TEC
Instituto Tecnológico de Costa Rica

TEC – Área Ingeniería en Computadores (CE)
Acta de Aprobación de Trabajo de Graduación

Con fundamento en lo que establece el "Reglamento de Trabajos Finales de Graduación del Instituto Tecnológico de Costa Rica", el Tribunal Examinador del Trabajo Final de Graduación, nombrado con el propósito de evaluar el proyecto final de graduación.

"Swift mobile platform analysis:
KMU Inventory approach"

Habiendo analizado el resultado general del trabajo presentado por los estudiantes:

Primer Apellido	Segundo Apellido	Nombre	No. De carné
Bolaños	Murillo	Alejandra	200938622

Emite el siguiente dictamen:

APROBADO

CALIFICACION: 100 puntos.

○ REPROBADO

● SE RECOMIENDA ○ NO SE RECOMIENDA

Brindarle una nueva oportunidad para la DEFENSA PUBLICA de su Trabajo Final

NUEVA FECHA: _____

Dando fe de lo aquí expuesto firmamos

M.Sc. Milton Villegas Lemus
Profesor Asesor

Dr. Roberto Pereira Arroyo
Profesor Lector

M.Sc. Aníbal Coto Cortés
Profesor Lector

M.Sc. Isaac Ramírez Herrera
Profesor Lector

04 de Mayo del 2015
Fecha

Biblioteca José Figueres Ferrer
Tel:2550-2263• Fax:2591-4820

ANEXO 3 : CARTA DE ENTENDIMIENTO
(Ejemplo)

Señores
Instituto Tecnológico de Costa Rica
Biblioteca José Figueres Ferrer

Yo Alejandra Bolaños Murillo carné 200938627 autorizo ☒ no autorizo ☐ a la Biblioteca José Figueres del Instituto Tecnológico de Costa Rica disponer del Trabajo Final realizado por mi persona, con el título Swift mobile platform analysis: KMU Inventory approach.

para ser ubicado en el Repositorio institucional y Catálogo SIBITEC para ser accesado a través de la red Internet.

Alejandra BT
Firma de estudiante
Cédula 206910654

Instituto Tecnológico De Costa Rica
Computer Engineering Department

"Swift mobile platform analysis: KMU Inventory approach"

Alejandra Bolaños Murillo

Cartago April, 2015

Abstract

This is an analysis of Apple's new programming language: Swift. The investigation takes place in the University of Applied Sciences FH Joanneum in Styria, Austria. A guideline for time and skills administration is written based on the experience developing an iOS mobile application. This application is developed for the administration of an inventory in iPhone and iPad. It is called KMU Inventory App. The outputs of this work are: a document for guiding new developers in this area and the analysis of the hardware and software involved.

Acknowledgments

Foremost, I would like to express my sincere gratitude to my advisor Prof. Milton Villegas and Jennier Solano for the continuous support and guidance during this project.

I would like to thank also the FH Joanneum professors and researchers for giving me the opportunity to work with them to accomplish this project.

Dedication

I dedicate this work to my parents Oliveth and Ligia, for their support, love and encouragement. Without them I would never have enjoyed so many opportunities.

I also dedicate this to my sisters for the encouragement and friendship. To Adriana for her support especially during my first years of studies. To Valeria and Natalia for the patience and help during my studies.

Table of Contents

List of Tables

List of Figures

1 Introduction

Mobile development is considered an important and growing technological industry. For this reason, different environments and technologies are being created in this field for final users and developers. As a consequence, there is an increasing amount of programming platforms, features, and devices. The technological industry today is therefore being filled by all kinds of opinions about best technologies and ways to develop them.

According to Computer World's website (Keiser, 2015), Apple's iPhone 6 and iPhone 6 Plus prompted a surge in iOS smartphone users. In the U.S, iOS market share climbed from 43.1% in October to 47.4% in November 2014. While during the same period of time, Android's U.S. share fell from 50.4% to 48.4%, and Google's operating system went just one point beyond iOS. It gives a clear image of the importance of iOS development for people involved in the mobile industry.

Recently, Apple has released a new programming language named Swift. It is a totally new approach for developers in iOS and OS X. This company developed it to simplify the designing and programming phases of a project using Apple technologies and to give the programmer a flexible mobile working framework.

Recently released languages like Swift do not have a strong background of references and previous user experiences in comparison to the most common languages, for instance Java and Objective C. Old languages have documentation from their developers and the programmers that used them for a long time.

New programming languages do not provide developers enough information about their performance. They are not properly characterized by users due to their limited time in the market. This lack of data creates the need to provide a clear image of the platform.

As a result of the new release, Swift programming language is a point of interest for research institutions like the FH Joanneum. Even though members have no previous experience with this language, they are familiar with Apple's technology, especially with

the previous language, Objective C. The need to develop an application in this university is not only to sell it, but also to investigate. The research department is looking for documentation of Swift's behavior in the mobile area in order to help students and developers learning this language.

1.1 Project Area

The project is oriented towards research in mobile platforms to create a software and hardware analysis of a programming language named Swift and its guideline. This guide intends to be used as a reference framework to help future developers and people interested in learning about Apple's technology. The information was gathered based on a mobile application developing process.

The application developed is an Inventory program for iPhone and iPad. It is implemented in the KMU-Goes-Mobile project. It takes place in the Research and Development area at the FH Joanneum University in Kapfenberg.

1.2 Project Context

Swift is a recently released programming language. Its development began in 2010 by Chris Lattner, and was presented to the public on September 9th, 2014 with the version 1.0. On October 22, 2014, it reached the version 1.1 alongside XCode 6.1.

There was a lack of expertise because this language was released by the time the present research project starts (October 2014). There were neither Swift experienced programmers in the FH JOANNEUM laboratory, nor Swift application performance analysis available to the public, and nor guidelines for beginners in this area.

The lack of information about this programming language created the need of a technical exploration of the development process to provide future developers and researchers with criteria to learn Swift and its features for their projects.

The mobile application developed to build this guide aims to help small and medium sized enterprises. These companies usually do not have their devices in an inventory which makes it hard to keep track of who is using them and where they are. As a solution,

KMU-goes-mobile developed a set of programs to help these enterprises to keep track of their actives.

This project was developed for iOS. Therefore, an environment compatible with Apple was set. The operating system used was OS X Version 10.9.5 and the platform Xcode. The project is oriented towards a research about mobile development, documentation, and testing in the Research and Development area in ITM department of FH JOANNEUM in Kapfenberg, Austria.

1.3 General Description

1.3.1 Background of the FH JOANNEUM

FH JOANNEUM University is located in Styria, Austria. It was founded in 1995. It is one of the leading universities of applied sciences (FHs) in Austria. It has forty programs in Health Sciences, Information, Design & Technologies, International Business and Live, Building, and Environment areas.

The university's history starts with the Archduke John of Austria, who was one of the most innovative figures of his time. He introduced an incredible number of inventions to his chosen home of Styria from 1811. Many of those continue to have an effect to the current day. He is the precursor of the Graz University of Technology, founded Styria's state library, and today's University of Leoben.

In 1815, after traveling to England, home of the industrial revolution, he encouraged the railway building throughout the Habsburg Empire and the modernization of Styria's mining and agricultural industries. He never lost sight of the social impact of all these upheavals.

It was due to Archduke John that Styria got developed from an agricultural state into a leading and highly innovative industrial region. John's thoughtful approach, his openness to all new things, and his affinity for the local population made him one of the most popular Habsburgs.

The Styrian government wished to highlight John's spirit in naming Styria's newly founded university of applied sciences "FH Joanneum" in 1995. (FH JOANNEUM, n.d)

1.3.2 Research and Development

The FH JOANNEUM campus in Kapfenberg has a department that aims to integrate innovative applied research and development in teaching in order to maintain the quality of degree programs and provide practice-oriented education for students.

The applied research and development is geared both to the needs of industry and public administration. It is the interface between basic research on one hand, and innovative services and products development on the other. FH JOANNEUM's staff and students work together with the clients to develop comprehensible solutions that meet international quality criteria. They also examine to what extent the results can be generalized or applied in other fields. Research achievements are reflected in innovations, publications, and patents.

The highly qualified and experienced FH JOANNEUM staff is continually involved in a wide range of research projects, generating knowledge for the university, business, and society. FH JOANNEUM takes an inter and trans disciplinary approach in tackling key research issues of the future in cooperation with partners from business, industry, and public institutions.

The best description is given by the FH Joanneum website: "KMU-Goes-Mobile is a research project in support of small and medium size enterprises founded by The Austrian Research Promotion Agency (FFG) to promote innovative ideas, concepts, and free applications for the use of smartphones and tablets for small enterprises that do not have their own IT departments for software development". (FH JOANNEUM, n.d)

1.4 Problem Description

1.4.1 Problem Context

The FH JOANNEUM IT department found it important to include Swift within its background because of having experience working on different mobile areas including Apple's technologies, and with mobile applications development projects. As it was a new programming language with less than a month being in the market, mobile developers did not have a current guideline to judge the path of their development approach or teaching approach in the case of professors using Swift.

The institution has worked in a project called KMU-Inventory. As they had an Objective C application already developed, the Swift application became the next step. That was an opportunity to retrieve information about development and learning to use Apple's new programming platform.

1.4.2 Problem specification

There is always a period of time where not enough documentation can be found when every new product is launched. It is normal because of the lack of users and experts. Swift was not the exception. Less than a month from the release date, there was not enough data and sample applications developed in Apple's new programming language.

As a technical university, the FH JOANNEUM was interested in building a reference guideline from this new technology to be used as a working framework that helps lecturers to teach the language and developers to start learning and evaluating the approach.

A reference framework was needed. The development of an application was one good step to create an authentic analysis. Thus, the implementation of a newer version of KMU Inventory app is done.

1.4.3 Need and Justification

Every programming language has its good and bad features, but having a reference guide could play a crucial role on a project development and its success. Being able to tell if a platform tool will suit the needs of the product to be developed is a real advantage for every project manager.

Also, having documentation about Swift's pitfalls and learning procedure is very helpful for new programmers in the area. For the FH JOANNEUM the new acquired knowledge is valuable information to provide a reference framework to those who want to start incurring in this new area. The users of this outcome could be investigators, students, or lecturers.

The development of a testing application generates knowledge of the features that a programming language and platform can offer. It also helps the community by providing small and medium enterprises with a mobile inventory platform.

1.5 Objectives

1.5.1 General

Construct a function of criteria to be used by developers and researchers in mobile applications to choose or not Swift programming language for their projects based on hardware and software variables.

1.5.2 Specific

-Construct a weight function that works as a reference framework and guideline for future developers and managers incurring in Swift technology to set a development strategy.

-Define variables based on the criteria documented through the development of an application for iOS using Swift language for the construction of the weight function.

-Construct a learning document as a guide for new programmers in Swift language, including best practices and suggested procedure to learn the programming language.

-Analyze the CPU and memory usage over time performance of an application using Swift language and compare them to other applications running on the same mobile.

1.6 Benefits and Beneficiaries

The beneficiaries are:

- Small and medium enterprises: Using the application developed, people running these enterprises can get easy access to a new tool to keep track of their items without any important economic cost.
- FH JOANNEUM lecturers: The guideline and the Wikibook constitute a base for students to learn the main difficult points of this platform in an informed way.
- KMU-goes-mobile project: Research results are important findings for every educational university, this new data leads to a better understanding of Apple's current and most recent programming language.
- Swift beginners: All new developers can take advantage of this research output by checking the reference framework of the learning context and a beginner's previous documented experience.
- Swift developers: All code is free licensed software and accessible by anyone everywhere. Developers are provided with an already programmed application that could be used as code reference.

1.7 Assumptions and Limitations

This project aims to develop a first approach of an application for a research process. It is neither meant to compete it with local industries, nor to offer a limitation of the market business to any of the current products.

The application is a fully working demo, not a final stable version. It is not offered accompanied with any kind of later support for enterprises. Also, it is developed using the FH JOANNEUM's framework and tools. All code generated is under Open Source Licensing and documented as part of KMU-goes-mobile project. All the information gathered and the analysis is described in a Wikibook available online.

1.8 Risk Analysis

1.8.1 Risks

- Swift language was released three months before this research started. Consequently, it was hard to find documentation about bugs or any complex actions.
- Some tools were still in the development process, so it leaded to an uncertain time of the releases and also an unexpected behavior.
- Specifications of the project were written in German and most of the clients were Austrian enterprises. Thus, there was the risk of misunderstanding some of the requirements and the need of changes of this project.

1.8.2 Mitigation Actions

- The first part of the project was an extended practice and learning experience. Taking time to make a complete guide about how to develop a mobile application in Swift can mitigate the first risk.
- A meeting with the coordinators and developers of these tools to discuss the releasing time and the available resources mitigates the second risk.
- All meetings and their notes were in English to mitigate the third risk.

1.9 Scope, Deliverables and Limitations

1.9.1 Work Characterization

This is a research of a mobile language behavior. The investigation is done during the development of an application. It leads the project towards a theoretical and practical approach. The output shows how the practical side of the project generates the knowledge showed in the theoretical side.

1.9.2 Process Description

Training, Setting Environment, and Language Investigation:

The tools used for documentation, version control, application API, and testing were developed or hosted by the local university. For this reason, a period of training was necessary.

The language was unknown by the team, so it had to be investigated to understand the project approach. The investigation includes the learning procedure to program in this environment. Apple's Swift programming was set and tested to learn to develop in this language and work with it.

Application Development and Swift Characterization

During this stage, the application development and presentation, and the meetings with the development team took place. The language tools and the developer's experience during this stage constitute the basis for the Application and language experience documentation.

Application and Language Experience Documentation

Based on the last step outputs, a guideline was constructed in this stage to work as a reference framework for future beginners. The work here is theoretical and analytical. The documentation of the language, its features, and learning process were written in a Wikibook available online and in a guideline presented in this work.

1.10 Tools

The software and hardware used during the development and testing of the Inventory Application are listed below:

- A distributed revision control system: Git repository. It is located in the university's private server: Innsbruck.
- A flexible project management web application: Redmine. It is a bugtracker written using the Ruby on Rails framework. It is cross-platform and cross-database.
- A task administrator: Scrumpy for Scrum-Board. This software was developed by the IT department in the FH Joanneum for scrum based projects to administrate tasks.
- Apple's IDE for Swift development: Xcode. Swift is used as the main programming language in the project. Leading to the mandatory usage of both platforms during the programming stage of the application development.

1.10.1 Deliverables Description

Functional Application:

It was necessary to develop and implement an application capable of managing the state of items in an inventory. The "Inventory Component" manages these devices. Each system user can borrow devices for himself or for another person. This was done via mobile device through the "Inventory App". This mobile application runs on iOS and was developed using Swift as the main programming language, and also the FH JOANNEUM's platforms for mobile development.

Swift documentation:

This document is a Wikibook about the best procedures containing the following data: developers learning experience, features for mobile development implemented in the

language, access to hardware features from within the program, and an Apple’s manual for its first usage including needing tools.

2 Theoretical Framework

2.1 Swift Programming Language

Swift is the iOS new programming language released on June 2014. It was created by Apple as their language for iOS and OS X projects development. It works side-by-side with Objective C, the previous standard language for iOS and OS X applications.

Swift is created as an innovative programming language for Cocoa and Cocoa Touch. Apple claims that the writing code is interactive and its syntax is concise but expressive. Swift is the result of Apple's latest research on programming languages combined with decades of experience building platforms.

2.2 XCode

According to Apple's developer website (Apple Inc, 2014d), XCode has got a series of tools to make programming in Swift easier and more intuitive. These two characteristics work along to provide a user friendly programming environment. About XCode, Apple says in its website: " It's easy to create a brand new app using 100% Swift code, add new Swift code or frameworks to existing apps, and view documentation in either Swift, Objective C, or both." With this quote they highlight the benefits and dependence between Swift and XCode.

Live Rendering takes the most important place within the main features of XCode. It makes the Interface Builder show the custom objects added by the programmer during the designing time and also shows the changes done in every save. This tool gives the developer an overview of its application and the applications flow since an early stage, the designing stage.

It is also possible to debug the code in Swift using XCode's View Debuggin tool, the XCTest framework is fully integrated into XCode to provide a performance tool capable of running, comparing performance, and displaying the change over time. It

clearly shows when test results change, alerting the user of regressions in performance or functionality as monitoring the quality of the app.

2.3 Wikibooks

The formal definition from its website about itself is: “Wikibooks is a project for collaboratively writing open-content textbooks that anyone can edit. Contributors maintain the property rights to their contributions, while the Creative Commons Attribution-ShareAlike License and the GNU Free Documentation License makes sure that the submitted version and its derivative works will always remain freely distributable and reproducible.”

A wikibook is a collection of open-content textbooks that can be written by anyone, leading to a common space where people can share their knowledge in order to create a free source of material that can be access all over the world. The site is a place where textbooks, guides, manuals, and annotations of different topics can be found in different languages and presentations. The information can be used for educational reasons or self-learning.

As said before, anyone can write a book, but other members of the community can modify, improve, and delete the Wikibooks. Also, the moderators can approve the information and review it “Wikibooks is not an in-depth encyclopedia on a specific topic nor are pages encyclopedia-formatted articles. Books build knowledge from one page to the next with inter-dependency between pages. Books in progress are sometimes organized in an encyclopedic manner until developed into proper books.” (Wikibooks, 2010)

2.4 Apple’s Terminology

Due to its own way of development and privacy, Apple has got some terms to refer to their elements and tools during software development. In this document, it is important to know that some of them are used and considered for the designing and implementation of the mobile application.

2.4.1 Storyboard

The storyboard is the main concept present in the flow of an iOS application. It is where the developer can create and modify graphically all the paths and configure the flow of the program between the screens, the views, and the user inputs. Figure1 shows the sketch of the application's storyboard.

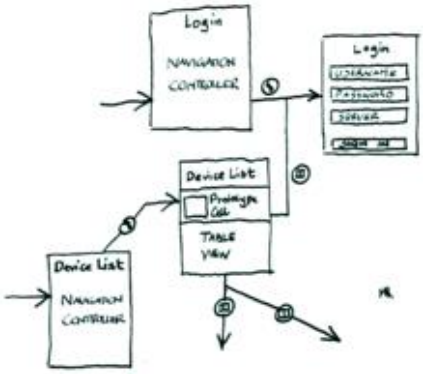

Figure1. Sketch for a Storyboard

2.4.2 View Controllers

These are classes that manage the data and visual appearance of the screen showed by the application. When the user interface is showed, the content is managed by one View Controller or more. That is why View Controllers are called the skeletal framework of the iOS apps. The programmer can create and customize these controller classes to manage the application behavior. The first View controller is called Root View Controller.

2.4.3 Navigation Controller

It controls the stack of View Controller for the app navigation and its flow. It contains and manages a set of View Controllers. Each of them contains and manages a set of views. Their purpose is to coordinate the navigation between the View Controllers.

As a summary, this hierarchy concept can be explained as follows: one Navigation Controller controls many View Controllers, and one View Controller can control many Views. This is shown in figure2.

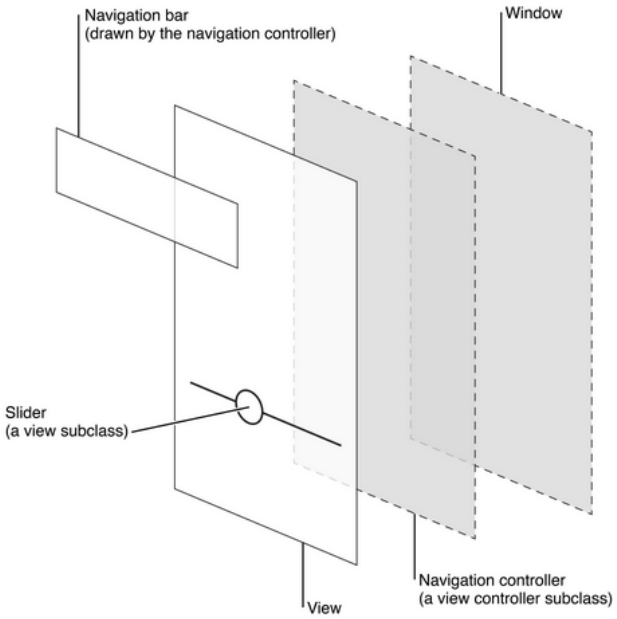

Figure2. View Hierarchy in iOS (Apple Inc, 2014d)

2.4.4 AV Foundation API

AV Foundation is one of the frameworks used to control audiovisual media in Swift language. It provides an interface that can be called directly from code in Apple devices.

2.5 User Experience

The user experience, also referred as UX, is the experience the end user gets from the program. In other words, it is formed by the interaction, behavior, perception, and satisfaction the user gets when using a product. This can be evaluated in a controlled environment observing the user interaction and reaction during a program run.

Jakob Nielsen, in his classic book, Usability Engineering (Nielsen,1993) says usability applies to all aspects of a system which a human might interact, thus usability is not a single and one-dimensional property of just the user interface. It includes learnability, efficiency, memorability, errors, and satisfaction.

As Chauncey Wilson points out in his book User Experience (Wilson, 2010), now the term “user experience” has emerged from “usability”, but the first one includes also

dimensions as aesthetics, pleasure, and consistency with moral values as important for the success of products and services. Then, user experience is formed by all interactions and impressions a user experiments while interacting with the product. The usage the developers give to this features will be crucial for the final user impression and perception of the product.

2.5.1 Aesthetics

There is a common polemic and deep discussion in the artistic field about the definition and meaning of the word "aesthetics". In this work, it refers as what is pleasurable to the senses. The most common usage of the term is to evaluate the beauty and attractiveness of something.

The UX designers have the shore to consider every stimulus that might influence the user interaction because aesthetics is not just about the artistic merit of Web buttons or other visual effects, but also about how people respond to these elements (Anderson, 2012).

For a designer, all these concepts constitute the rules and interest points for every work they do, but what about programmers who are not designers?

Apple has developed a guide for User Experience that aligns the most common requirements for a good interface, with the company's designing standards.

2.5.2 Apple's User Experience Guideline

For a private development, Apple does not define specific iOS Human Interface guidelines, but for the release and application in the Apple Store, there are requirements the programmer must follow. Here it is an overview of the rules and guides used in a project published in their website (Apple Inc, 2014a).

About the Design iOS embodies the following themes:

- **Deference**: The UI helps people to understand and interact with the content, but never competes with it. To make sure to accomplish this, the developer can follow the

next steps: (1) Take advantage of the whole screen. (2) Reconsider visual indicators of physicality and realism. (3) Let translucent UI elements hint at the content behind them.

- **Clarity**: Text is legible at every size, icons are precise and lucid. A sharpened focus on functionality motivates the design. Apples suggestion to accomplish this is: To use plenty of negative space, let color simplify the UI, use the default fonts and borderless buttons.
- **Depth**: Visual layers and realistic motion impart vitality and heighten people's delight and understanding. To do so, iOS often displays content in distinct layers that convey hierarchy and position, and that helps users to understand the relationships among onscreen objects.

2.5.3 About Layout

Apple's devices can work on landscape or portrait (see figure 3) and their screen sizes are variable between models, that is why an application should give a great experience in each environment, taking advantage of adaptability.

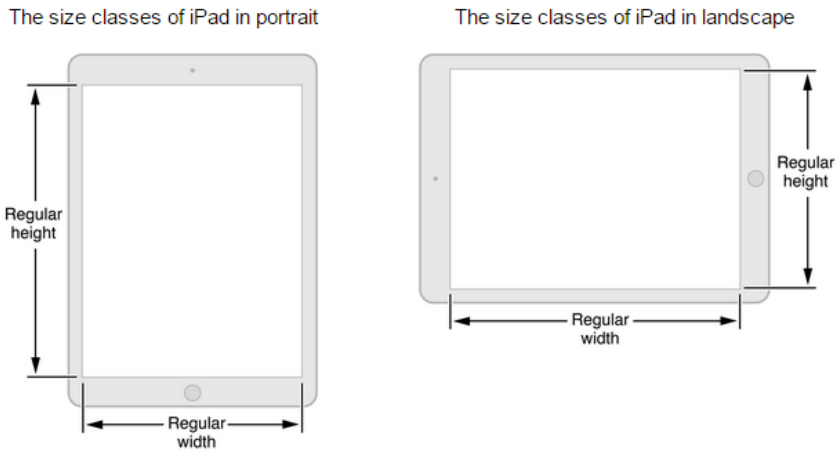

Figure 3. IPad screen orientation possibilities (Apple Inc, n.d b).

When a single device or orientation is chosen the guideline is not mandatory, but in this market, it rarely happens. The main highlights are:

- Maintain focus on the primary content in all environments
- Avoid gratuitous changes in layout
- Be straightforward if the app runs in only one orientation
- Avoid displaying a UI element that tells people to rotate the device
- Support both variants of an orientation
- Use layout to communicate
- Make it easy to focus on the main task by elevating important content or functionality
- Use visual weight and balance to show users the relative importance of onscreen elements
- Use alignment to ease scanning and communicate groupings or hierarchy
- Make sure that users can understand primary content at its default size
- Be prepared for changes in text size
- Avoid inconsistent appearances in the UI
- Make it easy for people to interact with content and controls by giving each interactive element ample spacing
- Give tappable controls a hit target of about 44 x 44 points

2.5.4 About Terminology and Wording

The communication between the application and the user should be done in a polite but simple way, to achieve this, following the next steps is necessary:

- Use terminology that the application users will understand
- Use a tone that’s informal and friendly, but not too familiar
- Avoid unnecessary words
- Give short labels to controls or use well-understood icons
- Be accurate when describing dates

- Make the most of the opportunity to communicate with the potential users by writing a great App Store description
- Correct all spelling, grammatical, and punctuation errors
- Keep all-capital words to a minimum
- Describe specific bug fixes in every new release

2.5.5 About Integrating the Application to the iOS Standard

Following this guideline the developer gives users the experience they expect. The first instruction given is to use the standard UI elements. In order to do so, the subsequent rules should be followed:

- Follow the guidelines for every UI element
- Do not mix UI element styles from different versions of iOS
- Avoid creating a custom UI element that performs a standard action
- Use the appearance customization APIs, access properties and attributes, then apply custom and system provided icons to the controls
- Do not use system-defined buttons and icons to mean something else
- If no system-provided button or icon has the appropriate meaning for a function, it is possible to create one following the instructions given by the website

2.5.6 About Navigation

The structure of every application is different; However, regardless the navigation style, the users path should be predictable and intuitive. A standard and useful UI element is the navigation bar. It is an easy way to move across a hierarchy of data or a tab bar and support flat information architecture as well as its persistence.

In general, to give users one path to each screen is the best to do. If there is one screen that users need to see in more than one context, consider using a temporary view, such as a modal view, an action sheet, or an alert.

2.6 Training, Setting Environment and Language Investigation In

This stage of the process is divided in two sections: (1) Training and environment setting and (2) Language investigation.

2.6.1 Training and environment setting

The tools to be used during the project require a period of training because they were never used by the developer before. Also, all the accounts needed during the project had to be set. A procedure was followed to do it and it is specified here.

The steps taken to accomplish this stage were:

1) The University's account for internet access was solicited and also the mail server settings were configured with the help of the IT Support department in the FH JOANNEUM.
2) The assigned PC Computer's software was updated in order to install the programming environment.
3) XCode installation was done.
4) The iOS developers account was created and all credentials to allow programing in physical devices were installed.
5) XCode configuration with the iOS account was done (this step is complemented by the language investigation stage).
6) Join the project Git repository for the iOS applications of the Inventory in the FH JOANNEUM's server.
7) Clone Git repository of the 6th step.
8) Create a user in the Joomla!! web service "KMU goes Mobile" project to join the development team.
9) Join the platform BugTracker to handle bug reports.
10) Install the iOS deployment credentials in the XCode session.
11) Set a developers account on the iOS testing devices.

2.6.2 Investigation

During this stage, the first approach towards the language experience is done. First, Apple's official website for Swift (Apple Inc, 2014c) is consulted to retrieve the details of its basic characteristics as a programming language. Also, other different websites are consulted for extra information and characterization as starting point for the guideline. Then, the main and useful findings are listed in this section.

Language Official Description (Apple Inc, 2014c)

Swift is a programming language for iOS and OS X that does not include the constraints of C compatibility as Objective C did. The developers simplified the memory management with Automatic Reference Counting (ARC). And its framework stack was built on the base of Foundation and Cocoa.

Swift is a multi-paradigm, compiled programming language built intended to be fast by using a high-performance compiler: LLVM. Swift's code is transformed into optimized native code, and tuned. The syntax and standard library have also been tuned with the purpose of making the most obvious way to write the code.

Swift is a successor to the C and Objective C languages. It includes low-level primitives such as types, flow control, and operators. It also provides object-oriented features such as classes, protocols, and generics.

The language developers claim Swift feels familiar to Objective C developers because it adopts the readability of Objective C's named parameters and the dynamic object model. They also claim Swift is friendly to beginners because it offers support to a new feature called Playground that allows them the experimentation of the Swift code. They are also able to see the results immediately.

Swift provides its own versions of all fundamental C and Objective C types as it is described in the official documentation in Apple's website. This includes Int for integers, Double and Float for floating-point values, Bool for Boolean values, and String

for textual data. Swift also provides versions of the two primary collection types: array and dictionary.

Swift uses variables and constants to store and refer to values by an identifying name. Constants are used throughout Swift to make code safer and cleaner. What makes Swift better is that it also introduces advanced types that are not found in Objective C: Tuples and Optionals.

Optionals are similar to using nil with pointers in Objective C, but they work for any type, not just classes. They are safer but more memory expressive than nil pointers in Objective C. They are called one of the strongest Swift's features, helping programmers to be clear about the types of values their code works with.

Swift and Objective C

A remarkable review that expresses the current lack of user's documentation and popular information of this new programming language is made by Alfie Hanssen in his blog:

"With Objective C being close to 35 years old, and it having been the language used to build Mac OSX and iOS applications for years, there's a substantial community and body of knowledge out there constantly documenting best practices, approaches to common and not-so-common problems, pretty much anything you can think of about using Objective C. When learning Swift, something that may be an issue for a little while will be the comparatively slim set of resources out there. Stackoverflow, blogs, tutorial sites are quickly compiling info on all aspects of Swift development, but it'll take some time to reach the saturation point that Objective C currently enjoys." (Hanssen, 2014)

Also the findings Jesse Squires describes in his web site (Squires, 2014) show that if Swift code is not optimized, Objective C is still noticeably faster than Swift. However, when safety features are deactivated, Swift orders of magnitude are better than Objective C's. Squires also points out the results in tables, those are shown on table 1 and 2.

It is easy to notice the improvement of the language speed if safety rules are dismissed. The difference between Objective C performance and Swift performance is positive when Swift is not optimized and negative when it is (see Table 1 and Table 2).

Table 1

Objective C performance and Swift performance time comparison with optimization.

T = 10 N = 10,000 Debug	**Std lib sort**	**Quick sort O(n log n)**	**Heap sort O(n log n)**	**Insertion sort O(n2)**	**Selection sort O(n2)**
Objective C	0.015732 s	0.011395 s	0.025252 s	1.931189 s	3.762144 s
Swift	1.536891 s	1.633227 s	4.714571 s	625.810322s	519.386646s

Note. From "Apples to apples," by Squires,J.

Table 2

Objective C performance and Swift performance time comparison without optimization.

T = 10 N = 10,000 Debug	**Std lib sort**	**Quick sort O(n log n)**	**Heap sort O(n log n)**	**Insertion sort O(n2)**	**Selection sort O(n2)**
Objective C	0.011828 s	0.010285 s	0.019763 s	1.776664 s	3.497402 s
Swift	0.001306 s	0.001426 s	0.002259 s	0.297713 s	0.068731 s

Note. From "Apples to apples," by Squires,J.

One of the findings of this investigation is that the performance was often measured on tests created especially for this task, not on a normal environment such as fully created app. So the results did not show information in a user friendly way to estimate the application performance. Also, time was the only variable to measure it, not taking into account CPU or memory usage.

User Experience rules listed in the Theoretical Framework in Apple's Terminology and User Experience sections were part of this investigation. They are not included in this section because they are already cited in the present document.

3 Methodological development

The development of this project is divided in three stages. Each of them is depends on the previous one. They are: (1) Training, setting environment and language investigation, (2) Application development and Swift's features analysis, and (3) Application and language analysis. Each step is also divided in procedures that are deeply explained in the sections below.

3.1 Application Methodological Development

It was the core of the experiment. The application was developed, the meetings with the development team were held and the programming language investigation was done. The language tools and the developer's experience during this stage constitute the basis for the next step: Application and Language Experience Documentation. The following sections explain the modeling and development process of the experiment.

3.2 User Stories

Defining the requirements of the experiment was the first step taken. In this case, the experimental application was divided in seven different user stories described below (see tables 3 to 11). These user stories are created by the client and given to the developer as a requirement for the application.

Table 3

User Story 1: Connect to Inventory-Server

Criteria	**Connect to Inventory-Server**
Description	As a new User of the inventory App, I want to connect to the Server with Username,

	Password and Server-Address
Acceptance Criteria	*Valid Username is filled in to the Username-Textbox
Definition of Done	*Valid Password is filled in to the Password-Textbox

Table 4

User Story 2: See all devices which are available

Criteria	**See all devices which are available**
Description	As a User of the Inventory App, I want to see all available devices.
Acceptance Criteria	*If login is set: Start App * If login not set: Proceed US1 -> automatically redirected to DeviceListView
Definition of Done	All devices are shown at the ListDeviceView

Table 5

User Story 3A: Lend a device from a list

Criteria	**Lend device (choosing from List)**
Description	As a User of the Inventory App, I want to lend a device by choosing it from the list.
Acceptance Criteria	*Device is chosen from the DeviceListView.

Definition of Done	* New View opens which shows the chosen device including most of its information (available or not, current owner, name, location, Serial number etc.).

Table 6

User Story 3B: Lend device (choosing from QR Code)

Criteria	**Lend device from QR Code**
Description	As a User of the Inventory App, I want to lend a device by capturing its QR-Code.
Acceptance Criteria	* The QR-Code Button at the Action-Bar at the top of the screen was pressed
Definition of Done	* When QR-Code Reader is opened, the QR-Code is placed in the rectangle-area of the screen.

Table 7

User Story 4A: Bring back device (choosing from List)

Criteria	**Bring Back device (choosing from List)**
Description	As a User of the iNventory App, I want to lend a device by choosing it from the list.
Acceptance Criteria	* Device is chosen from the DeviceListView.
Definition of Done	* New View opens which shows the chosen device including most of its information (available or not, current owner, name, location, Serialnumber etc.).

Table 8

User Story 4B: Bring back a lent device (with QR-Code)

Criteria	**Bring back a lent device (with QR-Code)**
Description	As a User of the iNventory App, I want to bring back a lent device by choosing it from the list.
Acceptance Criteria	* The QR-Code Button at the Action-Bar at the top of the screen was pressed.
Definition of Done	* When QR-Code Reader is opened, the QR-Code is placed in the rectangle-area of the screen.

Table 9

User Story 5: Edit Login-Information

Criteria	**Edit Login-Information**
Description	As a User of the Inventory App I want to change Login-Information (Username, Password, Server-Address).
Acceptance Criteria	* The Settings-Button is pressed and the “Login-Info” entry is chosen.
Definition of Done	* In the shown view the Login-Information is edited.

Table 10

User Story 6: Add a new device.

Criteria	**Add new device**
Description	As a User of the Inventory App I want to

	add a new device to the Inventory.
Acceptance Criteria	* The Settings-Button is pressed and the "Add device"- entry is chosen.
Definition of Done	* In the shown view all of the necessary information is given (DeviceName, Location, SerialNumber, Description, etc.)

Table 11

User Story 7: Edit Login-Information

Criteria	**Filter free devices**
Description	As a User of the iNventory App I want to see all free devices in the inventory.
Acceptance Criteria	* The Filter-Button is pressed and the "Free devices" filter is chosen.
Definition of Done	In the Device List window only free items show.

3.3 Program flow

For the designing of the application flow, Apple has a main file called Storyboard. In here, the application's flow is created graphically to set the structure of the whole application. Figure 4 shows the KMU Inventory App Storyboard. Each number in the image represents a sub flow in the application and they are explained next.

The application's main flow starts with the Log In screen. Then, it continues in a linear flow to a Tab bar controller. This is the center of the application's structure. The Tab Bar Controller handles all the sub flows:

1- Sub flow to add a new device

2- Sub flow to show information of the application

3- Sub flow of scanning a QR Code

4- Sub flow to control the view of the list of the products in the inventory

5- Sub flow to handle the access to login settings once the user is logged

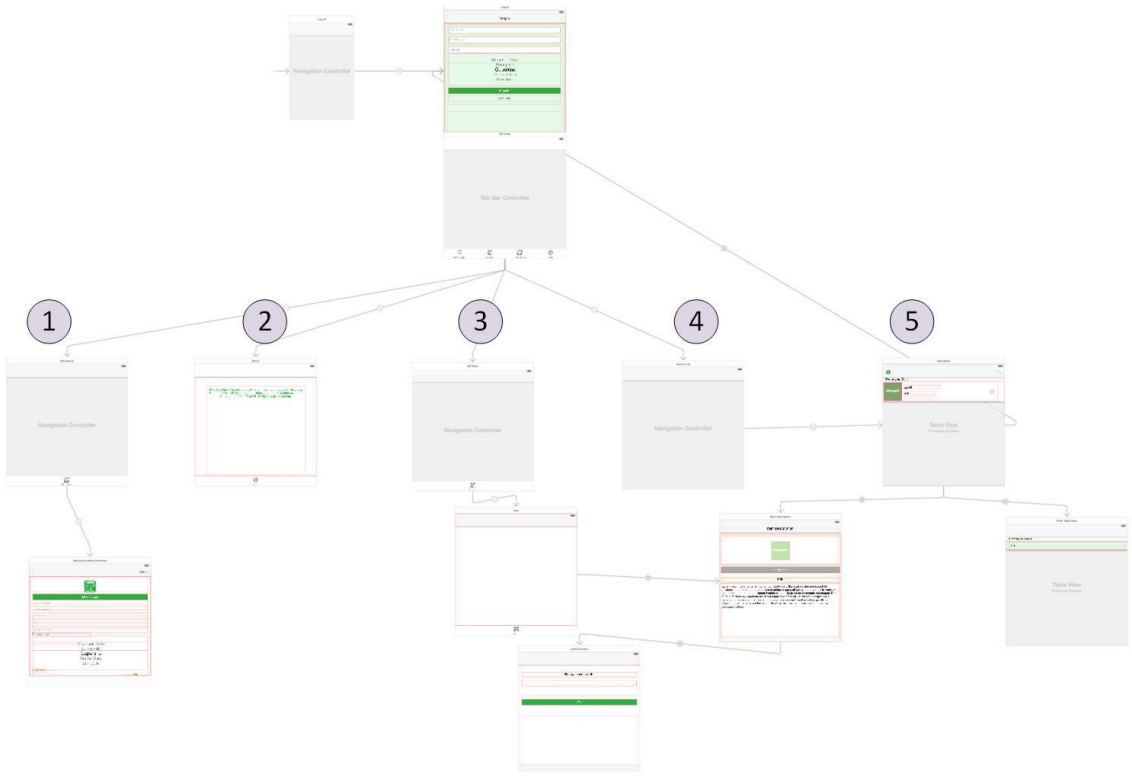

Figure 4. Application's program Flow

3.4 Block diagram

As a mobile application communicating with a server, the application is separated in two blocks, the backend and the frontend. The backend is an element that was designed previously. Therefore, the elements shown in figure 5 are just the ones that are being used in this project. The server backend is managed using Joomla!! It was already set and running by the time the project started.

For the description of the frontend, the block diagram of the application can be consulted on figure 5. It also shows that the iOS application includes nine modules:

1. KMU Server API Interface: This block handles all calls and communication between the mobile application and the KMU Server.
2. User Interface: Default user interface elements used by Swift to show the graphics on the phone screen.

3. View Controllers: Elements that control the behavior of the application and the views depending on the interaction between the user and the application.

4. QR Code Library: This is the library that handles the decoding of the QR code the camera captures.

5. QR Code Reader: This block is the only one that can access the QR Code library. The decision of implementing this extra block is made because it works as a wrapper in case any changes are made by the library creators.

6. Camera Interface: It handles all the camera accesses and ensures that only one instance is running at the same time.

7. Web Client Library: This block contains all the functions to the web services usage. The KMU Server API Interface is the only one that uses these functions.

8. Key Chain Service: This block manages the password storage and keeps it encrypted.

9. Session Settings: It handles current and previous sessions setting and storages.

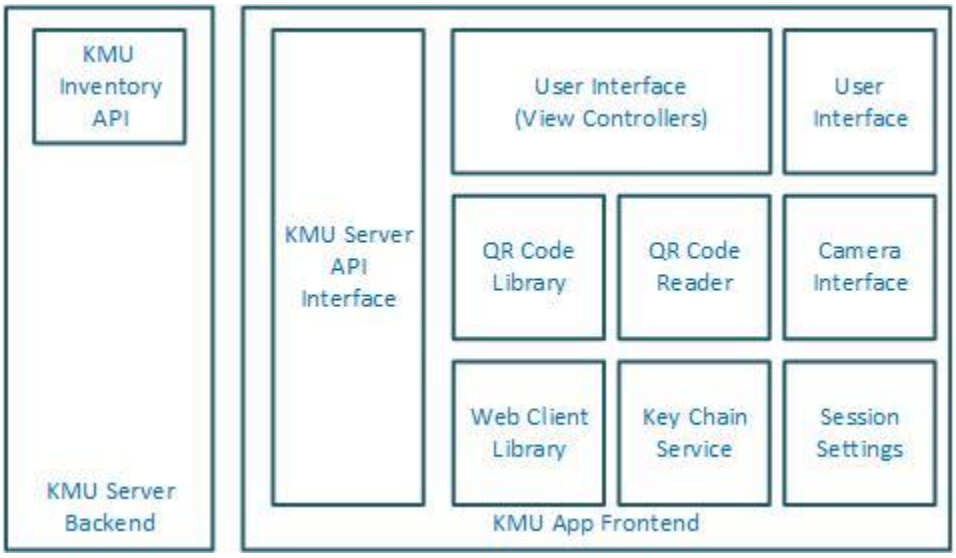

Figure 5. Application Block Diagram

Even though this step was a requirement for the development process, it was not significantly functional during the application's development, analysis, or documentation

stages.

3.5 Software Design Patterns

For the message passing between View Controllers and the navigation process of the application, communication design patterns were used in the programming classes. They were explained as follows:

3.5.1 Singleton

It is used by the KMU Server API interface to make sure every connection to the backend is handled by the same instance. The web server information, the user and login information (not the password) are stored in this instance during the application running.

This pattern is chosen because during the first approach all calls were handled by the class that needed to do the call. This made it necessary for the whole code to be rewritten in every class. On the second approach, only one class was created to handle all API calls, but there were cases in which several classes tried to make a server call, creating replicated instances of the API calls class. Finally, to make sure there was just one instance of the class and centralize the information, Singleton pattern was selected.

3.5.2 Observer

Every View Controller needs an observer pattern design to be notified when a user interacts with the application interface. This is done automatically by Swift, but it is also implemented in the QR reader and the camera handling View Controllers. Once the user finished using this feature, the class informs the parent VC that the user is ready and the application can continue.

3.5.3 Servant

The servant pattern is used only by View Controllers that are created by another View Controller. This is used to pass parameters between them. Especially when one is being initialized using the information the parent gathered. This is a standard pattern used in Swift's architecture.

3.6 Communicating with the server

The used server is located in: https://innsbruck.fh-joanneum.at.

The Backend used is referenced in: https://innsbruck.fh-joanneum.at/kmudev/index.php and is running by Joomla!!(c)

The list of calls used by the application are:

- Get device by ID
- Get all devices
- Get all orgUnits
- Get all devices lent for users
- Get a device by its QRCode
- Lend a device by its QRCode
- Return a device by its QRCode
- Return devices pictures.

3.7 Hardware Libraries

Swift main feature is safety in code programming. As a consequence, the access to hardware features of Apple's devices using this language is impossible. There are no direct ways. The AVFoundation API needs to be used. The camera devices presented in the mobile can be accessed and controlled using an AVCaptureSession. Therefore, it is a simple step that does not require any further hardware approach.

Due to language duality, a library for QR Code Scanner written in Objective C was used. It was published and distributed freely by Yannic Loriot (Loriot, 2014).

3.8 User Interface

The implemented user interface is shown in the following screenshots from figures 6 to 15. Figure 6 is the view of the login screen. The first entry space allows the user to insert the username. The second one is for the password. Finally, the last one is

the adress of the server to connect. The dropdown menu is for choosing the organizational unit of the devices to be displayed.

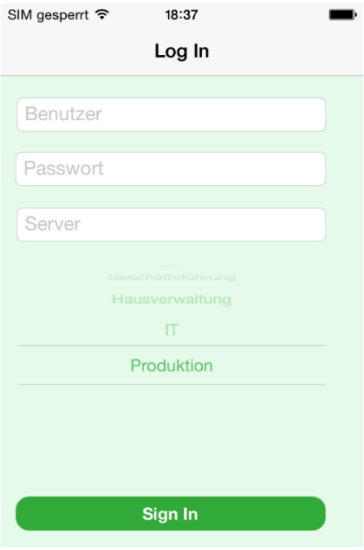

Figure 6. Login View

Figure 7 shows the main view from the list of products. The red frame around the device picture symbolizes that the device cannot be borrowed or returned because it is being used by another user. Then, the green frame means it is available. Finally, the orange frame means the user has borrowed the device.

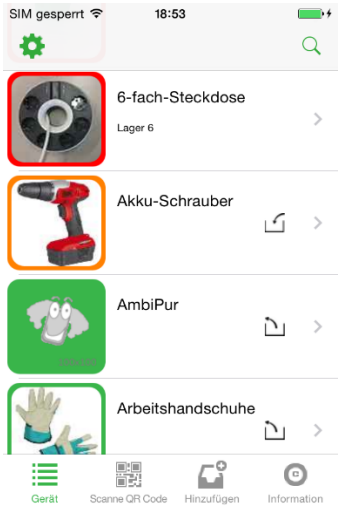

Figure 7. Main view from the list of products

Figure 8 shows the view of the description of a product borrowed by the user.

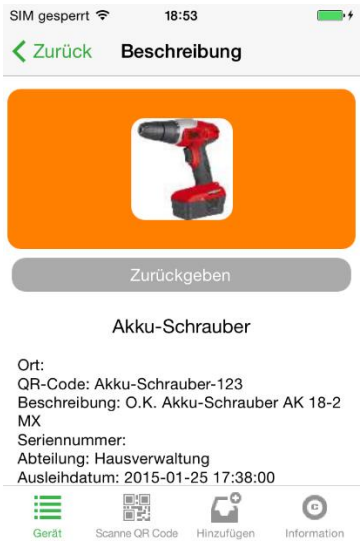

Figure 8. View from a product borrowed by the user.

Figure 9 shows the message displayed when a device is returned by the user.

Figure 9. Message: returned by the user.

Figure 10 shows the view from add product. Figure 11 is the screenshot of the picture being taken to add the new product.

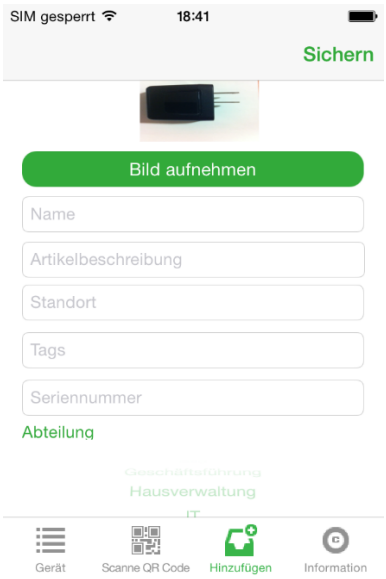

Figure 10. Add product view.

Figure 11. Take picture view.

Figure 12 shows the view from an available product. Figure 13 shows the view from an available product comments section when the user is borrowing it. Figure 14 shows the view from a product borrowed successfully.

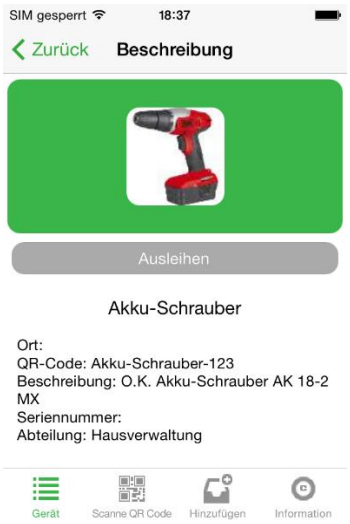

Figure 12. Available product view.

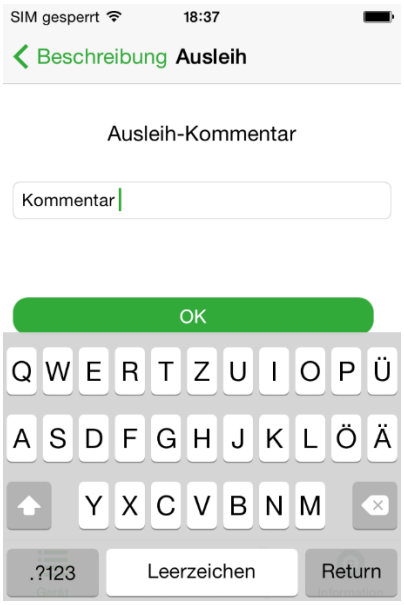

Figure 13. Comment view.

Figure 14. Successfully borrowed device message.

Figure 15 shows the list of products using the Filter Available. So, all products that are displayed have to be framed on green, meaning they are available.

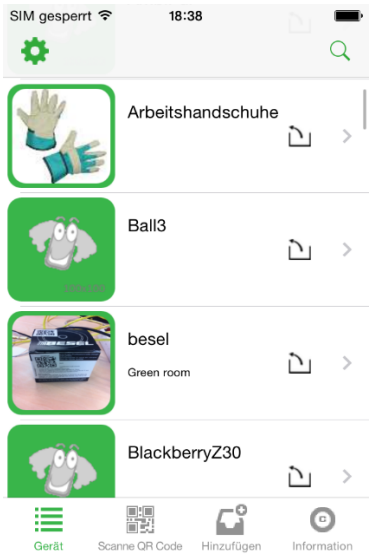

Figure 15. Filter Available implemented

3.9 Application Performance Measurement

The behavior of this application memory usage is tested at the end of every User Story consecutively because Swift does not allow manual memory management. Also a final average consumption is measured.

The CPU usage is also measured in these experiments to provide an overview of the impact that KMU Inventory has on the device. It is measured in terms of its capability percentage. The maximum peak of CPU usage during the process of the User Story is the one presented in the table of results. Having the results of just one application does not give an accurate characterization. Consequently, the older KMU application written in Objective C is tested as well.

The memory usage and CPU performance for each User Story were traced using the XCode tool and Apple Instrument: Activity Monitor. All measurements can be consulted in the Apendix I.

During this measurement, the following assumptions were done:

-Considering the relation between user stories. It is not possible to load a single one without running the previous one. The memory usage and CPU performance are measured per run specified in the table.

-The process of creating a device is not a feature of the Objective C application and cannot be compared. For this reason, US 6 is not taken into account on these experiments.

Memory and CPU behavior

Measurements of memory usage in Swift application and CPU load were made (see table 12 and table 13). Also, the same tests were done on an Objective C application developed by the KMU laboratory on 2013 (see table 14 and table 15). These tests were ran using an iPhone 4.

Table 12

Swift application peak memory usage.

User Story	**Memory Peak Usage (MB)**
US 1	4.02

US1 US2	4.33
US1 US2 US 3 A	7.63
US1 US2 US 3 B	7.93
US1 US2 US 4 A	7.26
US1 US2 US 4 B	7.48
US1 US2 US5	6.63
US1 US2 US7	6.47

Table 13

CPU peak usage by Swift application.

User Story	**Peak CPU Usage by the application (%)**
US 1	36.4
US2	60
US1 US2 US 3 A	65
US1 US2 US 3 B	60
US1 US2 US 4 A	68.8
US1 US2 US 4 B	62.3
US1 US2 US5	55.2
US1 US2 US7	63.5

Table 14

Objective C application peak memory usage.

User Story	**Peak Memory Usage (MB)**
US 1	3.48

US 2	5.75
US1 US2 US 3 A	8.64
US1 US2 US 3 B	6.66
US1 US2 US 4 A	9.97
US1 US2 US 4 B	6.78
US1 US2 US5	7.81
US1 US2 US7	8.33

Table 15

CPU peak usage by Objective C application.

User Story	**Peak CPU Usage (%)**
US 1	30.4
US2	46.1
US1 US2 US 3 A	44.2
US1 US2 US 3 B	59.9
US1 US2 US 4 A	46.9
US1 US2 US 4 B	58.4
US1 US2 US5	44.6
US1 US2 US7	43.2

3.10 ApplicationUsability Testing Strategies

The IT department of the FH Joanneum has a User Experience Laboratory. The KMU Inventory App is tested by the laboratory using it as an advantage for mobile testing and the application evaluation. The result handled to the KMU GOES Mobile team is:

Tester: Matthias Eckhart

Test Report -iOS App Device: iPhone 4, iPad 4

US1 Status: passed Notes: - Please add a message after successful login in German: Login erfolgreich

US2 Status: passed

US3a Status: passed Notes: - Please add whitespace between “Ausgeliehen von” (“Lent by”) and - Please add colon between “Abteilung” (“Organization unit”)

US3b Status: passed

US4a Status: passed Notes: - Please change the return message from “user credentials valid, device returned” to: o German: Gegenstand wurde erfolgreich zurückgegeben.

US4b Status: passed US5 Status: passed US6 Status: passed.

3.11 Application and language experience documentation

This section is constructed by the knowledge acquired during the investigation described in section 3.1 of this document and the experience during the development of the application as described in section 3.2.

The first document is a guideline created to work as a reference framework for future beginners in Swift programming language or to anyone interested in the process of developing an application in this language. The second one is a documentation of the language’s learning process that is located on a wikibook available online. It is meant to be modified and improved by people from all over the world.

3.12 Swift's Guideline

This guideline is based on the skills the developer required to build the test application described in the previous sections. The values are obtained from the experimental application development using the following procedure:

1. All needed skills during the platform environment setting, the tutorials, and the programming stages were documented. The needed skills were:

- Apple Knowledge
    - Mac OS experience as user
    - Encoding and keychain management previous experience
    - Apple's auto size feature experience
    - Cocoa Framework knowledge
    - Having released an application to the Apple Store before
- Programming experience in any language
    - Software Design Patterns knowledge
    - Experience in object oriented programming
    - Knowledge or experience in Graphic Programming
- Mobile experience
    - Objective C knowledge
    - Basic knowledge of mobile structure
    - Backend-frontend structure
    - Server connection with a mobile platform experience
    - Library import and usage knowledge
- Settings and Access
    - Backend for the application is already set
    - iOS programming environment is already set
- CPU and memory behavior interpretation
    - Knowledge of CPU and memory behavior in mobile devices
    - Interpretation of memory leaks
- User experience/aesthetics knowledge

- Experience as a designer or with Apple's design rules

2. The number of hours the programmer took to learn the new skills was documented and can be seen on table 16. If the skill was not new to the programmer, then a zero goes in the documented hours.
3. The number of times the skill was needed was also documented as:
    - Once: One time. The value for this in the table is 1.
    - Twice: Two times. The value for this in the table is 2.
    - A few: three to six times. The value for this in the table is 3.
    - Several: six to nine times. The value for this in the table is 4.
    - Always: more than nine times. The value for this in the table is 5.

As before starting the project the programmer already had some of the listed skills, an approximate time for learning them is substituted in table 16 in step 2. The metric used to obtain the values (see table 16) of the following procedure is:

Calculation of weights:

The variables present are:

C: Amount of times the skill is needed by the programmer.

T: Time required learning a specific skill.

F: The factor that characterizes the weight of a skill.

M: The multiplier defining the value of every unit of the factor.

W: The weight the skill has on the guideline.

The values C and T are multiplied to obtain the factor F that will characterize the skill as the following formula shows:

$$F_k = C_k t_k$$

Then all factors from all skills are added and divided by 100 to get the multiplier M that defines the value of a unit of the factor:

$$M = \frac{1}{100} \sum_{k=0}^{n} F_k$$

Then every weight of each skill is calculated as follows:

$$W_k = MF_k$$

Table 16

C Swift beginner's table for guideline estimation

Skill	Learning hours (t)	Times used (C)	Factor (F)	Skill weight (W)
Mac OS experience as user.	24	5	120	6.05
Encoding and keychain management previous experience.	16	1	16	0.81
Apple's auto size feature experience.	64	5	320	16.13
Cocoa Framework knowledge.	40	5	200	10.08
Having released an application to the Apple Store before.	64	3	192	9.68
Software Design Patterns knowledge.	32	5	160	8.06
Experience in object oriented programming.	32	5	160	8.06
Knowledge or experience in Graphic Programming.	8	5	40	2.02
Objective C basic/beginners knowledge.	16	1	16	0.81
Basic knowledge of mobile structure.	40	5	200	10.08
Backend-frontend structure	24	4	96	4.84
Server connection with a mobile platform experience.	24	3	72	3.63
Library import and usage knowledge.	8	3	24	1.21
iOS programming environment is already set.	5	40	1	40
Knowledge of CPU and memory behavior in mobile devices.	1	8	3	24
Interpretation of memory leaks.	2	16	3	48
Experience as a designer or with Apple's design rules.	64	4	256	12.90
Total	520	61	1984	100

3.12.1 Wikibook

The wikibook created is stored in Wikibooks English web page hosted on the following link: http://en.wikibooks.org/wiki/Swift_learning. The information contained there is listed here:

1. This wikibook
2. Basic Concepts
   2.1. What is Swift?
   2.2. Main differences between Swift and Objective C
   2.3. Swift Navigation Architecture (Windows and Views)
3. Programming
   3.1. XCode
   3.2. Start Programming
   3.3. Adding elements to your project
   3.4. Configuring for development
   3.5. Configuring for deployment
4. Tricky Section
   4.1. Auto Layout
   4.2. Communication between View Controllers (Design Patterns)
   4.3. Manage of Segues
      4.3.1. Create-Call segues
      4.3.2. Dismiss a Segue
5. Next Steps
   5.1. Other worth to see Web Sites

This website contains the developer's result notes during the learning curve, making it a manual for Swift's first usage. It also includes information about the tools needed during this project.

4 Results Analysis

The results of this project can be divided into two groups. (1) The experimental development of the mobile application, and (2) The information gathered about the language. Thus, the analysis of these results is divided by these criteria.

4.1 Experimental Application Implementation

4.1.1 Application requirements

The application requirements, including all user stories, were implemented successfully. The results from the user stories are shown in this section. Also, the new requirements added by the client during the development process are explained here.

Screenshots of the application are included to show each User Story implementation of the applications. (see figures 16 to 23.)

Figure 16. Screenshots from US1 and US2

US3A is implemented in the same way as US3B, but the difference is the initiating screen. The second, third, and fourth views were called and used in the same way in both user stories, implementing code recycling for these stages.

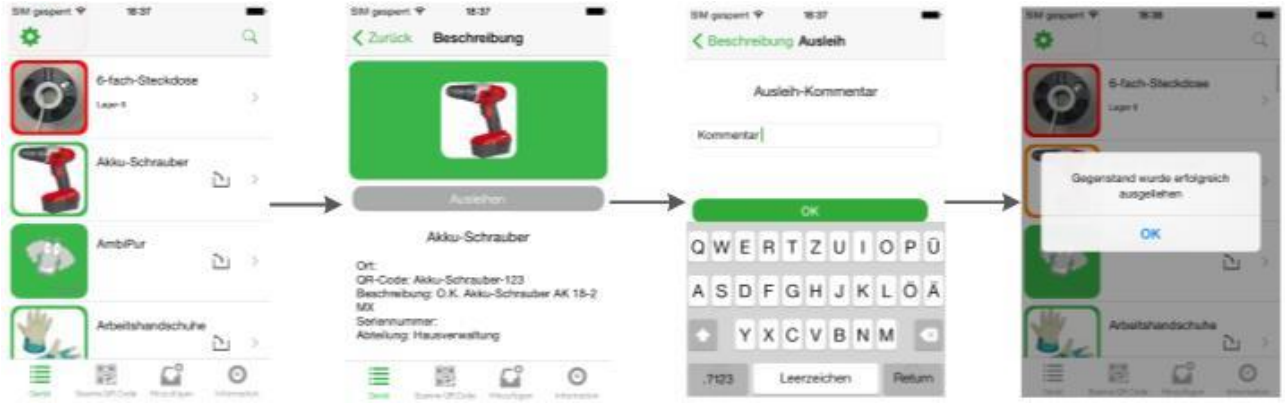

Figure 17 Screenshots from US 3A

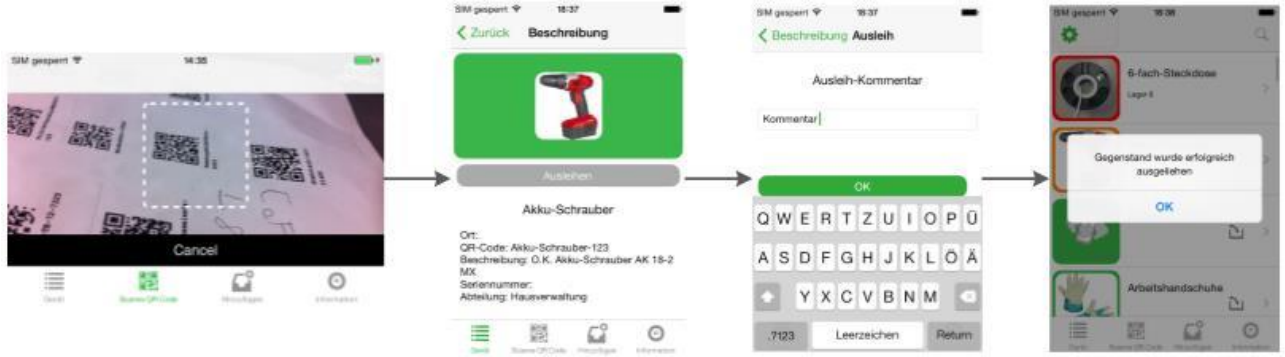

Figure 18 Screenshots from US 3B

As US3A and US3B, US4A and US4B reuse the code of the second and third view, it necessary to build just one structure in the Storyboard.

Figure 19 Screenshots from US 4A

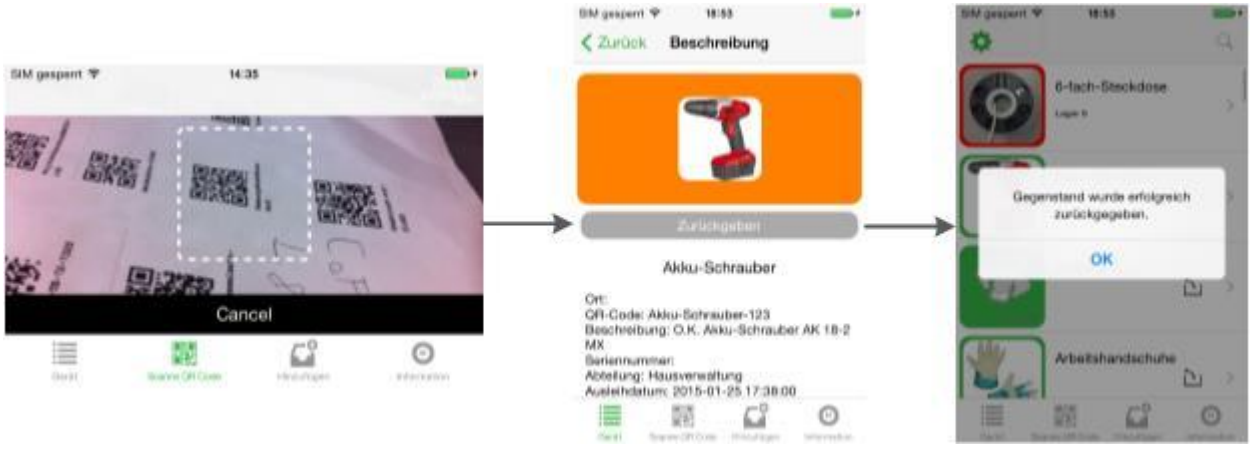

Figure 20 Screenshots from US 4B

US5 is called from the top left corner button shown on figure 21. It opens the login view to test the connection.

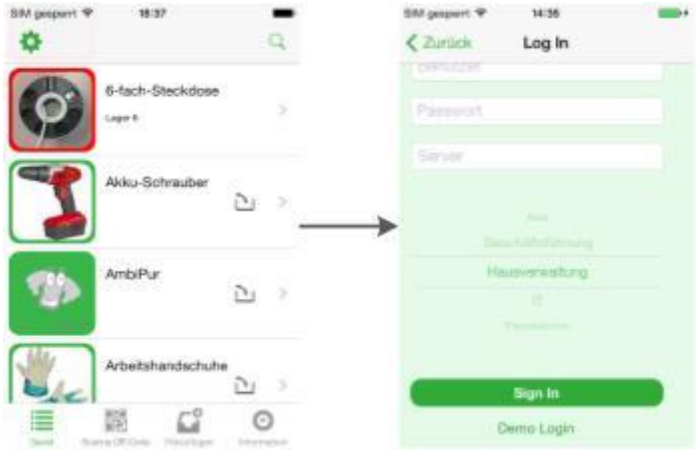

Figure 21 Screenshots from US 5

In US6 (see figure 22), it can be noticed that the picture taken is resized in a not suitable way for the space available. Even though, it just happens sometimes, the error could not be corrected unfortunately.

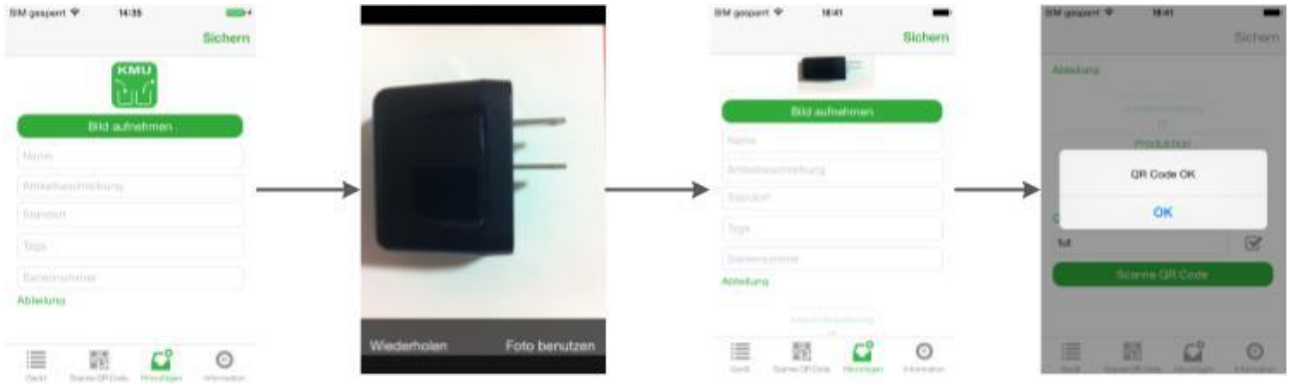

Figure 22 Screenshots from US 6

Figure 23 Screenshots from US 7

During the development process two new requirements were included. The About Window and a Filter for the products list. The About Window requirement was included in the Main flow as an extra sub flow and a new view was added to the design. (See figure 24)

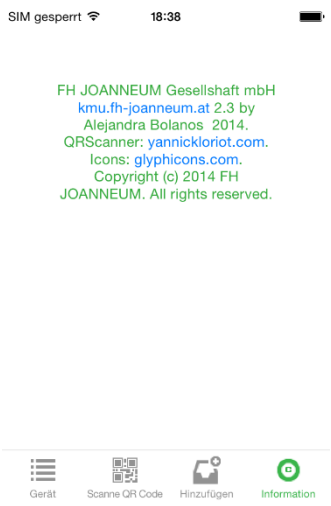

Figure 24. About View

The Filter was implemented by adding an option in the top-right description of the main list view. Figure 25 shows the menu displayed when the filter button is pressed. The options to filter in the order shown on the figure are: All products, Products borrowed by me, Available products, Not available products, by organization: All, Business, Home Administration and IT. Figure 26 shows the selection of the filter: Borrowed by the user.

Figure 25. Menu

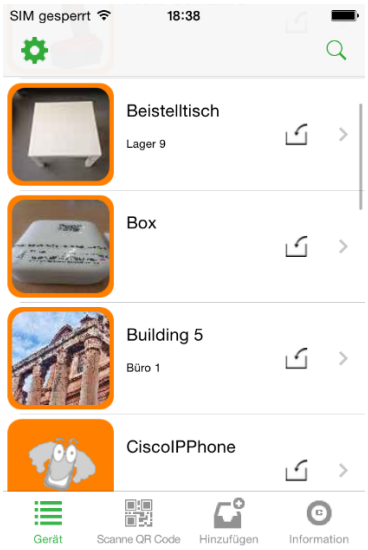

Figure 26. Filter Borrowed by the user selected

4.1.2 Performance measurements

For a better view of the results of the memory and CPU behavior of the application, the graphics of figure 27 and figure 28 provide the data of the comparison. The memory usage and CPU load between the Swift application and the Objective C application.

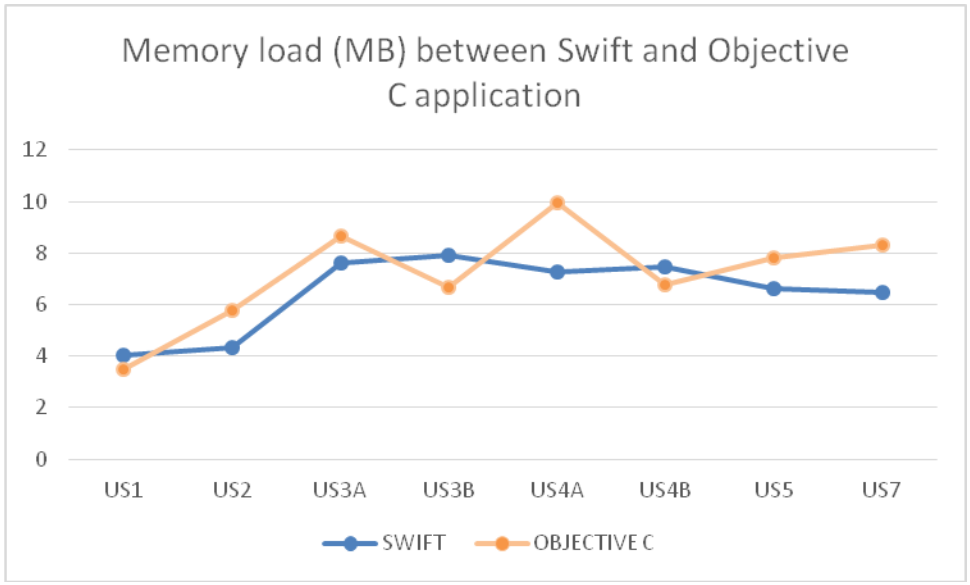

Figure 27. Memory load in Swift vs Objective C application

Memory consumption tendency is lower using Swift than Objective C in all of the user stories, but in US3B and US4B. It can be observed that these two cases are the ones using the camera during the QRCode scan and decoding. This implementation is using more memory resources than the application made before in Objective C, it but cannot be optimized because the camera functionality is programmed through an external library.

The QR Code scan library used is written in Objective C, but it is not the same used in the Objective C application. This could mean that the way the library is programmed is less efficient than the one used in the Objective C application.

It can also be observed that the maximum memory consumption ever made by the application in Swift is 8MB. As considered by Apple, it is a safe amount of memory consumption for a mobile phone and makes the application suitable for its usage.

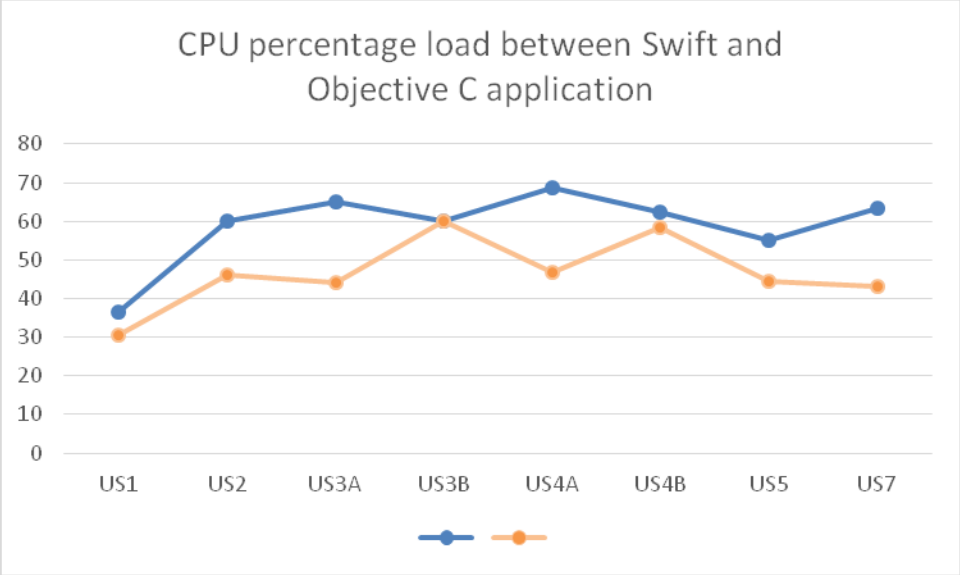

Figure 28. CPU load in Swift vs Objective C application

In figure 28, the CPU load in Swift and Objective C applications are compared. It is clearly observed that Swift application is consuming more CPU processing percentage than the implementation made in Objective C. The tendency is that the consumption of the application written in Swift is higher than the Objective C application when ran in the same device.

Findings made by Jesse Squires in his article cited (Squires, 2014) suggest that as the application is not implemented removing all safety features (array bounds-checking, integer overflow checking, etc.), the expected result is that the application would be slower than the Objective C one. This situation will worsen the numbers because an increase of CPU load indicates that the power consumption should be also higher. It makes Swift application spend more resources than the version programmed in Objective C.

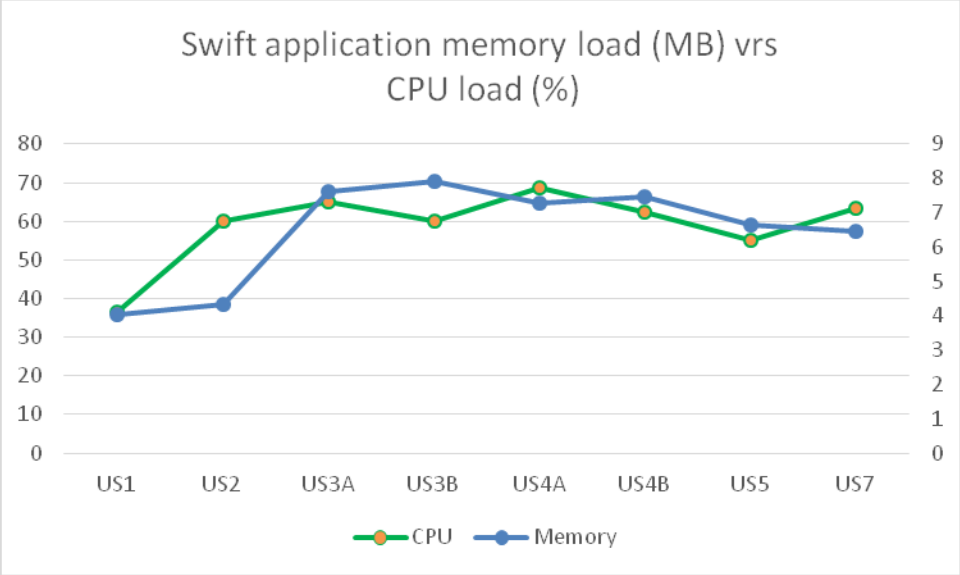

Figure 29. CPU load vs Memory load in Swift application.

The relation between memory usage and CPU consumption (See figure 29) does not show any particular behavior that can lead to any conclusion.

4.1.3 Usability test

The corrections suggested in the test were made in the string document of the application and a second test was made after the bugs were corrected. All US passed the test. This indicates the KMU Inventory app is suitable for mobile users for its first approach.

4.1.4 Application upload to the store

The previous version of the applications had no problems when uploading to the Apple Store, but the final version did. The problem found in this stage is described as follows:

Once all tests ran correctly in the simulators and the devices, the application was uploaded to the iTunes Store for its final review and got rejected with the following message:

"Apps that crash will be rejected.

During review, your app crashed on iPad running iOS 8.1.2 when we:

1. Launch the app

2. Select "Demo Login"

3. App crashes

This occurred when your app was used:

- Offline

- On Wi-Fi

- On cellular network

Next Steps

Please revise your app and test it on a device to ensure that it runs as expected."

When this fault was analyzed to be corrected, it was found that it only happened if the optimization was enabled. The C-API for Keychain Access used to encrypt the password makes a call:

let opaque = dataTypeRef?.toOpaque()

This call returned nil and to solve. The optimization had to be disabled. This is a bug already reported to the language developers and it is a frequently asked question in Apple's forums.

4.2 Language learning documentation and Guideline

4.2.1 Guideline

The guideline (see figure 30) was originally created taking into account just the time it took to learn the ability. Afterwards, it included the amount of times it was used.

A linear function was chosen to maintain simplicity. Every variable has a linear influence on the guideline weights. The weight system is included in this guideline to make the criteria easily measurable for anyone that needs to plan a project. It also propitiates to add future improvements based on measurable variables of the development environment in a very organized way,

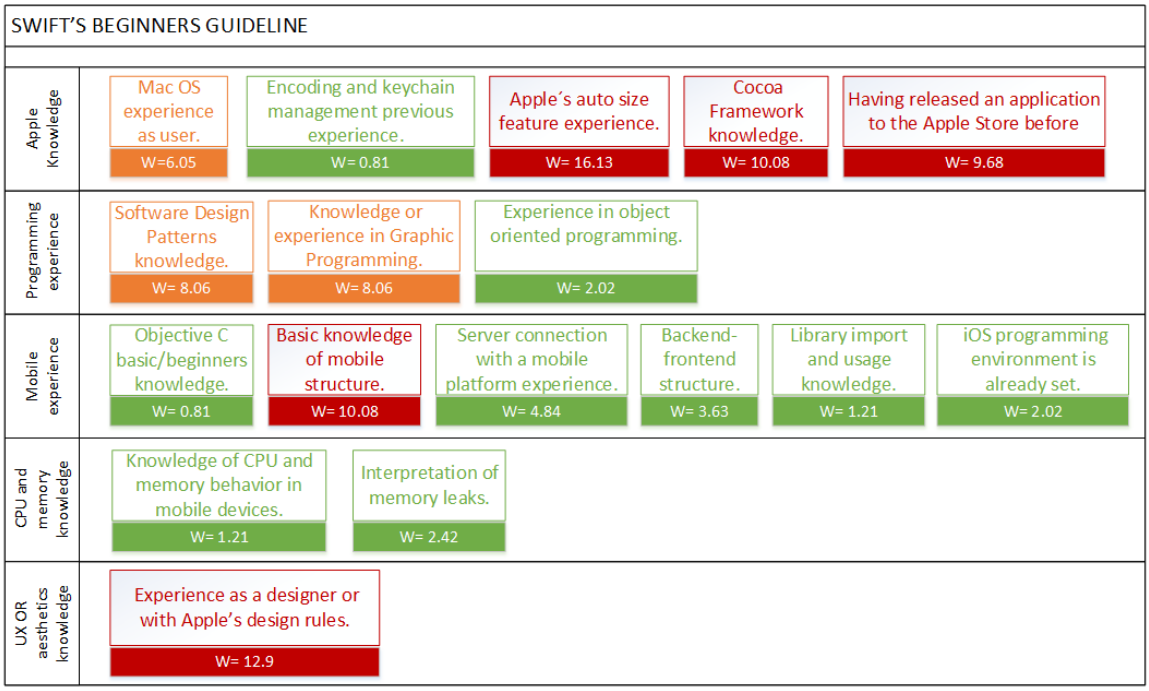

Figure 30. Swift's guideline

How to use this Guideline:

•Every skill has a weight and all weights together sum up 100.

•The skills in green are not so important, the skills in orange are important, and the skills in red are crucial.

•This guideline is meant for a learning process of 3.25 months or 65 days. The 100 of the weight represent the full time.

This guideline helps to calculate the learning time of Swift language and the programming of an application. It is a reference framework for future developers and managers incurring in Swift technology to set a development strategy.

The criteria documented in this guideline are validated by the experience documented through the development of the experimental application for KMU Inventory. In this way, a guideline for future skills and time planning of Swift learning process can be consulted in an understandable way.

4.2.2 Wikibook

The learning document is a guide for new programmers in Swift language uploaded to the Wikibooks site. It is available for modification from anyone. It makes it a suitable place for information about best practices and suggested procedures to learn the programming language. This work also encourages the exchange of information of the Swift community.

4.2.3 Future Work

The evaluation of CPU consumption and memory in this investigation has been developed using Swift safety feature activated. A further step could be evaluating the same application removing this feature to prove the efficiency gained or lost.

As Swift is a constantly changing language, a future evaluation comparing the language to Objective C could result in a very useful piece of information and even the results can change drastically. It could keep track of the language evolution, and at the same time, change the low efficiency language image this investigation gives.

Feedback from other developers about their timing and learning experience of Swift can improve the weights of the guideline. It would lead to more exact criteria of skills and time needed in this process.

5 Conclusions and Recommendations

5.1 Conclusions

-The implementation of the mobile application KMU Inventory app worked as a reference experiment to create a guideline for future developers and investigators in Swift language.

-A weight function was built to work as a reference for future developers and managers incurring in Swift technology to set a development strategy for the time and skills needed. This criterion was validated through the development of the KMU Inventory application using Swift language.

-The experience of developing the document about Swift language in a wikibook to be used as a learning document for guiding new programmers contributes to the programming community sharing knowledge and helping beginners in the area.

- In Apple's standards, the application implemented in Swift language does consume a safe amount of the device memory. That is why it is safe for deploying it to the Apple store.

-The comparison between memory usage and CPU consumption did not provide a predictable result; however, this information could be studied to explain this behavior.

-Even though, in average, Swift implementation consumes less memory than Objective C implementation, Objective C implementation of the KMU Inventory app is less power consuming than the Swift implementation done in this project. There is a tradeoff between these measurements.

-The user experience test shows that the UX first approach made in the application KMU Inventory is suitable for mobile users, but it could be improved using feedback from users and clients.

-Future works can improve this project guideline weights and criteria using the time and skills needed by new developers as new inputs. It could make the investigation more trustworthy and stable.

5.2 Recommendations

There can be misunderstandings when working with clients speaking a different language. Also, it changes the requirements. To mitigate this risk, there has to be a frequent meeting to show the advances and products being developed.

The storyboard should always be the first feature implemented when programming a mobile application in Swift language. It shows the structure of the program and makes the flow of the application clearer.

It is easier to use the tool provided by XCode to measure CPU usage and memory consumption than the device's task manager. It will not only provide the results directly on the computer, but also in the mobile device.

When learning a new language, the creation of an application from scratch leads the developer to a deeper understanding of the languages functionalities and features.

When programming to upload an application to the Apple Store, a good understanding of the user experience makes the work faster. It is possible that the developer does not have experience in this area. Thus, a study of Apple's rules and suggestions should be done.

References

Anderson, S (2010). *Seductive Interaction Design*. Berkeley, CA, USA: New Riders.

Apple Inc (2014a). iOS Human Interface Guidelines *iOS Developer Library*. Retrieved February 22, 2015 from https://developer.apple.com/library/ios/documentation/UserExperience/Conceptual

Apple Inc (n.d.b). IPad size class [Figure]. Retrieved October 13, 2014, from developer.apple.com/swift/.

Apple Inc (2014c). Swift *iOS Developer Library*. Retrieved October 13, 2014, from developer.apple.com/swift/.

Apple Inc (2014d). XCode *iOS Developer Library*. Retrieved February 10, 2015 from https://developer.apple.com/xcode/

FH JOANNEUM (n.d). *About US*. Retrieved October 15, 2014 from http://www.fh-joanneum.at/aw/home/~cwd/leitbild/?lan=en

Hanssen, A. (2014). Swift vs. Objective C. [Blog] One Month. Retrieved September 21 from https://onemonth.com/blog/swift-vs-Objective C.

Keiser, G (2015). iPhone 6 boosts iOS market share. Retrieved February 22 from http://www.computerworld.com/article/2866441/iphone-6-boosts-ios-market-share-android-slips.html

Kohs, G (2010). *What_is_Wikibooks*. Retrieved February 2, 2015 from http://en.wikibooks.org/wiki/Wikibooks:What_is_Wikibooks

Loriot, Y (2014). QR Code Reader. Retrieved November 20, 2014 from https://github.com/yannickl/QRCodeReader.swift

Nielsen, J (1993). Usability Engineering. San Francisco, (CA): Morgan Kaufman

Squires, J (2014). *Apples to apples* Jesse Squires. Retrieved September 10 from http://www.jessesquires.com/apples-to-apples/

Wilson, C (2010) *User Experience*. Burlington, MA, USA: Morgan Kaufman

Appendix

Appendix I: Swift Memory Measurements

Table 17

Swift memory measurements 1.

US 1 Process ID	**Peak Memory Usage (MB)**	**Lowest Memory Usage (MB)**	**Other process usage (MB)**
7805	4	4	280,8
7807	4	2,8	289,8
7811	4	0,92	289,5
7816	4	3,5	289,9
7820	4	2,7	289,8
7823	4	2,9	289,7
7827	4,1	1,9	289,7
7830	4	2,9	289,7
7847	4,1	1,8	288,4
7856	4	2,1	288,6

Table 18

Swift memory measurements 2.

US1 US2 Process ID	**Peak Memory Usage (MB)**	**Lowest Memory Usage (MB)**	**Other process usage (MB)**
7863	5,3	0,9	288,0
7867	5,4	2,6	288,1
7871	5,3	4,1	287,8
7876	5,4	3,5	288
7879	5,3	0,45	288,1
7883	5,3	0,89	288

7887	5,3	2,9	287,9
7890	5,3	2,9	287,7
7895	5,4	1,6	287,9
7899	5,3	0,96	288,1

Table 19

Swift memory measurements 3.

US1 US2 US3 A Process ID	**Peak Memory Usage (MB)**	**Lowest Memory Usage (MB)**	**Other process usage (MB)**
7812	7,3	2,7	294,4
7816	7,7	0,98	294,4
7820	7,8	3,5	290,7
7826	8,4	2,9	289,7
7830	7,6	2,8	290,5
7835	7,6	2,1	290,5
7839	7,2	2,8	290,7
7843	7,3	1,7	290,7
7846	7,6	1,4	289,9
7851	7,8	3,6	290,8

Table 20

Swift memory measurements 4.

US1 US2 US3 B Process ID	**Peak Memory Usage (MB)**	**Lowest Memory Usage (MB)**	**Other process usage (MB)**
7855	7,6	3	290,6
7859	7,5	3	290,4
7863	7,4	0,48	289,9
7878	7,5	2,6	290,7

7872	8	0,84	290,7
7875	8,9	2,9	290,4
7879	8,2	0,45	290,7
7884	8,2	1,7	289,7
7888	7,6	0,94	289,4
7893	8,4	0,45	289,2

Table 21

Swift memory measurements 5.

US1 US2 US4 A Process ID	**Peak Memory Usage (MB)**	**Lowest Memory Usage (MB)**	**Other process usage (MB)**
8003	7,3	3,6	307,9
8008	7,4	1,4	307,6
8013	7	1,6	305,3
8016	7,3	3,6	306,5
8022	7,3	1,9	307,8
8026	7	3	306,7
8030	7	0,45	306,4
8035	7,4	3,5	305,4
8039	7,6	0,91	306,2
8043	7,3	0,45	304,9

Table 22

Swift memory measurements 6.

US1 US2 US4 B Process ID	**Peak Memory Usage (MB)**	**Lowest Memory Usage (MB)**	**Other process usage (MB)**
8047	7,2	3,2	307,5
8052	8,3	3,6	304,2

8056	7,3	3	306,2
8060	7,2	2,71	307,1
8066	7,4	2,2	308
8070	7,2	2,9	308,5
8073	7,9	2,9	304,8
8077	7,3	2,7	305
8082	7,6	2,1	290,9
8086	7,4	0,78	308,1

Table 23

Swift memory measurements 7.

US 5 Process ID	**Peak Memory Usage (MB)**	**Lowest Memory Usage (MB)**	**Other process usage (MB)**
8091	6,7	3	290,7
8096	6,4	2,6	291
8099	6,4	0,48	291,2
8103	6,4	1,02	291,3
8111	8,5	1,8	288,6
8115	6,4	3	289,7
8119	6,3	1,5	289,9
8123	6,3	2,9	289,7
8127	6,5	1,6	289,9
8131	6,4	0,92	289,9

Table 24

Swift memory measurements 8.

US7 Process ID	**Peak Memory Usage (MB)**	**Lowest Memory Usage (MB)**	**Other process usage (MB)**

8135	6,8	0,45	289,9
8139	6,4	0,45	290
8143	6,5	1,3	289,9
8147	6,1	3,1	290
8151	6,8	0,45	290,3
8155	6,4	0,48	290
8158	6,4	4,4	289,8
8163	6,8	3,5	290,1
8167	6,4	3,1	290,2
8171	6,1	4,4	290

Table 25

Swift memory measurements statistics.

Statistic	**US1**	**US2**	**US3A**	**US3B**	**US4A**	**US4B**	**US5**	**US7**
Mean	4.02	5.33	7.63	7.93	7.26	7.48	6.63	6.47
Standard Error	0.01333333	0.01527525	0.10857665	0.15567059	0.06359595	0.11430952	0.21084486	0.08306624
Median	4	5.3	7.6	7.8	7.3	7.35	6.4	6.4
Mode	4	5.3	7.6	7.6	7.3	7.2	6.4	6.4
Standard Deviation	0.0421637	0.04830459	0.34334951	0.49227364	0.20110804	0.36147845	0.66674999	0.26267851
Sample Variance	0.00177778	0.00233333	0.11788889	0.24233333	0.04044444	0.13066667	0.44455556	0.069
Kurtosis	1.40625	-1.22	2.11	-0.19	-0.62	2.10	9.20	-0.98
Skewness	1.77	1.03	1.08	0.78	-0.13	1.61	3.00	0.03
Range	0.1	0.1	1.2	1.5	0.6	1.1	2.2	0.7
Minimum	4	5.3	7.2	7.4	7	7.2	6.3	6.1
Maximum	4.1	5.4	8.4	8.9	7.6	8.3	8.5	6.8
Sum	40.2	53.3	76.3	79.3	72.6	74.8	66.3	64.7
Count	10	10	10	10	10	10	10	10
Confidence Level	4.02	5.33	7.63	7.93	7.26	7.48	6.63	6.47

Appendix II: Swift CPU Measurements

Table 26

Swift CPU measurements 1.

US 1 Process ID	Peak CPU Usage (%)	Lowest CPU Usage (%)	Other processes (%)
7805	15	0	7
7807	40	0	5
7811	68	0	6
7816	24	0	4
7820	35	0	8
7823	31	0	8
7827	66	0	5
7830	26	0	31
7847	26	0	8
7856	33	0	6

Table 27

Swift CPU measurements 2.

US1 US2 Process ID	Peak CPU Usage (%)	Lowest CPU Usage (%)	Other processes (%)
7863	66	0	31
7867	65	0	5
7871	56	0	4
7876	52	0	6
7879	66	0	5
7883	65	0	15
7887	58	0	20
7890	40	0	27
7895	67	0	5

7899	65	0	6

Table 28

Swift CPU measurements 3.

US1 US2 US3 A Process ID	**Peak CPU Usage (%)**	**Lowest CPU Usage (%)**	**Other processes (%)**
7812	67	0	8
7816	68	0	5
7820	77	0	5
7826	54	0	13
7830	56	0	5
7835	71	0	8
7839	52	0	17
7843	55	0	4
7846	65	0	5
7851	85	0	5

Table 29

Swift CPU measurements 4.

US1 US2 US3 B Process ID	**Peak CPU Usage (%)**	**Lowest CPU Usage (%)**	**Other processes (%)**
7855	47	0	8
7859	51	0	7
7863	52	0	15
7878	59	0	6
7872	66	0	17
7875	54	0	5
7879	65	0	53

7884	64	0	15
7888	77	0	5
7893	65	0	5

Table 30

Swift CPU measurements 5.

US1 US2 US4 A Process ID	**Peak CPU Usage (%)**	**Lowest CPU Usage (%)**	**Other processes (%)**
8003	80	0	50
8008	62	0	46
8013	57	0	49
8016	84	0	59
8022	70	0	62
8026	66	0	68
8030	63	0	47
8035	64	0	52
8039	64	0	58
8043	78	0	67

Table 31

Swift CPU measurements 6.

US1 US2 US4 B Process ID	**Peak CPU Usage (%)**	**Lowest CPU Usage (%)**	**Other processes (%)**
8047	73	0	85
8052	72	0	47
8056	50	0	45
8060	56	0	45
8066	72	0	47

8070	60	0	46
8073	40	0	47
8077	66	0	45
8082	68	0	40
8086	66	0	45

Table 32

Swift CPU measurements 7.

US 5 Process ID	**Peak CPU Usage (%)**	**Lowest CPU Usage (%)**	**Other processes (%)**
8091	48	0	6
8096	40	0	5
8099	67	0	15
8103	62	0	5
8111	37	0	6
8115	55	0	7
8119	62	0	19
8123	66	0	6
8127	47	0	7
8131	68	0	5

Table 33

Swift CPU measurements 8.

US7 Process ID	**Peak CPU Usage (%)**	**Lowest CPU Usage (%)**	**Other processes (%)**
8135	63	0	7
8139	79	0	6
8143	69	0	12
8147	44	0	17

8151	64	0	18
8155	63	0	8
8158	68	0	5
8163	71	0	5
8167	48	0	4
8171	66	0	7

Table 34

Swift CPU measurements statistics.

Statistic	**US1**	**US2**	**US3A**	**US3B**	**US4A**	**US4B**	**US5**	**US7**
Mean	36.4	60	65	60	68.8	62.3	55.2	63.5
Standard Error	5.536144	2.748737	3.438346	2.871314	2.819771	3.425558	3.641733	3.290559
Median	32	65	66	61.5	65	66	58.5	65
Mode	26	65	-	65	64	72	62	63
Standard Deviation	17.50682	8.69227	10.873	9.079892	8.9169	10.83256	11.51617	10.40566
Sample Variance	306.4889	75.55556	118.2222	82.44444	79.51111	117.3444	132.6222	108.2778
Kurtosis	0.360351	2.207444	-0.56195	-0.26653	-0.88488	0.479733	-1.43041	0.585705
Skewness	1.122866	-1.57214	0.510205	0.331736	0.646739	-1.07906	-0.4398	-0.82838
Range	53	27	33	30	27	33	31	35
Minimum	15	40	52	47	57	40	37	44
Maximum	68	67	85	77	84	73	68	79
Sum	364	600	650	600	688	623	552	635
Count	10	10	10	10	10	10	10	10
Confidence Level(95.0%)	12.52363	6.218075	7.778079	6.495364	6.378766	7.74915	8.238173	7.443761

Appendix III: Objective C Memory Measurements

Table 35

Objective C memory measurements 1.

US 1 Process ID	**Peak Memory Usage (MB)**	**Lowest Memory Usage (MB)**	**Other process usage (MB)**
7520	3,5	3,5	288,9
7523	3,4	1,7	288,5
7527	3,5	2	288,2
7533	3,4	2,7	288,4
7536	3,5	3,5	288,4
7540	3,5	1,7	289,8
7544	3,5	3,5	288,5
7547	3,5	1,7	288,2
7552	3,5	2,1	288,5
7555	3,5	3,5	288,5

Table 36

Objective C memory measurements 2.

US1 US2 Process ID	**Peak Memory Usage (MB)**	**Lowest Memory Usage (MB)**	**Other process usage (MB)**
7563	5,8	2	286,6
7567	6,5	2,3	285,7
7571	5,8	2,1	286,9
7575	6,6	1,9	286,7
7579	5,8	2,2	285,9
7583	5,8	3,5	285,9
7586	3,4	0,62	287,4

7591	6,8	2,4	285
7595	7,6	2	284,7
7599	3,4	3,4	280,4

Table 37

Objective C memory measurements 3.

US1 US2 US3 A Process ID	**Peak Memory Usage (MB)**	**Lowest Memory Usage (MB)**	**Other process usage (MB)**
7600	8,4	3,5	284,4
7605	9,3	3,4	282,8
7609	8,5	2,8	282,9
7613	8,3	3,4	282,7
7617	7,8	3,4	282,8
7624	8,8	2,3	282,9
7636	8,4	3,4	283,9
7641	8,4	3,4	283,4
7644	8,8	3,4	283
7648	9,7	3,4	282,4

Table 38

Objective C memory measurements 4.

US1 US2 US3 B Process ID	**Peak Memory Usage (MB)**	**Lowest Memory Usage (MB)**	**Other process usage (MB)**
8177	6,4	2	289,2
8182	5,8	1,7	290,2
8185	7,1	3,5	288,4
8190	5,8	3,5	289
8194	7,8	1,8	289,3

8201	6,9	3,5	289,2
8206	6,9	3,5	289,5
8210	7	2,8	289,9
8214	7,1	3,5	288,9
8219	5,8	3,5	290,4

Table 39

Objective C memory measurements 5.

US1 US2 US4 A Process ID	**Peak Memory Usage (MB)**	**Lowest Memory Usage (MB)**	**Other process usage (MB)**
7669	10	3,5	280,7
7678	10	3,5	280
7683	10,2	2,3	279,5
7687	10	3,4	281,6
7690	10,1	3,4	280,3
7696	9,9	2,3	280,6
7699	9,9	3,4	280,4
7704	10	3,4	282
7708	10,1	0,62	281,6
7712	9,5	3,4	277,9

Table 40

Objective C memory measurements 6.

US1 US2 US4 B Process ID	**Peak Memory Usage (MB)**	**Lowest Memory Usage (MB)**	**Other process usage (MB)**
8223	7,1	3,5	287,8
8227	5,8	1,7	288,4
8231	6,6	1,6	286,9

8236	7,1	2,1	285
8240	7,1	3,4	285,6
8244	7,2	3,4	285,5
8247	5,8	2,1	285,8
8252	8,3	2,6	285,6
8257	5,8	2,1	286,5
8260	7	3,4	286

Table 41

Objective C memory measurements 7.

US 5 Process ID	**Peak Memory Usage (MB)**	**Lowest Memory Usage (MB)**	**Other process usage (MB)**
7726	7,6	3,5	284,9
7729	7,3	1,8	285,4
7734	7,6	3,5	285,2
7738	7,3	3,4	285,2
7743	7,9	2,3	285,1
7746	7,6	3,4	285,5
7750	8	1,8	285,6
7754	8,4	3,4	285,3
7758	7,3	2,1	285,6
7763	9,1	2,4	284,7

Table 42

Objective C memory measurements 8.

US7 Process ID	**Peak Memory Usage (MB)**	**Lowest Memory Usage (MB)**	**Other process usage (MB)**
7767	8,1	3,4	286,7

7770	8,2	3,5	285,4
7774	7,8	3,4	285,4
7778	7,9	3,4	287,2
7783	9,2	2	284,9
7786	7,7	3,4	286
7790	8,4	3,5	285,3
7794	8,3	3,5	285,6
7798	8,8	3,4	285,6
7802	8,9	3,5	285,5

Table 43

Objective C memory measurements statistics.

Statistic	**US1**	**US2**	**US3A**	**US3B**	**US4A**	**US4B**	**US5**	**US7**
Mean	3.48	5.75	8.64	6.66	9.97	6.78	7.81	8.33
Standard Error	0.013333	0.432884	0.17075	0.216128	0.059722	0.252894	0.181628	0.157797
Median	3.5	5.8	8.45	6.9	10	7.05	7.6	8.25
Mode	3.5	5.8	8.4	5.8	10	7.1	7.6	#N/A
Standard Deviation	0.042164	1.368901	0.539959	0.683455	0.188856	0.799722	0.57436	0.498999
Sample Variance	0.001778	1.873889	0.291556	0.467111	0.035667	0.639556	0.329889	0.249
Kurtosis	1.40625	0.30762	0.731908	-0.90881	4.619617	0.020667	1.87895	-0.81565
Skewness	-1.77878	-0.94373	0.715036	-0.09	-1.8112	0.23501	1.417603	0.508649
Range	0.1	4.2	1.9	2	0.7	2.5	1.8	1.5
Minimum	3.4	3.4	7.8	5.8	9.5	5.8	7.3	7.7
Maximum	3.5	7.6	9.7	7.8	10.2	8.3	9.1	9.2
Sum	34.8	57.5	86.4	66.6	99.7	67.8	78.1	83.3
Count	10	10	10	10	10	10	10	10
Confidence Level(95.0%)	0.030162	0.979253	0.386263	0.488914	0.1351	0.572087	0.410872	0.356962

Appendix IV: Objective C CPU Measurements

Table 44

Objective C CPU measurements 1.

US 1 Process ID	**Peak CPU Usage (%)**	**Lowest CPU Usage (%)**	**Other processes (%)**
7520	7	0	7
7523	66	0	37
7527	31	0	5
7533	21	0	4
7536	6	0	4
7540	66	0	7
7544	6	0	6
7547	66	0	5
7552	29	0	2
7555	6	0	5

Table 45

Objective C CPU measurements 2.

US1 US2 Process ID	**Peak CPU Usage (%)**	**Lowest CPU Usage (%)**	**Other processes (%)**
7563	43	0	9
7567	45	0	5
7571	47	0	4
7575	46	0	6
7579	50	0	10
7583	41	0	6
7586	61	0	5

7591	45	0	6
7595	43	0	7
7599	40	0	3

Table 46

Objective C CPU measurements 3.

US1 US2 US3 A Process ID	**Peak CPU Usage (%)**	**Lowest CPU Usage (%)**	**Other processes (%)**
7600	38	0	5
7605	47	0	4
7609	48	0	11
7613	50	0	5
7617	43	0	5
7624	49	0	6
7636	40	0	9
7641	37	0	4
7644	45	0	32
7648	45	0	5

Table 47

Objective C CPU measurements 4.

US1 US2 US3 B Process ID	**Peak CPU Usage (%)**	**Lowest CPU Usage (%)**	**Other processes (%)**
8177	64	0	5
8182	67	0	5
8185	54	0	7
8190	58	0	5
8194	61	0	10

8201	64	0	26
8206	69	0	6
8210	63	0	4
8214	58	0	6
8219	41	0	10

Table 48

Objective C CPU measurements 5.

US1 US2 US4 A Process ID	**Peak CPU Usage (%)**	**Lowest CPU Usage (%)**	**Other processes (%)**
7669	41	0	33
7678	50	0	7
7683	42	0	5
7687	44	0	7
7690	51	0	6
7696	52	0	21
7699	41	0	7
7704	42	0	7
7708	59	0	8
7712	47	0	2

Table 49

Objective C CPU measurements 6.

US1 US2 US4 B Process ID	**Peak CPU Usage (%)**	**Lowest CPU Usage (%)**	**Other processes (%)**
8223	64	0	9
8227	67	0	9
8231	65	0	30

8236	59	0	6
8240	57	0	4
8244	60	0	5
8247	47	0	4
8252	60	0	5
8257	52	0	6
8260	53	0	8

Table 50

Objective C CPU measurements 7.

US 5 Process ID	**Peak CPU Usage (%)**	**Lowest CPU Usage (%)**	**Other processes (%)**
7726	47	0	6
7729	46	0	6
7734	41	0	5
7738	41	0	4
7743	46	0	14
7746	47	0	6
7750	45	0	4
7754	44	0	4
7758	43	0	4
7763	46	0	5

Table 51

Objective C CPU measurements 8.

US7 Process ID	**Peak CPU Usage (%)**	**Lowest CPU Usage (%)**	**Other processes (%)**
7767	42	0	5

7770	45	0	4
7774	45	0	5
7778	44	0	5
7783	43	0	7
7786	43	0	11
7790	38	0	7
7794	39	0	9
7798	51	0	7
7802	42	0	6
7767	42	0	5

Table 52

Objective C CPU measurements statistics.

Statistic	**US1**	**US2**	**US3A**	**US3B**	**US4A**	**US4B**	**US5**	**US7**
Mean	30.4	46.1	44.2	59.9	46.9	58.4	44.6	43.2
Standard Error	8.317585	1.894143	1.451436	2.531798	1.9	1.989975	0.718022	1.133333
Median	25	45	45	62	45.5	59.5	45.5	43
Mode	66	43	45	64	41	60	46	42
Standard Deviation	26.30251	5.989806	4.589844	8.006248	6.008328	6.292853	2.270585	3.583915
Sample Variance	691.8222	35.87778	21.06667	64.1	36.1	39.6	5.155556	12.84444
Kurtosis	-1.51678	4.438447	-1.18622	3.038582	0.037567	-0.47212	-0.88634	1.970208
Skewness	0.596322	1.904056	-0.43643	-1.53988	0.853233	-0.44383	-0.73181	0.810281
Range	60	21	13	28	18	20	6	13
Minimum	6	40	37	41	41	47	41	38
Maximum	66	61	50	69	59	67	47	51
Sum	304	461	442	599	469	584	446	432
Count	10	10	10	10	10	10	10	10
Confidence Level(95.0%)	18.81568	4.284849	3.283377	5.727324	4.298099	4.501636	1.624279	2.563778