History-of-Multi-Touch-Technology&id=3443137

[67] Steve McConnell, *Code Complete, 2nd ed.* Redmond, WA, USA: Microsoft Press, 2004.

[68] Wallace B. McClure, Martin Bowling, Craig Dunn, Chris Hardy, and Rory Blyth, *Professional iPhone® Programming with MonoTouch and.NET/C#.*: Wrox, 2010.

[69] David Lyons. (2010, April) newsweek. [Online]. http://www.newsweek.com/2010/04/01/microsoft-s-unsung-success.html

[70] Alan Kay. (2003, August) Dr. Alan Kay on the Meaning of "Object-Oriented Programming". [Online]. http://userpage.fu-berlin.de/~ram/pub/pub_jf47ht81Ht/doc_kay_oop_en

[71] Joab Jackson. (2011, July) pcworld. [Online]. http://www.pcworld.com/businesscenter/article/235399/ballmer_windows_phone_7_not_successful_yet.html

[72] Mathew Honan. (2009, January) Macworld. [Online]. http://www.macworld.com/article/54769/2007/01/iphone.html

[73] Horace Dediu. (2011, January) asymco. [Online]. http://www.asymco.com/2011/01/31/fourth-quarter-mobile-phone-industry-overview/

[74] College of Lake County. (1998, January) Gopher - Introduction. [Online]. http://www.clc.cc.il.us/home/com589/gopher.htm

[75] Robert Cailliau. (1995, November) A Short History of the Web. [Online]. http://www.inria.fr/Actualites/Cailliau-fra.html

[76] Mette Lindegaard Attrup and John Rydding Olsson,.: Økonomforbundets Forlag, 2008, p. s. 148.

[77] F. Anklesaria et al. (1993, March) The Internet Gopher Protocol. [Online]. http://tools.ietf.org/html/rfc1436

[78] Monty Alexander. (2008, March) ezinearticles. [Online]. http://ezinearticles.com/?Touch-Screen-Smartphone&id=3661945

[79] Microsoft. (2011) Microsoft. [Online]. http://msdn.microsoft.com/en-us/library/ff402535(v=vs.92).aspx

[80] O'Reilly Media. (2005, August) O'Reilly Media. [Online]. http://www.oreillynet.com/wireless/2005/08/23/whatissmartphone.html

[81] Deloitte LLP. (2009) Deloitte. [Online]. http://www.deloitte.co.uk/TMTPredictions/telecommunications/Smartphones-clever-in-downturn.cfm

[82] Nokia Communications. (2011, February) microsoft. [Online]. http://www.microsoft.com/presspass/press/2011/feb11/02-11partnership.mspx

[83] ArticleBase. (2011, July) articlebase. [Online]. http://www.articlesbase.com/marketing-tips-articles/mobilising-your-brand-web-vs-native-apps-4992936.html

[84] Apple. (2010) Apple. [Online]. http://developer.apple.com/library/ios/#documentation/Miscellaneous/Conceptual/iPhoneOSTechOverview/Introduction/Introduction.html#//apple_ref/doc/uid/TP40007898

[85] Apple. (2010) Apple. [Online]. http://developer.apple.com/library/ios/#referencelibrary/GettingStarted/URL_iPhone_OS_Overview/_index.html#//apple_ref/doc/uid/TP40007592

[86] Open Handset Alliance. (2009) What is Android? [Online]. http://developer.android.com/guide/basics/what-is-android.html

Acknowledgements

Christian W. Probst, Associate Professor, Project Supervisor

Jeff Haynie, Appcelerator CEO, Interviewee