The University OF Nottingham

School OF Computer Science

Building The First Learn To Code Experience on Mobile

Oluwatosin Afolabi (ooa02u)
Supervisor: Dr. Julie Greensmith

Submitted May 2015, in partial fulfilment of the conditions of the award of the degree BSc. (Hons) Software Engineering.

I hereby declare that this dissertation is all my own work, except as indicated in the text:

Signature: Oluwatosin Afolabi

Date: 11th May 2015

Acknowledgements

I am grateful to my supervisor, Dr. Julie Greensmith, for her continuous support throughout the duration of this project. For being very understanding of my cirmstances and inspiring the idea that made this project feasible.

Abstract

This project was both a practical implementation of an iOS application that aims to teach the fundamentals of coding, and theoretical research into how best to teach people to code on a mobile platform. This report details the design and implementation of a minimum viable product (mvp) of the iOS app focus on some the difficult aspects such as designing and implementing the visual programming language. In the end, we concluded that the mvp was a great start and that it was effective way to teach people to code. Further work will include finishing the implementation of all requirements and further testing with users.

Words in text | 15,000+

Calculated with the TeXCount web service

http://app.uio.no/ifi/texcount/online.php

Contents

List of Figures

Chapter 1

Introduction

Figure 1.1: Project Logo

This section of the dissertation will give a general introduction to the project, Slang. We will start by analyzing the problem that Slang aims to solve, model what the solution will be, research the current alternatives, review the aims and finally explore my reasons as to why I decided to pursue Slang as my project.

In the next section, a literature review will focus on the technical details behind Slang as well as the research on how best to teach the fundamentals of code to others. This combined with the introduction chapter will form the foundation for the discussion of the functional and non functional requirements in the third chapter.

1.1 Problem Description

For the last few years there has been a growing hype about learning to code. This is not primarily about equipping the next generation to work as software engineers, it is about promoting computational thinking.[1] Cath Davidson said it best when she talks about the 4th R; "A student today needs a fourth R: Reading, riting, rithmetic and rithms, as in algorithms, or basic computational skills."[15]. This is because computational thinking gives people a new way to think about the world by teaching them how to tackle large problems by breaking them down into a sequence of smaller, more manageable problems. The applications of this approach stretch beyond writing software. Fields as diverse as mechanical engineering, fluid mechanics, physics, biology, archeology and music are applying the computational approach.[14]

With so many overwhelming benefits, why isn't everyone learning to code? This is because unfortunately, all of the options currently available to learn to code require a significant time investment at a period when the learner is unsure of the return on investment. For example, most resources teach specific languages at a very detailed level. Also, user feedback I have conducted has also shown that people usually feel like they are 'thrown in the deep end' with the available services and only those with a determined will to learn get past this initial stage. This becomes a huge stumbling block and a point of friction in the learn to code experience.

I believe that we need a step before users move on to the current platforms that exist. At this proposed earlier stage, a learner is eased into what code is about at a fundamental level while making it a fun, interactive and rewarding experience.

1.2 Proposed Solution

Following on from our problem description, The goals of our proposed solution can be summarized as follows.

1. Provide easier access to fundamental coding literacy (lower the barrier to entry)

2. Be the first step in solving the problem of a lack of skilled workers in the tech industry

In order to achieve the goals outlined above, it has been identified that the proposed solution should take the form of a mobile app due to their portability, availability, speed and ease of use.

In Chapter 2, we will study and analyze various learning app platforms such as Code Hour, Swifty and Duolingo as well as conduct research in how to teach computer programming. Knowledge gained from those activities will guide us when creating a list of requirements in Chapter 3.

[1] Computational thinking is how software engineers solve problems. It combines mathematics, logic and algorithms.

1.3 Aims and Objectives

There are two main aims of this project namely; Building a technically challenging and unique iOS application while the other is researching how best to teach the fundamentals of coding on a mobile interface with its limited screen real estate.

I am already familiar with iOS Development having built five previous applications. Thus with this project, I decided on specific areas of iOS Development that I wanted to improve on namely

- Learn Unit Testing with Quick and Nimble
- Learn UI Testing with FBSnapshotTestCase
- Learn about Continuous Integration
- Learn to use Autolayout rather than using manual frame calculation to layout views

From an educational side, Teaching code on a mobile phone is a very interesting problem to solve. At a high level scale, the areas we need look at in order to teach code and measure the teaching effectiveness include

- Creating a system that allows users to interact with code (in a mobile context)
- Creating a series of lessons that teach the fundamentals of code
- Creating quizzes and challenges that will allow the user to apply the coding knowledge they've gained
- Creating a process to measure the effectiveness of teaching code with Slang compared to the other existing solutions

1.4 Motivation

The ability to code is such an exciting and rewarding skill. Ever since Ive learnt to code its opened up many opportunities for me, its why I get to travel to New York once a week every month and Im still just a University student. Thus, I have become very passionate about more people also learning to code and reaping the wonderful benefits. I want others to discover how empowering and creative the ability to code is and how it can be leveraged to forge their own path in the world

I also believe that basic coding literacy can be the first stop in solving graduate unemployment, especially here in the UK. Many students are graduating with degrees for jobs that don't exist. Meanwhile in the tech industry, there is an exceeding demand for skilled workers. Services exist to help people learn code; but as was mentioned earlier, the barrier to entry is high. With Slang, we can lower the barrier entry, making learning to code as simple as downloading an app and going through it's bite sized lessons. This way, people eager to learn what code is about can take the first step and exponentially improve their career prospects.

1.5 Scope

While building Slang, the ultimate question is Is Slang an effective way to learn to program?. In order to ensure that we answer this question to a reasonable level of accuracy, we will limit the development of Slang to a Minimum Viable Product (MVP)[2]. The product will then be deployed to beta users who will provide valuable feedback.

The MVP of Slang will include lessons and quizzes that cover the fundamentals of programming. Beta users will then be asked to write short programs that will assess their understanding of the content taught so far.

1.6 Overview of Chapters

In chapter 2, we conduct a literature review of how best to teach the fundamentals of code and how best to evaluate if the concepts taught have been understood by the learners. After, we investigate why we believe the learn to code experience can be greatly improved by creating an application on a mobile platform. Lastly, we explore some of the existing learn to code experiences on mobile and summarize our findings of the important features that must be included when we try to build our own learn to code experience.

In chapter 3, we define the software requirements specification for the project. We look at the functional requirements (goals that the software must achieve), and non-functional requirements (constraints placed upon the system). The SRS provides an overview of the planned functionality that the project will have (upon submission to the AppStore) as well as acts as a framework on which we can develop test cases/criteria for the project.

In chapter 4, we focus on the design process. This involved drawing up multiple low fidelity prototypes for each of the views in the app. For each view, we analyzed the pros and cons of the different prototypes using a focus group and picked which we believed best fulfills our aims with the app.

In chapter 5, we discuss the implementation details. The idea was to go in enough detail that an intern would be able to understand the codebase and be able to maintain it going forward. In order to do so effectively, we covered aspects such as the iOS development cycle, the development tools, design patterns used, the software architecture and data structures created.

In chapter 6, we focus on the testing of the project. The testing is broken into two sections. First, we test components of the app using unit testing and then we test that the user experience of the app matches the software requirements set forth in chapter 3. Next, we perform an experiment in order to objectively answer the question, Is Slang an effective tool for learning to program. The experiment asks the user to go through the lessons and then provides a coding problem set so as to evaluate how much of the taught content was understood. Lastly, we present the results of the experiment and perform some analysis.

[2]In product development, A Minimum Viable Product is that version of a new product which allows a team to collect the maximum amount of validated learning about customers with the least effort.[28]

The final chapter, chapter 7, we conduct a final reflection on the project and outline further work that can be taken to improve it. This chapter outlines problems that were encountered, how these were overcome and a final personal reflection on the project.

Chapter 2

Background

The first half of this chapter will investigate the literature review behind teaching computer programming to novices. We will first define what we mean by computer programming; then review the research conducted that identifies some of the difficulties novices face when learning and see how we might lessen the impact to create a great learning experience for users. Next, we will go into more detail about the course details such as what will be taught, how, and with what language.

The second half of this chapter will review existing services on the iOS Appstore that aim to provide a learn to code experience on mobile and are designed for a mature audience. We will look at the features offered by several apps and analyze the trends we see between them. Next, we will discuss their benefits and limitations using user app reviews obtained through App Annie[1].

The literature review combined with a review of existing services will help us form a requirements specification (Chapter 3) that will guide the development of Slang.

[1] App Annie is the secret weapon for app store marketers, providing industry-leading app store analytics and market intelligence.

2.1 Literature Review

Teaching computer programming on mobile is a challenging problem; Due to the limited screen area, a mobile platform for learning to code requires the invention of a unique solution and user experience that is vastly different from those we may find in books, videos or web applications. Usually we look to existing solutions as a guide but before that; It is important that we research and answer the following questions: (1) What do we mean by the term learn to code? (2) What are the difficulties involved when novices learn to code? (3) What can we do to lessen some of the difficulties involved? (4) What approach will we take to teach coding? (5) What concepts will we teach? And, (6) What language will we use to teach these concepts with?

As we answer these questions, we will need to refer back to our projects aims in Section 1.2; which is to give the user a strong understanding of the fundamentals of code which will properly equip them to take their learning further by using other services.

2.1.1 What do we mean by the term learn to code?

At a high level, teaching computing to students takes many forms namely;

- Computing - Using a computer to achieve goals, like being able to type, using a spreadsheet program, editing video, etc.[21]
- Computer Programming - Understanding how to program a computer using one of the many programming languages in the world, either to solve math and science problems or to create interactive apps, games, and experiences[21]
- Computer Science - Gaining a deep understanding of the science and engineering of computers, both on the hardware side (electrical engineering) and the software (algorithms)[21]

If we refer to our aims we see that Computer Programming best describes our particular use case. Our ideal user knows absolutely nothing about programming but has gained some interest due to the hype about learning to code. The theoretical aspects of Computer Science will be good to know if the user decides to learn further but at the initial stage it is more important they are engaged with the practical aspects which show the value that can be gained by learning how to give commands to a computer.

In computer programming, there are four stages namely problem representation, design, coding and debugging. When we mean to teach computer programming we mean to give the user a thorough understanding of these four stages. Thus eventually when face with a problem, the learner will be able to understand the problem, design a solution, implement the solution and finally debug the solution if there are any errors in the program output.

2.1.2 What are the difficulties involved when beginners learn to code?

The learning curve for programming is rather steep because learning to program involves acquiring complex new knowledge, related strategies and learning how to apply the knowledge

gained to solve problems. To be more specific, Du Boulay describes five potential sources of difficulty that the typical novice may face namely: (1) general orientation , what programs are for and what can be done with them; (2) notional machine, a model of the computer as it relates to executing programs; (3) notation, the syntax and semantics of a particular programming language; (4) structures , creating a strategy to solve a problem; (5) pragmatics, various other important skills such as planning, developing, testing and debugging.[7]

> None of these issues are entirely separable from the others, and much of the 'shock' [...] of the first few encounters between the learner and the system are compounded by the students attempt to deal with all these different kinds of difficulty at once.[7]

Rogalski and Samurcay summarize the task of learning to code as follows:

> Acquiring and developing knowledge about programming is a highly complex process. It involves a variety of cognitive activities, and mental representations related to program design, program understanding, modifying, debugging (and documenting). Even at the level of computer literacy, it requires construction of conceptual knowledge, and the structuring of basic operations (such as for loops, conditional statements, etc.) into schemas and plans.[35]

2.1.3 What can we do to lessen some of the difficulties involved?

Students behavior and Approaches

Potential learners will have a diverse range of backgrounds and attitudes towards learning to code. Perkins[33] distinguishes between two main kinds of learners 'stoppers' and 'movers'. When confronted with a problem or a lack of clear direction, stoppers are likely to give up quickly; movers on the other hand will try to independently figure out their issue.

It is important that there is a process to identify the type of learner and provide teaching elements that complement their type. For example, in the case of stoppers, helpful errors and easy methods for them to gain clarification or help with concepts will be extremely valuable. For 'movers', they are likely to get through the course material quicker; as such there should be extra challenges and further material that they can take advantage of and advance their knowledge even further.

Deep Learning

The goal of teaching is learning[32]; to foster deep learning of principles and skills which creates independent and reflective students. Deep learning is the difference between being able to simply define what a for loop is compared to understanding how to use it appropriately in a program.

The methods for deep learning involve clearly stated course goals and objectives, stimulating the students interest and appropriate assessment/feedback. Linn and Dalbey propose a 'chain of cognitive accomplishments' that should arise from ideal computer programming instruction.[27]

In computer programming, appropriate assesment/feedback is achieved by using coding assignments. This is particularly effective as it encourages leanring by doing and the student gets to actively practise and develop their coding skills.

> The chain starts with the features of the language being taught. The second link is design skills and the procedural skills of planning, testing and reformatting code. The third link is problem-solving skills, knowledge and strategies (including the use of the procedural skills) abstracted from the specific language that can be applied to new languages and situations. This chain of accomplishments forms a good summary of what could be meant by deep learning in introductory programming.[27]

Programming Language Choice

The choice of programming language used to teach concepts is extremely crucial in an introductory programming course. Most beginners will not care about the theory behind computer science but rather the things they can get a computer to do. This is best done with as little overhead and boilerplate code as possible because the less code a beginner needs to write, the less errors and bugs they might encounter and the less likely they are to become frustrated in those crucial early days and give up on programming altogether.

2.1.4 What approach will we take to teach coding?

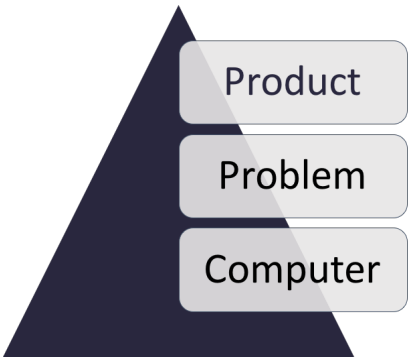


Figure 2.1: Programming Conceptions Hierarchy

When teaching computer programming there are three conceptions. Together they form a hierarchy based on logical inclusiveness. The three conceptions are computer programming as a computer-oriented activity, a problem-oriented activity or a product-oriented activity. Product orientation logically includes the problem orientation and both include the computer orientation.

The problem-oriented level is the ideal approach to teaching code in our case. If we refer to our aims, we understand that the mobile platform is limited and eventually any further learning will have to be done on a laptop or using a web application. Thus we need to develop a curriculum that is designed to teach the fundamental concepts of programming and how to apply it. This will allow the learner to break down a complex problem into simple components and construct

a program to solve it. Once understood, this skill is easily transferable to other programming languages and contexts.

The curriculum will be guided by three principles adapted from Code.org's K-12 Curriculum Philosophy and Goals.[11] They are as follows:

- Not emphasizing the memorization of factsrather, creating authentic opportunities for students to build knowledge and skills [11]
- Creating assessments that give learners an opportunity to demonstrate their understanding and soldering their understanding of concepts [11]
- Focusing on the concepts and transferable skills that will allow students to scale their knowledge and adapt to other programming paradigms and concepts [11]

2.1.5 What concepts will we teach?

The order of topics in programming courses has traditionally been bottom-up: start with the building blocks of programs such as variables and assignment then control, data structures and so on. This approach gives the students a good practical understanding of the fabric of programs.

In alignment with that idea; the list of topics that will be covered (in order) are as follows:

- Basic statement syntax
- Variables
- Math operators, assignment operators
- Logic and conditionals
- Loops
- Strings
- Functions
- Arrays
- Objects

This list was collated after observing the course structure of introductory programming classes offered by Khan Academy[1] as well as online courses from universities such as Harvard[20] and MIT[31]. With a strong understanding of these concepts, the learner will be able to solve simple computational problems such as printing strings, fizzbuzz and many others like those listed on the website, simple programming problems[2].

The nature of delivery of the the course content will be explored further in Section 2.2, where look at how existing mobile-focused learn to code solutions teach their users.

2.1.6 What language will we use to teach these concepts with?

In Section 2.1.3, we note that the choice of programming language used to teach concepts is extremely crucial. The prevalent standard amongst introductory courses used to be Java but has

[2]It is a collection of progressively more difficult programming exercises that are suitable for people who just started learning. http://adriann.github.io/programming_problems.html

now changed to Python. Python is superior to many other languages for teaching introductory programming because it allows the student to focus less on details such as types, compilers, and writing boilerplate code and more on the algorithms and data structures relevant to performing their intended tasks.[24]

In our case though, we will be choosing a different but similar language instead, Javascript. This is for two reasons namely; (1) Python requires indentation which isnt as easy on a mobile keyboard (2) Apples iOS doesnt have the python runtime included but it does have a Javascript Bridge[13] which allow you to run JavaScript code and observe its output.

Javascript will be a great choice but to achieve the least possible cognitive load for students at the beginning of the course, we could introduce a visual programming language (VPL)[3]. A VPL allows programming with visual expressions, spatial arrangements of text and graphic symbols[47]. Another benefit of using a VPL is it a more natural interaction (drag n drop) on mobile than typing text.

Figure 2.2: Scratch Language Preview

The most popular VPL is currently Scratch[26] and was designed with learning and education in mind. It allows the user to easily create games and provides a stepping stone to the more advanced world of computer programming[26]. There is no direct port of Scratch onto iOS but we can develop a simple implementation (adapted for our use case) to use in teaching the beginning parts of the course. More details about this will be discussed in the next chapter.

[3]A VPL is any programming language that lets users create programs by manipulating program elements graphically rather than textually.

2.2 Existing Solutions

In this section, we will begin by looking at Udacity[42] which offers a passive learning process. Next, we will move on to the Hour of Code App[10] and Swifty[40] which offer a more active learning process. Finally we will have a look at a very successful app called Duolingo[19], an app that teaches users several foreign languages. It will serve as a good reference app for how best to teach a skill through a mobile interface.

At the end of this section, we will then use the trends weve seen in the existing apps to create a list of required features that Slang should have.

2.2.1 Udacity

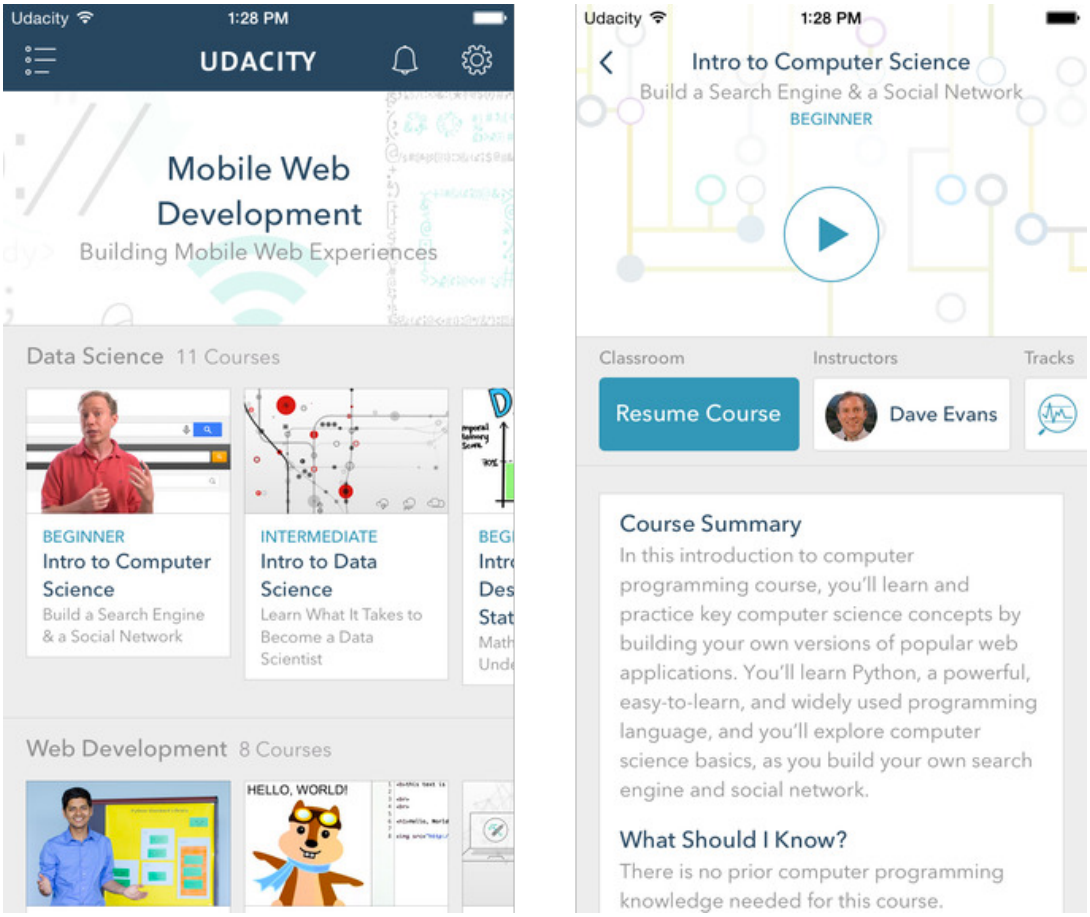


Figure 2.3: Udacity App Screenshots.

The Udacity app is a learning platform that aims to create a digital university for Computer Science students. It offers a wide range of programming courses from Intro to Computer Science and Intro to HTML and CSS to more advanced courses such as Artificial Intelligence for Robotics. Thus the app is not only aimed at beginners but intermediate coders alike.

The app flow is that a user picks a course which contains a mixture of lecture videos and interactive quizzes. It is however a passive learning environment as an app reviewer noted that 90% of the users time is spent watching the lecture videos.

The app benefits from the fact that the course are taught by industry experts. Other features include offline access to content, quizzes to reinforce concepts learned as well as progress indicators that motivate the user to keep learning.

2.2.2 Code Hour by Codecademy

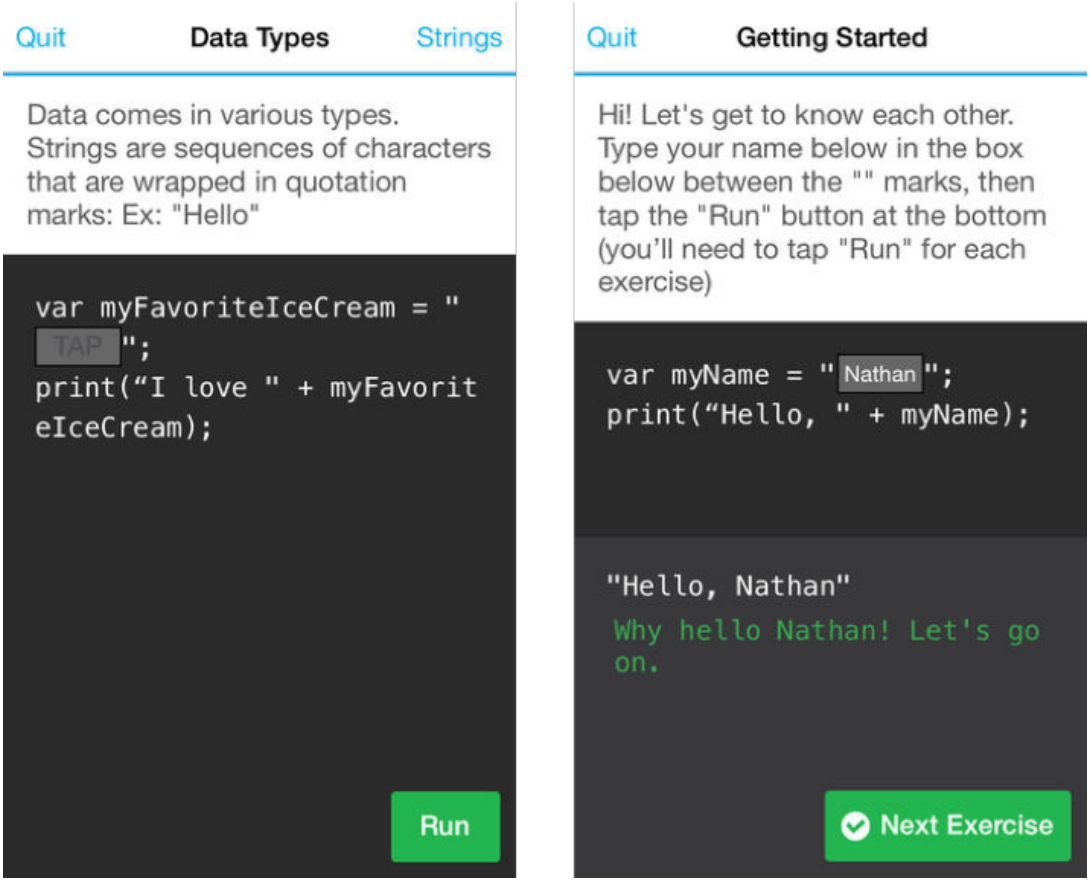


Figure 2.4: Code Hour App Screenshots.

Codecademy developed Code Hour for the Hour of Code in 2013. Its sole purpose is to introduce the user to the fundamentals of code. It offers a small number of lessons as it was designed to completed in an hour. Each lesson consists of a sentence or two of instruction, a snippet of code, and part of the code for the user to manipulate.

Code Hour uses an active learning approach by asking the user to perform an action before moving on the next lesson. If the user successfully completes the action, the user proceeds to the next lesson. If not, some helpful feedback is shown to clarify any problems. Also, a progress indicator is visible throughout the app so the user knows how far they are from

completion.

The combination of bite-sized lessons, friendly feedback and clear progress indicators gives users lots of positive feedback as they learnt the fundamentals of code. Looking over app reviews, there was a general consensus that the learning experience was excellent and most users wished that the app would be updated with more content.

The few features that could be said to be missing were the lack of quizzes, preventing the user from reviewing their knowledge. There is also no way for users to write custom programs or run any code limiting the users creativity.

2.2.3 Swifty

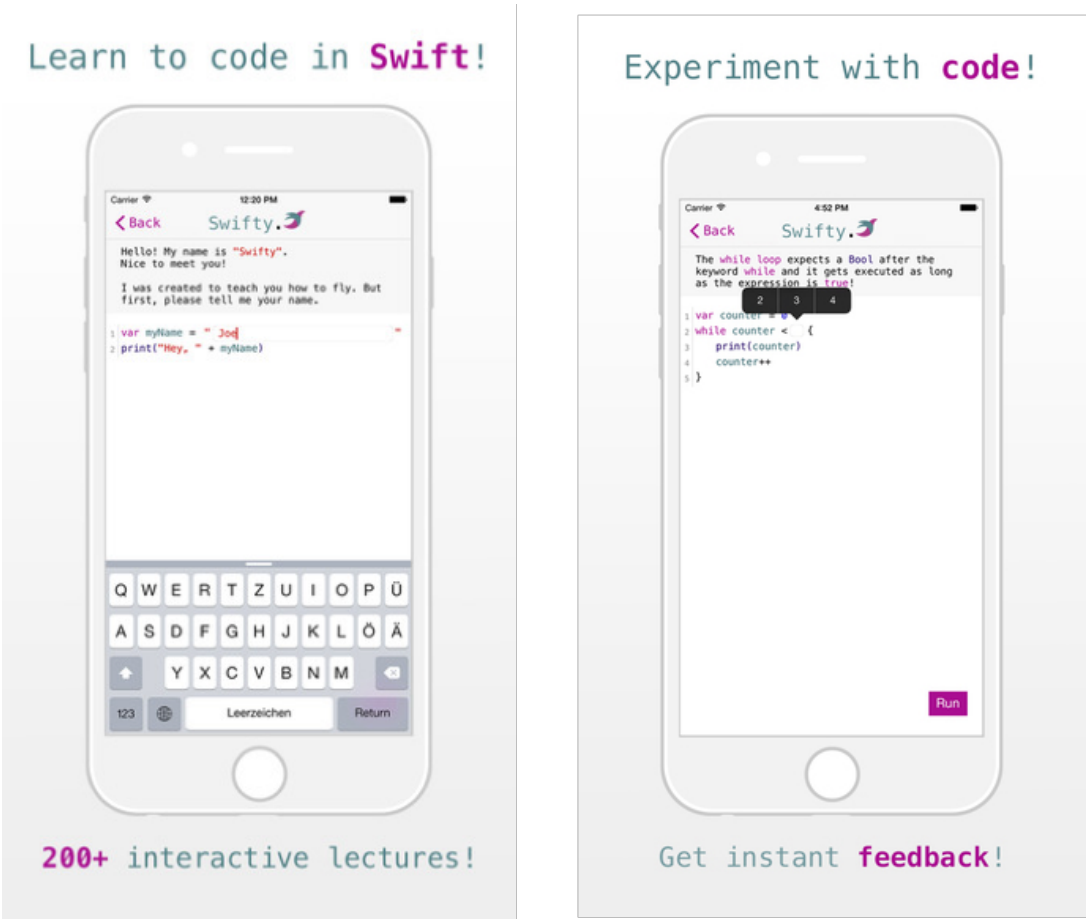


Figure 2.5: Swifty App Screenshots.

Swifty was released recently in March. Its aim is to gradually guide a user through the basics of Swift using an interactive set of tutorials. Each tutorial in Swifty starts with a one-to-three-sentence explanation of a new concept or an important aspect of a concept previously covered. Quizzes are added in-between tutorials. The content of the tutorials vary from basic concepts (variables, if and else, loops) to advanced topics like optionals, tuples and classes.

Similarly to Code Hour, the lessons are bite-sized, instructions are short and the UI is made deliberately simple to ensure that the users focus is on the lesson but unfortunately still fails to let the user actually create their own programs within the app which limits the users engagement with the app.

Swiftly received glowing reviews from users on the app store. Most thought it was a great platform but were concerned that the knowledge gained was basic but also wasnt accessible to people who werent already familiar with programming.

2.2.4 Duolingo

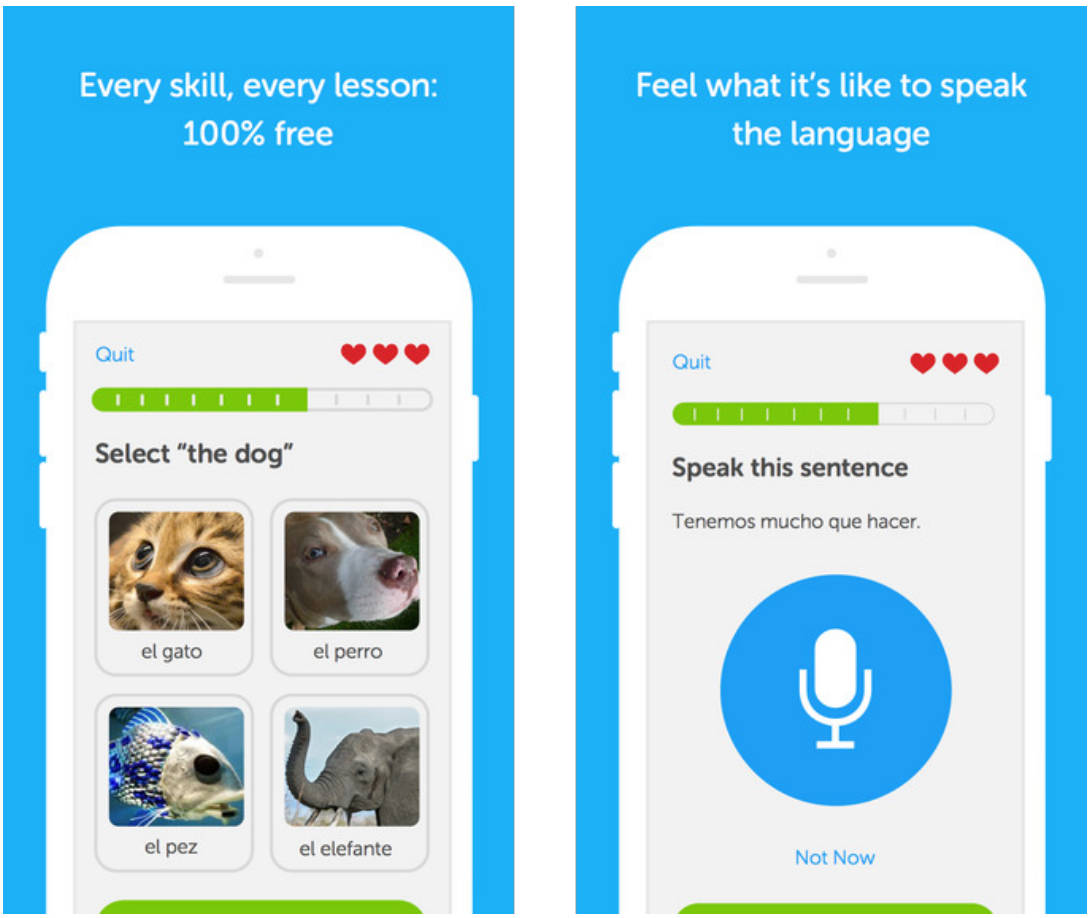


Figure 2.6: Duolingo App Screenshots.

Duolingo is different from the apps previously mentioned. It aims to teach foreign languages rather than code. So far, it has gained over 80 million users and was voted iPhone App of the Year, 2013 by Apple. Duolingo has done for foreign languages what I aim to do with Slang and learning to code. Thus, I felt it would be extremely beneficial to study what Duolingo does.

Similarly to the apps mentioned earlier, Duolingo offers a very simple and user friendly interface.

Lessons are bite-sized but where it differs from others is that the lessons are very dynamic and require a high level of user interaction. It uses a series of engaging mini-games that test your reading, writing, and speaking skills.

The bite-size lessons model has been proven very powerful by Duolingo. According to an independent study conducted by the City University of New York and the University of South Carolina, "an average of 34 hours of Duolingo are equivalent to a full university semester of language education."[43]

Another unique feature of Duolingo is its social integration. With Duolingo, users can challenge themselves to achieve new high scores or compete against friends to earn badges. Users can also easily track their progress and share their accomplishments to platforms such as Twitter and Facebook. This adds a social, competitive aspect to the app that drives engagement and organic growth for the app.

2.3 Criticism of Existing Solutions

The main issue that occurs with Swifty and Codecademy is the lack of any opportunity for the user to write and run their own programs. In Section 2.1.3, we note that the ability to write and run code is integral to the deep learning process; thus Swifty and Codecademy are failing to provide deep learning with their current solutions.

On the other hand, Duolingo constantly prompts the user to speak and write in the language being learnt. This is likely what has led to their huge success in teaching languages and as such I believe it is important that Slang provides a feature that allows users to write and run code.

2.4 Summary of Findings

By studying various learning app platforms such as Duolingo and using feedback gained from conversations with several beginners at hackathons as well as other interested candidates from CodeFirst Girl we discovered several key features that the app will need to have namely;

2.4.1 Short, Bite Sized Lesson Chunks (Duolingo Model)

The app should include short lessons so that users can learn a bit of code while waiting in the queue at Starbucks or on the tube on their way to work. It is important that these lessons require user interaction in order to ensure that the learner plays an active role in the lesson which will help reinforce concepts that are being learnt.

2.4.2 Fun, Interactive Programming Language

The app should include a fun, dynamic and natural way for the user to interact with code. In an earlier prototype, the user was allowed to write pure javascript into a textview. Several

discussions with beta users and my supervisor showed that this would be an ineffective way for the users to get to grips with code on a mobile interface.

A better choice would be to let the user write programs by dragging and dropping blocks of logic. Further details about this language will be explored in the design chapter.

2.4.3 Effective Emotion Design

In order for the app to achieve it's aims we need to keep the user engaged and coming back for more. We can effectively increase our user engagement rate by making good use of emotion design. Many successful apps like Facebook and Twitter use similar methods.

Emotion design involves appealing to the user's subconscious perception of the app and trying to turn the use of the app into a habit; albeit a healthy habit. We will delve further into what we mean by emotion design in a later chapter but essentially we can achieve a habit-like nature with subtle features, some of which are listed below;

- Puzzle Tasks and Challenges that provide the user with a sense of progress
- Humorous / Witty Push Notifications that will remind the user that they haven't used the app in some time
- Alert Views that provide positive reinforcement by congratulating the user after achieving certain tasks such as completing lessons or challenges.

2.4.4 Playground

Outside of lessons and challenges, the app will need to provide a playground to allow the user to create their own programs. This will be an opportunity for the user to practice some of the concepts learnt while also providing an avenue for them to become creative with their newly acquired knowledge.

2.4.5 Social Integration

The ability to share a users progress to platforms such as Facebook and Twitter is essential to driving the growth of the app. With more people sharing their progress and talking about the app, we can lower the barrier to learning code even more as it will seem like a more common interaction if the user sees their friends learning as well. Users can then also embark on this journey together and ask for help in concepts they havent quite understood yet.

With these key features implemented, the main aim is that once a user goes through the lessons provided in the app, they will have a strong understanding of the fundamentals of code and are properly equipped to take their learning further by using services such as Codecademy or Treehouse.

Chapter 3

Requirements Specfication

In this chapter, we will define the software requirements specification (SRS) for the project. The SRS consists of functional requirements (goals that the software must achieve), and non-functional requirements (constraints placed upon the system). The SRS provides an overview of the planned functionality that the project will have (upon submission to the AppStore) as well as acts as a framework on which we can develop test cases/criteria for the project.

We should note that the focus of this project is on evaluating if Slang can effectively teach people how to code. Thus only a subset of these requirements will be implemented also known as a Minimum Viable Product (MVP). This is done in order to validate whether or not the application can achieve it's aim. If it can, then the remainder of the requirements will be fulfilled.

3.1 Requirements Specification

3.1.1 Functional Requirements

1. On First Application (App) Start, The User should be presented with an Onboarding View
   - 1.1 The Onboarding View should display the app logo
   - 1.2 The Onboarding View should display copy that explains the app's purpose
   - 1.3 The Onboarding View should display a 'Login with Facebook' button.
   - 1.4 Upon a user logging in, the app should request permissions for the user's public profile, email address and friends.
   - 1.5 The Onboarding View should display a skip button to bypass logging in with Facebook.
2. On Further App Starts, The User should be presented with a Home View
   - 2.1 The Home View should display the app logo.
   - 2.2 The Home View should have a button that on tap presents the 'Playground View'
   - 2.3 The Home View should have a button that on tap presents the 'Profile View'
   - 2.4 The Home View should show a list of the lessons provided
     - 2.3.1 Each lesson should have a title and a subtitle.
     - 2.3.2 The subtitle text should give a summary of what will be learnt during the course of the lesson.
     - 2.3.3 If a lesson has opened previously, a progress bar should be shown to indicate how much progress has been made in the lesson.
     - 2.3.4 A completed ribbon should be shown if a lesson has been previously completed.
   - 2.5 After the user picks a lesson, the first page of the lesson should be displayed.
     - 2.4.1 Each page of a lesson should split into two.
       - a. The first part of a lesson page should contain instructions as to the action that the user should perform in order to progress. It should also provide any contextual information that will explain the purpose of what the user is to do.
       - b. The second part of a lesson page should contain the VPL interface which the user will use to perform the instructions mentioned in the first part
       - c. The second part of a lesson page may sometimes need to be pre-configured with data based on the instructions presented in the first part of a lesson page

2.4.2 The Lesson Page View should have a close button that dismisses the Lesson Page View and presents the Home View

2.4.3 The course content of the lessons should teach the fundamental concepts behind computer programming.

2.4.4 The course content should include

a. Basic statement syntax
b. Variables
c. Math operators
d. Assignment operators
e. Logic and conditionals
f. Loops
g. Strings
h. Functions
i. Arrays
j. Objects

2.4.5 Each page of a lesson should require some user interaction to occur before the user can proceed to the next page.

A. When the user is allowed to continue, a Next Button should be displayed.

B. When the user taps on the next button, the next lesson page view should be displayed.

2.4.6 There should be quizzes in-between pages of lessons that tests the user's memorization of concepts.

2.4.7 There should be a coding challenge at the end of each lesson that tests knowledge of concepts learned in the current lesson as well as those covered in previous lessons.

2.4.8 After a user completes a lesson, there should be a feature by which the user can share their progress on social media.

3. The app should include a Visual Programming Language (VPL)

3.1 A program using VPL should be created by dragging and dropping elements onto a board and placing them in order of execution

3.2 The VPL should have seven elements

3.2.1 A Blank Element with no arguments. Blank elements should be ignored when executing the code

3.2.2 A Variable Element that can be given a name and a value or statement. The name should be alphanumeric. The value can be either an integer, double, bool or string

3.2.3 A Repeat Element that can be given a count value. The count value must be an integer.

3.2.4 An If Element that can be given two statements and one mathematical comparison operator. The If Element can either evaluate to true or false

3.2.5 An Else Element with no arguments

3.2.6 An End Element with no arguments

3.2.7 A Print Element that be given one statement

3.3 Any elements between an If element and an End element should be only executed if the If Element evaluates to true.

3.4 Any elements between a Repeat element and an End element should be executed as many times as the count argument of the Repeat element states

3.5 There should be an execute button that on tap executes the code. The results should be displayed in a console view that pops up.

3.6 The console view should be dismissed by tapping outside the console view.

3.7 The VPL should provide feedback for common error scenarios

3.7.1 An error message should be displayed if any elements to be executed have missing arguments.

3.7.2 An error message should be displayed if any elements to be executed have arguments of the wrong type

3.7.3 An error message should be displayed if the IF or Repeat Element are not ended with the End Element

3.8 The error feedback messages should be displayed in the console.

3.9 This VPL should be used to teach the lessons as well as for the 'Playground View'

3.10 Upon completion of a lesson page or a challenge, accompanying text should be displayed in the console that congratulates as well as educates the user on the task they have just accomplished.

4. In the Playground View the user should be able to write and run arbitrary programs and view the results in the console

4.1 The Playground View should have an info button that on tap presents an overlay with information on the function of each of the VPL elements

4.2 The Playground View should have a close button that dismisses the Playground View and presents the Home View

4.3 The Playground View should contain the VPL Interface

4.4 The Playground View should have a button that on tap presents the 'Challenges View'

4.4.1 In the Challenges View, there should be a list of easy and intermediate level of coding challenges.

4.4.2 Each challenge item in the list should have a title, description and level of difficulty.

4.4.3 Upon picking a challenge, the user should be presented with the Playground View, with the challenge instructions shown at the top of the view.

5. The App should include a Profile View

   5.1 The Profile View should contain a profile picture obtained from Facebook as well as their name.

   5.2 The Profile View should show the number of lessons and challenges completed.

   5.3 The Profile View should have a share button, that on tap, will let the user share their profile and progress on a social network of their choosing.

   5.4 If the user has not logged in yet, a Login Overlay should be shown in the Profile View.

   5.5 The Login Overlay should include the Login with Facebook button as well as copy describing the benefits gained from logging in.

6. The Lesson List, Lesson Content (individual lesson pages) and the Challenge List should be stored in a JSON file.

   6.1 The JSON file should be stored locally on the device

   6.2 The Lesson Picker View, Lesson Page View and Challenge List View should be populated using the JSON

   6.3 When the app is opened, it should perform a network request to check if the JSON file has been updated on the server. If so, it should fetch the new JSON file and overwrite the existing JSON file.

7. If the user hasn't opened the app in a week, a push notification should sent to remind the user to continue learning with the app.

   7.1 If the user still hasn't opened the app, a new push notification should be sent two weeks after. If they still haven't, another notification should be sent four weeks after. After that if the app still hasn't been opened, no more notifications will be sent.

   7.2 Push Notifications should be sent using the Parse service

3.1.2 Non-Functional Requirements

Accessibility / Platform Choice

It is very important that the application be widely available in order to ensure that as many as people that want to learn coding are able to do so easily. Unfortunately cross platform technologies for building native apps offer a poor user experience[1] thus a single mobile platform has to be chosen. The Android ecosystem is much larger the iOS but there is a much greater variation in device capabilities which will require extra development time. As such, the app

[1]Due to their reliance on web views which have limited API access and performance issues

should be developed for iOS, should support the iPhone 5, 6 and 6 Plus Screen Sizes and target devices on iOS 8.0 and above.[2]

Usability

Learning to code requires a certain level of mental effort from the user. As such the user interface and user experience of the app should have as minimal cognitive overhead[3] as possible. The users focus should be on learning how to code, not learning how to navigate the app. After the design stage of the application, the usability of the app should be evaluated using the '10 Usablitiy Heuristics' by Jakob Nielsen[30].

Security

Due to the limited user data that is stored, security is not a major concern in this application. Users have a choice to sign up or not. For those that do, they should do so using the Facebook Login SDK[4] and the session tokens should be stored on Parse[5]. The use of these third party systems (benefitting from enterprise grade security) will ensure that our users can sign in with confidence and trust that their details are safe.

Another concern is that of malware. Malicious users should not allowed to install malware and distribute copies of the app. Fortunately, this is not possible on iOS as Apple code signs[6] every app distributed on the AppStore.

The last major concern is that of API keys. Any API keys stored in the application should be obfuscated in order to prevent malicious users from easily discovering it and hacking the applications systems.

Software Quality

The software should be as robust and as bug free as possible. This is especially important with iOS apps as any updates to the app have to be reviewed by Apple before being released; a process which takes about two weeks. For any bugs that do show up, a support email should be created so that users can report them.

Performance

As the application is being developed on a mobile platform, we have to take into account the constraints on mobile device with regards to battery life and cpu speed. The application should

[2] iOS 8 adoption rate is at 85%[29]

[3] How many logical connections or jumps your brain has to make in order to understand or contextualize the thing youre looking at.

[4] https://developers.facebook.com/docs/facebook-login/ios/v2.3

[5] Powerful cloud database - https://www.parse.com/

[6] Code Signing is a means of identifying where a particular piece of software originated

be built with these concerns in mind, ensuring that all resources are used efficiently and that long running tasks are processed on background threads

Chapter 4

Design

In this chapter, we start by describing in detail, the design process that was used to create prototypes of the application. We will then give a brief overview of the tools used in the design process. After that, we begin to discuss the prototypes developed. Before and after figures of the prototypes are included to illustrate how the prototypes have evolved using the valuable feedback provided by the focus group. Finally, we discuss the high level design of the software system that will be implemented.

4.1 Design Process

Learning to code on a mobile platform is a challenging problem to solve as I've said many time before. Unfortunately, I'm already proficient at coding and as such I am not part of the target user group. This means no matter how hard I may try, I could never be confident that my prototypes are a suitable solution for my target users without extensive usability testing.

To solve this problem, I decided to use a design process that placed extra emphasis on the prototyping phase, initiating it as early as possible. Early integration of prototyping is said to work well especially on projects with complex technicalities like ours.[8]

4.1.1 Rapid Prototyping

The design process I used for this project is called Rapid Prototyping. It involves multiple iterations of a three-step process; Review, Prototype & Refine.[9] An overview of each step is given below:

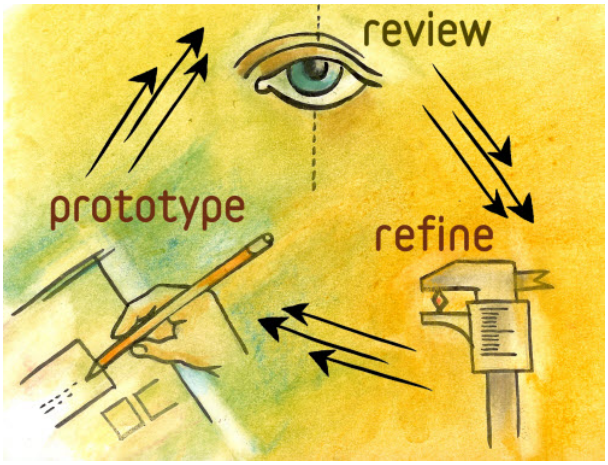


Figure 4.1: Rapid Prototyping Process[9]

- **Prototype**
  Convert the requirements of the solution into mock-ups, factoring in user experience standards and best practices.[9]
- **Review**
  Share the prototype with users and evaluate whether it meets their needs and expectations.[9]
- **Refine**
  Based on feedback, identify areas that need to be refined or further defined and clarified.[9]

Rapid prototyping lets one experiment with multiple approaches and ideas quickly. It also facilitates discussion through visuals instead of words, ensuring that all stakeholders share a common understanding of what's being developed. This also guarantees that no requirements are missed, leading to a better design faster.

What needs to be prototyped?

I choose to prototype the views in the app (as set forth by the requirements in Section 3.1) as well as the Visual Programming Language. The views that were prototyped are listed below:

- Lesson Picker View (Home View)
- Lesson Page View
- Lesson Complete View
- Profile View
- Challenge View
- Playground View
- Quiz View

For each of these views, multiple low fidelity prototypes were developed using Balsamiq Mockups. Details of the tools used in the design process are explored later on in this chapter.

4.1.2 Review Method

The second step in the 3-Step Rapid Prototyping process is Review. There are two possible ways one can conduct this review; focus groups or usability testing.

- **Focus Groups**
  Focus groups gather approximately 6-8 representatives of the target market together with a moderator and have them discuss their feelings, ideas and opinions on topics. They attempt to gather many people's thoughts and attitudes on ideas and/or designs.[18]
- **Usability Testing**
  Usability testing involves using a 1-on-1 (1 person and 1 facilitator) interaction with a system or website. The facilitator runs through key tasks with the user and analyses how well they perform these tasks and how they find the whole experience. It focuses on the interaction between people and a website/system.[18]

Usability testing is usually the better choice as you can get more detailed information from each user but I decided to go with a focus group as in my design process, I decided to create multiple prototypes of each screen. A focus group evaluating each view and the merits of the different prototypes for that view would be a better fit than usability testing which involves testing one complete system.

4.1.3 Review Process Using Focus Groups

In partnership with Code First Girls, we reached out to find 10 people that fit the criteria for our target user base. The criteria for selection was as follows:

- Must have an interest in learning to code.
- Must have no previous coding knowledge.
- Their primary phone must be an iOS device.[1]

[1]This is so that participants will already be familiar with iOS design patterns

At the start of the focus group, the participants were made aware of application's aim as well as its requirements. During the focus group, each person was given a booklet filled with the different prototypes of each view. We proceeded through the process by focusing on one view at a time. Each view was allocated 15 minutes. The first 10 minutes were used to engage in a general discussion about the different prototypes. The other 5 minutes were used to let the each person in the focus group pick which they felt was the best prototype and note down their final comments.

After looking through each view, we compiled the most popular prototype for each view and combined that into a system. The focus group then went through the system and discussed any new thoughts or ideas that came up.

A list of design choices for each view was also compiled so that discussion could be guided on the right path; thus ensuring that the feedback would be relevant.

4.2 Tools

Apart from a pen and paper, there are several applications that can be used in order to prototype an app; two of which I used in my design process. The first is Balsamiq[5] which I used to create low fidelity prototypes and quickly experiment with different layouts. Once I decided on a single prototype, I used Sketch[37] to create a high fidelity prototype of the concept and used it's accompanying app, Sketch Mirror, to test out the prototype on a physical device.

4.2.1 Balsamiq Mockups

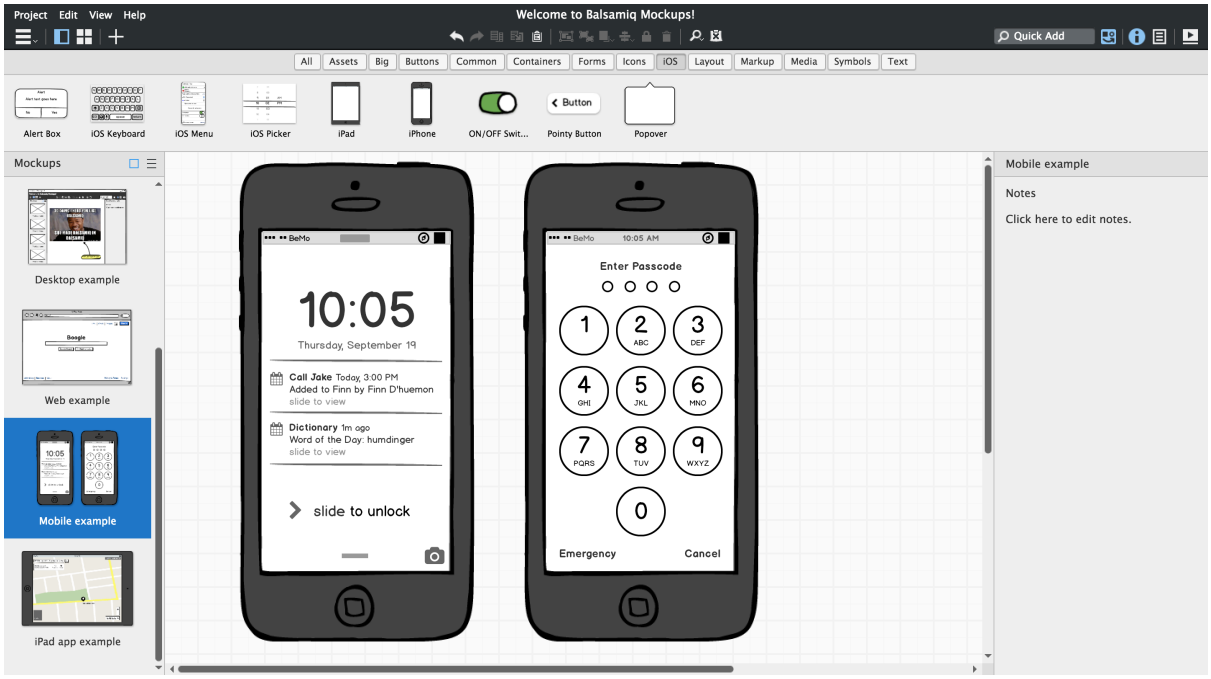


Figure 4.2: Balsamiq App Preview.

Balsamiq Mockups is a small graphical tool to sketch out user interfaces, for web and mobile applications. I choose to use this instead of the many options available due to it's focus on the ideation phase of design. Balsamiq was intentionally design to be be simple and limited in it's features. This ensures that I stay focused on the structure of the design rather than colors and icons.

4.2.2 Sketch

Sketch is an application specifically made for designing interfaces. It comes packaged with iOS design templates as well as iOS icon templates. This makes it fast and easy to get started with designing iOS applications. Another great feature of Sketch is that any interfaces or icons designed are vectors. This is extremely important when exporting assets for the different screen

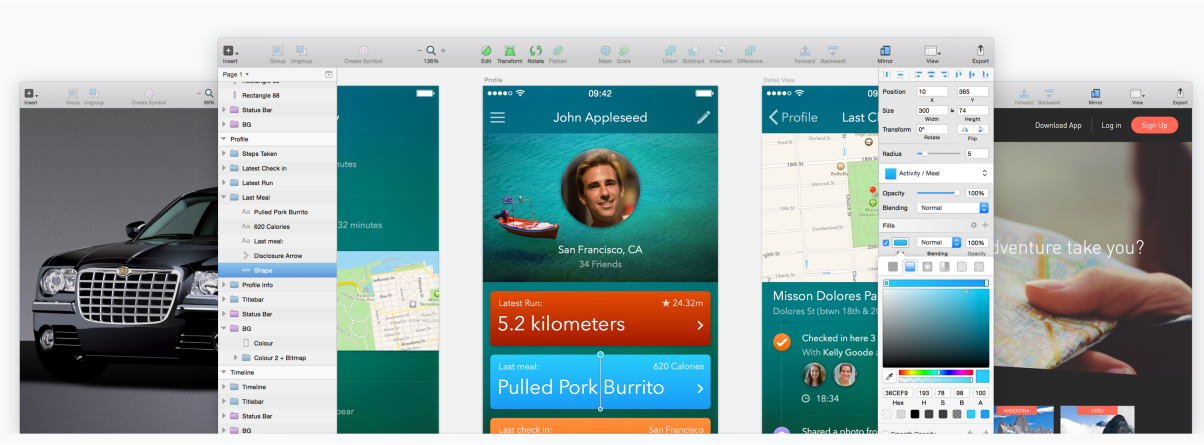


Figure 4.3: Sketch App Preview.

sizes and resolutions of iOS devices. These features made it easy to design high fidelity versions of the prototypes I had developed in Balsamiq.[38]

Sketch Mirror

Another amazing feature of Sketch that deserves mention is Sketch Mirror[38]. It shows a live preview of your design on the iOS device your pair the application with. This makes it easy to test designs on an actual device and make changes which you see instantly.

Sketch vs. Photoshop

Photoshop[2] is the current standard for designing interfaces but recently a lot of designers have transitioned over to Sketch. The main reason for this is due to the fact that Photoshop has always been a photo editor first and design tool second. This has led to Photoshop's interface being rather clunky and a high barrier to learning Photoshop. With Sketch, it's solely focused on interface design resulting in a simpler interface and quicker design workflow.[39]

4.3 Education Framework

For the design of lessons, we decided to base them on Experiential learning. Experiential learning is defined as the process of learning through experience.

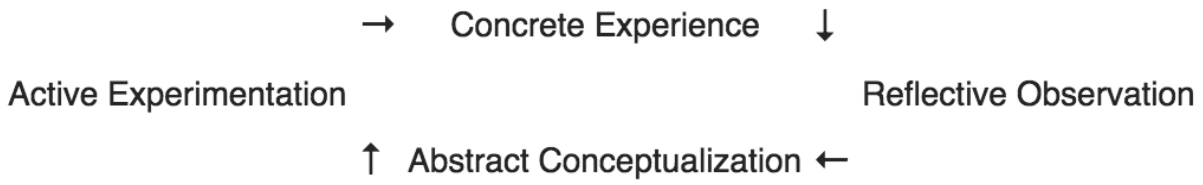


Figure 4.4: David Kolbs Experiential Learning Model (ELM)[45]

We will be using David Kolbs Experiential Learning Model which is illustrated above. Kolb states that in order to gain genuine knowledge from an experience, the learner must have four abilities:[45]

- The learner must be willing to be actively involved in the experience[45]
- The learner must be able to reflect on the experience[45]
- The learner must possess and use analytical skills to conceptualize the experience[45]
- The learner must possess decision making and problem solving skills in order to use the new ideas gained from the experience[45]

Using the Kolb Model, a lesson was designed as follows:

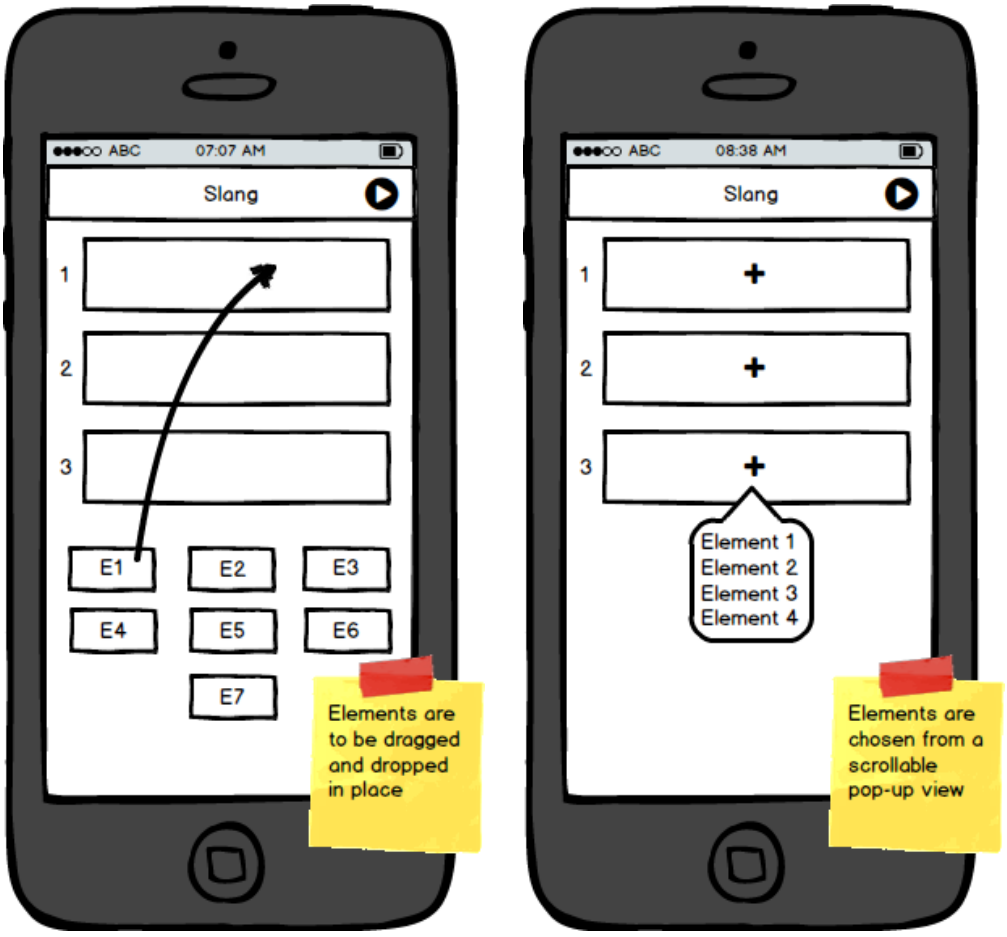
- **Abstract Conceptualization** - A short piece of information is displayed at the top of the lesson page. The information will also contain some action that the user has to perfrom.
- **Active Experimentation** - The user experiments and figures out how to perform the action required and clicks the execute button.
- **Concrete Experience & Reflective Observation** - The user can then see the result of their action ad assess if it is what they expected. If it isn't there will be helpful information at the bottom stating where they might have gone wrong. If their action is correct then, a message will shown provide supporting context and congratulate them on getting the task right (positive reinforcement)

Experiential activities are among the most powerful teaching and learning tools available.[45] From our implementation of the model that we have just described, we validate the need for a visual programming language so that our users can able to experiment and have a more active role in their learning. Also, the extra playground mode listed in the requirements will serve as another way for our users to continue using this model to learn without any guidance and at their own pace.

4.4 Visual Programming Language Design

Requirement 3 in the Functional Requirements section states that the app should have a visual programming language. This requirement is informed by the Literature Review (Section 2.1.3) where we note that deep learning is one of the possible ways we can lessen some of the difficulties involved with learning to code. The idea is that code education can be very effective if students are provided the opportunity to write code, execute it and view the results.

4.4.1 Language Interface Design



Prototype A Prototype B

Figure 4.5: Language Interface Prototypes

Shown above are the two possible prototypes for the interface of the visual programming language. The main difference between the two prototypes is that in Prototype A, the elements are always visible while in Prototype B, they are not.

Prototype B's hidden elements resulted in a cleaner interface than Prototype A but the user feedback panel all agreed that although cleaner, discovery of the different elements was limited and slow. They attributed that fact to the two taps[2] required to select an element. While in Prototype A, they were more confident about the choices they had as all of them are presented at once. They also noted that the drag n' drop mechanic was a lot more fun and allowed them to easily experiment with several elements much faster than would have been possible in Prototype B.

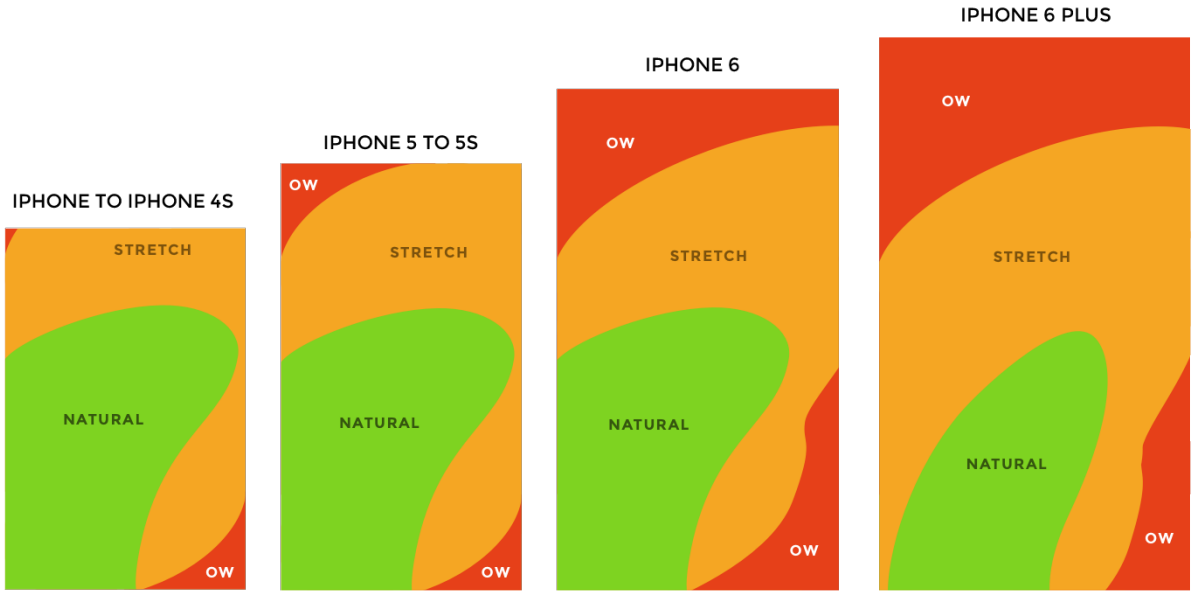


Figure 4.6: Thumb Zone Heat Map[25]

Another concern was the reachability of the execute button. On the iPhone 6 and iPhone 6 Plus, the execute button is well into the red 'ow' area meaning that it will much harder for a person's thumb to reach. This is especially an issue in this case due to the frequency at which we expect that a user will tap the execute button.

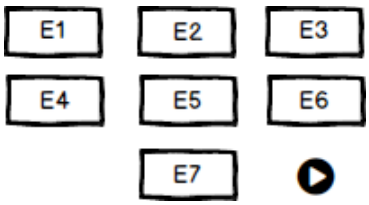


Figure 4.7: Execute Button

Using that feedback, an alternate solution was developed. The execute button was moved from the navigation bar and placed beside the elements(as can seen on the left). This also had the added benefit that similar functions were being grouped together. This is essential when creating an intuitive user interface. With the following changes made to prototype A, members of the focus group were happy with the final prototype and felt that it was a very simple, intuitive interface.

[2]First tap to bring up the pop-up, second tap to to pick an element

4.4.2 Language Element Design

Figure 4.8: Initial Language Elements Prototype

Shown above is the initial prototype for the language elements. I wanted a very clear interface so that it would easy to teach users how to manipulate the elements. At the top left of every element is its title, in the center, its argument, if any. If an element contains more than one argument subsequent arguments will be separated by a colon. While an argument is empty, its name is used as the placeholder text. Looking closer at the arguments, we see that they are given a circular border. This to indicate to the user that the arguments are tappable. Tapping on an argument is the method by which a user can edit it.

When this prototype was shown to the focus group, they responded by raising some concerns. Their feedback was as follows:

1. The border around the arguments was found to be distracting. The focus group viewed the code for a small program written using the elements. The repeated border was very heavy visually and they all agreed that they did not need it in order to be reminded that they could edit the argument. Once was more than enough.
2. Once an argument was filled in, the placeholder text disappears. The focus group found it hard to remember what the arguments meant once they were filled in. They requested that some more context be provided. For example, putting the name of the argument below the edited argument when filled in.
3. In the spirit of the previous point, the focus group wanted a mechanism by which they could quickly view information about the use of a particular element.
4. The biggest issue the focus group encountered was the lack of direction on how to delete an element from the program. A blank element does exist but the focus group didn't believe that that solution was very intuitive. They felt an actual delete action would be best choice.

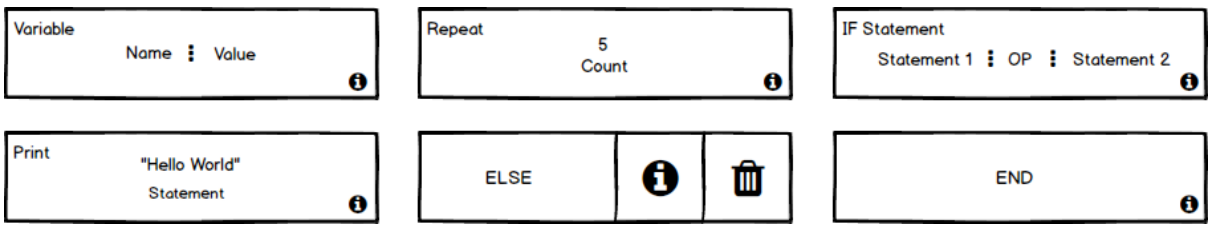


Figure 4.9: Final Language Elements Prototype

With the above concerns in mind, I reviewed the prototype and shown in Figure 3.8 is the final prototype. The first striking difference is that the borders have been removed. In the Repeat

and Print elements, we see that once an argument is filled, the arguments name is displayed below. In order to provide quick access to information about an element, an info button is placed at the bottom right corner. The info button may look small but the actual tap target would be made bigger during implementation. This was done to ensure that the element remains clutter free and the focus remains on the arguments.

In order to solve the element deletion problem, I considered adding the delete button on the element, much like the info button but realized that user's might accidentally tap the button which is a concern because of the delete's button destructive effect. An alternative solution was sought and I decided to use an established iOS design pattern of swipe-able cells[3]. When the user swipes on a cell, two buttons are shown. One for info, the other for deletion. This ensured a clutter free design while ensuring that extra actions are only a swipe away. This can seen on the Else element in Figure 3.8. An extra detail that I will mention is that on swipe of the element, the original info button on the face of the element is faded out. This is again consistent with keeping the element clutter free and avoiding repetition of actions[4].

[3]Apple uses this design pattern in the default mail application.
[4]Having two actions that perform the same effect.

4.4.3 Language Interface + Element Design

Figure 4.10: Language Interface + Elements

The culmination of the work done in Section 3.4.1 and 3.4.2 can be seen in Figure 3.9. The result is a simple, intuitive interface that feels right at home on mobile due to the innovative drag n' drop function by which users can create programs. The drag n' drop pattern is extended further for another important interaction, rearranging elements. When a user long presses on any placed element, they can begin to rearrange the order of elements.

Another innovative interaction can be seen on the second prototype in Figure 3.9. When the initial language interface + elements prototype was shown to the focus group, they raised a concern with the inability to add new elements while the keyboard was on screen. The solution at the time was to tap outside the keyboard area to dismiss it and then proceed to drag a new element. 'Not a simple nor an intuitive process especially with the delay of the keyboard's appearance and dismissal' was one of the quotes from the focus group. After reviewing their feedback, I came up with a solution of having a vertical list of the elements above the keyboard. With this solution, users could scroll the list, drag an element into order and proceed to edit the elements arguments.

4.5 App User Interface Design

4.5.1 Home View / Lesson Picker Design

Figure 4.11: Lesson Picker Prototype

Figure 3.10 shows the final prototype for the Lesson Picker View. At the top, we have a navigation bar with the title of the app and two icons. The first icon leads to the profile view and the other leads to the playground view. Next, we have the list of lessons. Each lesson has a banner, icon, title and a lesson summary. If a lesson has been completed previously, a small checkmark is shown on the lesson banner.

4.5.2 Lesson Page Design

(a) Lesson Page (b) Lesson End

Figure 4.12: Lesson Interface Prototypes

Figure 3.11a shows the final prototype for the Lesson Page View. To save on space, the navigation bar was removed and used to display instructions for the lesson page. The rest of the interface consists of the VPL interface discussed in Section 3.4 with a minor difference; the look and position of execute button is changed. It now appears at the bottom right of the view. While the lesson is in progress, the execute button will display the title 'Run'. Once the user successfully completes the lesson, the title of the button will be changed to 'Next Exercise' and used to proceed to the next lesson page.

Lesson End Prototype

Figure 3.11b shows the final prototype for the Lesson End View. The purpose of this view is to congratulate the user on finishing a lesson and spurring them on to attempt the next lesson.

The red icon is the badge that will be collected and shown in the user's profile. The view also contains two buttons, the first is to allow the user to share their progress on social media. The second is to allow the user to move on to the next lesson. One last icon is place at the top left of the view. It allows the user to return to the lesson picker view. It is deliberately placed there to discourage it's use as we want the user to continue on to the next lesson.

The Playground View and Quiz View are very similar to the lesson page view.

4.5.3 Profile Design

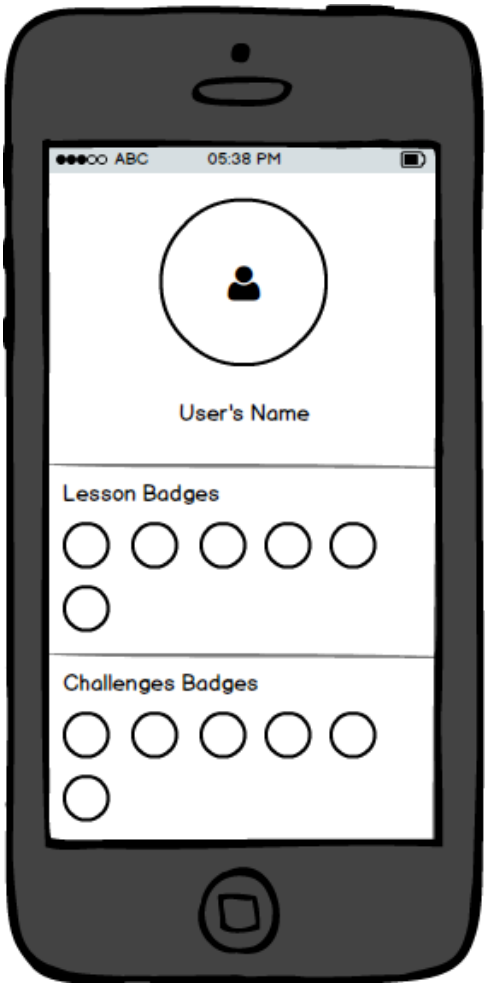


Figure 4.13: Profile View Prototype

Figure 3.12 shows the final prototype for the Profile View. The interface includes the user's picture and name. Below that, we have an icon list of the lesson and challenge badges earned. The addition of badges gamifies the application and will hopefully encourage the user's to collect more of these badges by completing lessons and challenges.

4.5.4 Challenge List Design

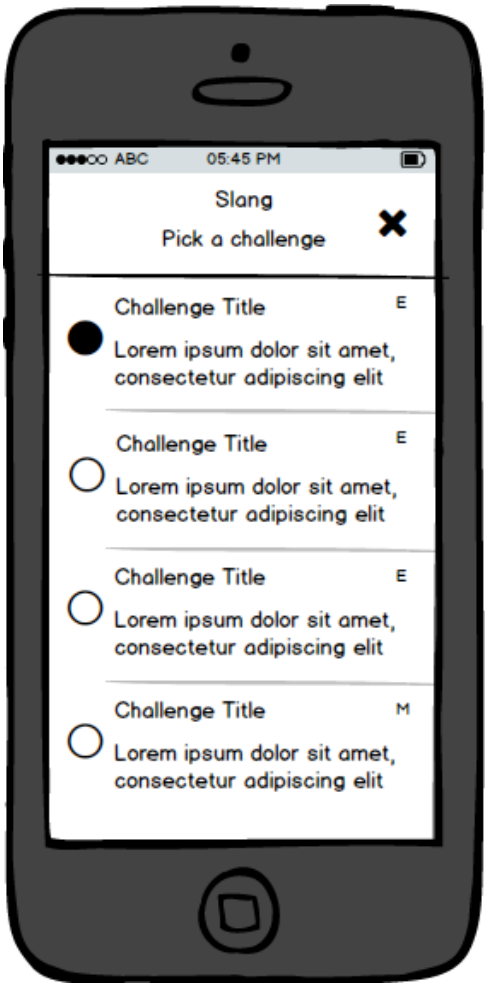


Figure 4.14: Challenge List Prototype

Figure 3.13 shows the final prototype for the Challenge List View. Majority of the interface contains a list of challenges. Each challenge item has a title, description and level of difficulty (E for Easy, M for Medium). In addition to that, there is a circular icon on the left which acts as a progress indicator; if a challenge is completed then circle is filled.

4.6 System Design

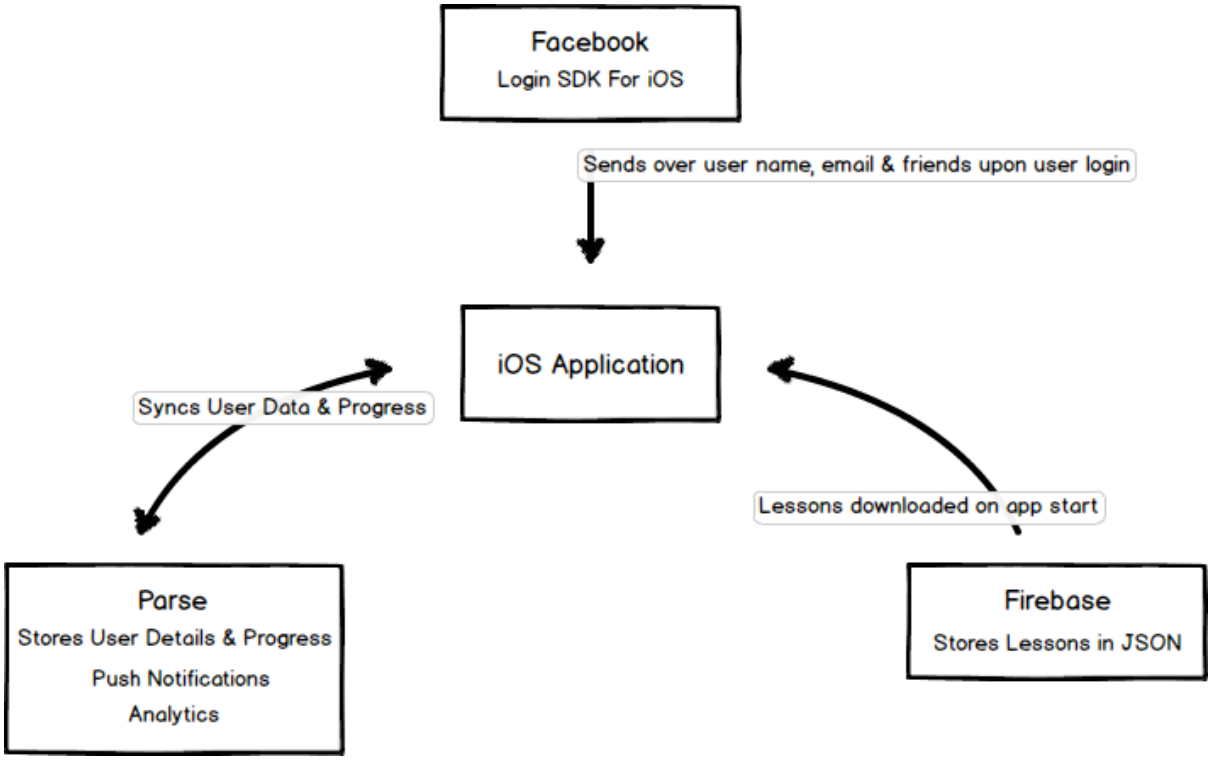


Figure 4.15: System Design

Figure 4.14 shows a high level overview of the system design for the application. As you can see, we require three third party services to provide the required functionality. Facebook's Login SDK v2.3 for iOS will provide the login mechanism for the application. When the user logs in, Facebook will return an object with the user's name, email and list of friends. Facebook will also provide an Access Token which prevents the need for the user to login again.

For maximum security, we will not store this information on the device but rather on a database provided by Parse. Parse is a cloud app platform that enables the addition of a scalable and powerful backend to any app. Parse helps developers avoid having to build a back end solution for their application. It also has other added benefits such as tight integration with the Facebook Login SDK, ability to send targeted push notifications and support for app analytics.

The last component in this trifecta is Firebase. Firebase is another Backend as a Service (BaaS) like parse but with a much sharper focus on databases. With Firebase, we can easily host the JSON of all the lessons (including their pages) and fetch them once the app is opened(a local cache will be stored in the event that there is no internet connection). This system is critical as it enables us to add more lessons to the application without requiring users to update the application.

The remainder of the required functionality is handled with the iOS application and we discuss this in the next chapter.

Chapter 5

Implementation

In this chapter, we begin with an overview of iOS development and the design patterns that we will be using to develop the app, the most important of them being Model-View-Controller (MVC). Next we will discuss the methodology used to develop the app. After that, we will outline the tools and technologies that were used during the implementation of the software.

The next sections in this chapter will discuss in detail, specific aspects of the implementation that were interesting or challenging to accomplish. We will then end discussing any general difficulties that were discovered while building the application.

5.1 MVP Requirements

As this projects main aim was to validate is Slang was an effective way to teach coding, we decided to build only a subset of the requirements. What will be included in the MVP is the lesson picker view, the visual programming language, introductory lessons (with no quizzes or challenges) and the playground mode.

5.2 Building Native iOS Applications

Building on the iOS platform is a great experience due to the extensive resources and frameworks that have been developed by Apple. However it does have a high barrier to entry for two reasons. Firstly, to develop iOS apps, one needs a Mac computer and Xcode[1]. The second reason is that iOS development has a steep learning curve as one must learn either Objective-C or Swift; programming languages that are hardly found outside of iOS app development.

5.2.1 Objective-C vs. Swift

Objective-C has long been the de-facto standard for developing iOS applications. It is a very popular language which is constantly being improved by Apple but the technical debt due to it's age (32 years) has limited its adoption of some of the useful features that are present in more modern languages. Apple's answer to this was to release a whole new language called Swift[2]. Swift is a safe, modern and powerful coding language. It contains some of the best features from other languages such as types, generics and optionals. All this while being completely interoperable with Objective-C meaning an app can be written in both Swift & Objective-C. This is an important feature as it ensures that libraries written in Objective-C can still be used when writing an application in Swift.

David Bolton best illustrates the difference between Objective-C and Swift when he says

> 'Objective-C as a manual lawn mower where you sweat a lot but it gets the job done. Swift by comparison is like an electric mower and will reduce the amount of work needed. It also has better protection so you cant slice off your fingers.'[6]

Ultimately, it was decided that the app should be written in Swift. There are many benefits to be gained from Swift; modern language features, interoperability but by far the biggest factor in choosing to use Swift was its focus on safety. Swift eliminates entire classes of unsafe code. For example, variables are always initialized before use, arrays and integers are checked for overflow, and memory is managed automatically. Thus, just by choosing Swift, we can ensure a higher level of software quality (than we could otherwise get with Objective-C) which is one of the non functional requirements listed in Section 3.

It should be noted that Swift is a new language which is constantly influx as apple continues to work on it but we believe that the benefits of using Swift (mentioned earlier) outweigh the

[1] Apples integrated development environment (IDE)
[2] https://developer.apple.com/swift/

extra development time that may be incurred. Issues we came across while using Swift will be detailed in Section 5.6, Problems Encountered.

5.2.2 Design Patterns

There are several design patterns that occur with great frequency in iOS development. Some of the these patterns are detailed below. An understanding of these concepts will aid in the readers understanding of some of the implementation details in Section 5.4.

Model-View-Controller (MVC)

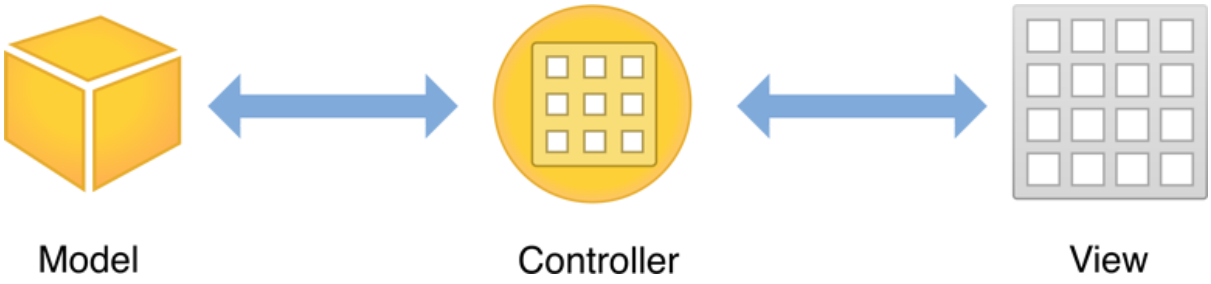


Figure 5.1: MVC Illustration[4]

The Model-View-Controller pattern is the standard approach to building iOS apps. MVC assigns the objects in an app to one of three roles: model, view, or controller. In this pattern, models keep track of the apps data, views display the user interface and content and controllers manage the views. By responding to user actions and populating views with content from the data model, controllers serve as a gateway for communication between the model and views.[4]

A fundamental flaw with the MVC structure is the over reliance on the controller. Controllers are responsible for managing the view hierarchy of the view they own. They respond to the view loading, appearing, disappearing, and so on. They also tend to get laden down with the logic that won't fit in any of the other components(model or views)[23]. For example, network calls. We can try to put them in the model objects, but that can get tricky because network calls should be done asynchronously, so if a network request outlives the model that owns it, we lose the data. That leaves us with the view and the controller. Network code should not be in the view as that would violate the single responsibility principle of Object Orientated Design[3]. In the end, we are left with the controller. This leads to what people sometimes call 'Massive View Controller'[23]. Massive view controllers are difficult to maintain and difficult to test because they have so many possible states. In order to solve this, we introduce the idea of Model-View-ViewModel (MVVM)

Model-View-ViewModel (MVVM)

Under MVVM, the view and view controller become formally connected; we treat them as one. Views still dont have references to the model, but neither do controllers. Instead, they reference the view model. With this augmented version of MVC, we have an alternative option for any code that would otherwise have gone in the controller such as the network code (mentioned

[3]The single responsibility principle states that every class should have responsibility over a single part of the functionality provided by the software.

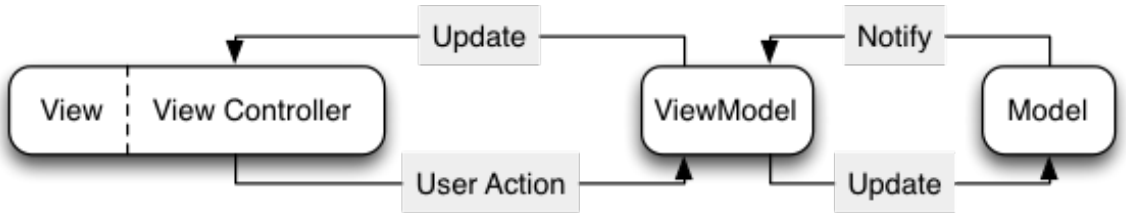


Figure 5.2: MVVM Illustration[22]

earlier) as well as validation logic for user input and presentation logic for the view and other miscellaneous code.

With MVVM, view controllers themselves become less bloated and since the view model contains all the presentation logic and doesnt reference the view, it can be fully tested programmatically.[23]

Delegation

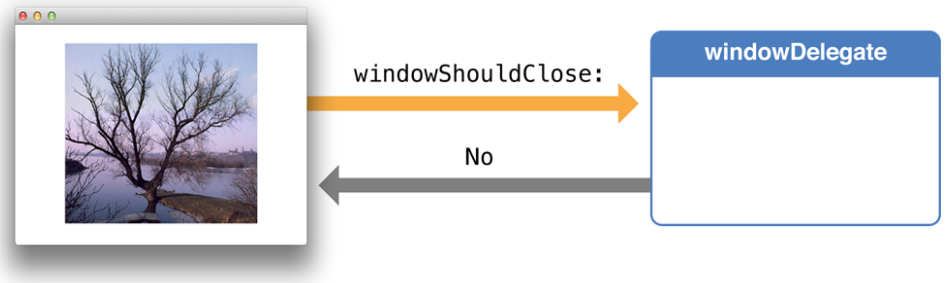


Figure 5.3: Delegation[4]

Delegation is a simple and powerful pattern in which one object in an app acts on behalf of, or in coordination with, another object. The delegating object keeps a reference to the other object the delegate and at the appropriate time, the delegating object sends a message to the delegate. The message informs the delegate of an event that the delegating object is about to handle or has just handled. The delegate may respond to the message by updating the appearance (or state) of itself or of other objects in the app, and in some cases it will return a value that affects how an impending event is handled.[4]

A common design uses delegation to allow a child view to communicate with the parent view controller. Specific occurrences where this pattern was an appropriate solution will be detailed in Section 5.5, Implementation details.

5.3 Implementation Methodology - Kanban

In order to ensure the quality and timely delivery of the the project, an implementation methodology is required. There are many methodologies available but we have to factor in that there is only one developer on the project. After researching various methodologies, we decided to proceed using the Kanban methodology. The Kanban methodology matches the amount of work in progress to the team's capacity.[34]

In our case, the developer along with the members of the focus group (from the design process mentioned in Section 4) reviewed the requirements, selected the core features needed for a minimum viable product and created user stories. Each user story detailed a specific action that a user of the system would be able to accomplish, how important it is, how long development should take and how we may test it upon completion.

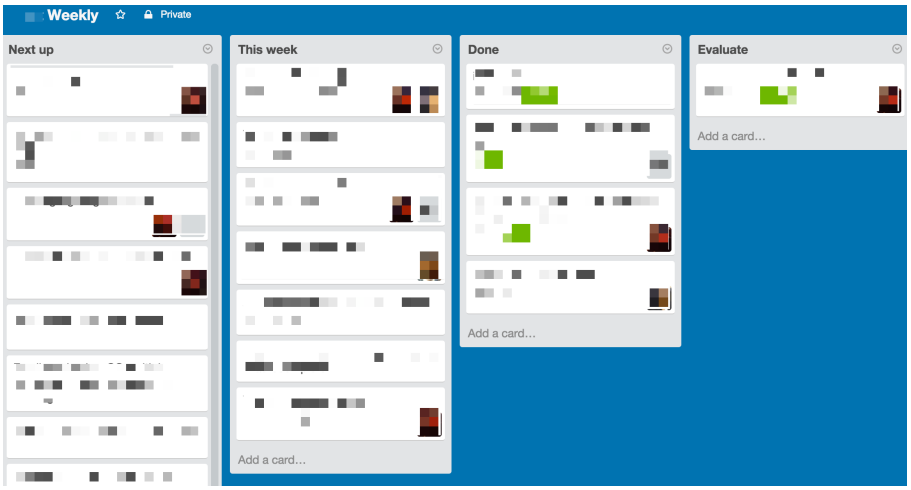


Figure 5.4: Digital Kanban Board Using Trello

As you can see in Figure 5.5, we used Trello[4] as a digital kanban board and organized the user stories under four different card headings namely 'Next Up', 'This Week', 'Done' and 'Evaluate'. Every week, we would evaluate how much progress was made from the week before and use the estimated development time of the user stories in 'Next Up', to form a new task list (ranked in terms of priority) for the 'This Week' card. During the week, a tasks would be selected from the top of the task list and would be worked on till completion. After which, the next task is selected and worked on.

This methodology was a great choice as it ensured constant progress was made throughout the duration of the project and the implementation was finished just in time. We attribute the success of the methodology to that fact that it creates 'efficiency through focus'. It is known that the more work items ongoing at any given time, the more context switching, which slows down progress.[34] Kanban and the accompanying board ensured that there was only one ongoing task at any given time. The Kanban board also helped the us visualize how much progress was being made so that we could adjust accordingly, if we ever ended up behind schedule.

[4]Web-based project management application - http://trello.com

5.4 Tools & Technologies

Listed in this section are some of the tools and technologies that were used during the implementation of the project.

5.4.1 Git - Version Control

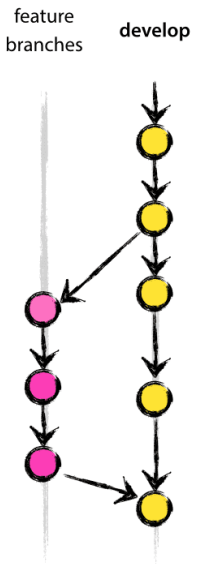


Figure 5.5: Git Workflow[16]

The version control system used for this project is Git[5]. Git is a distributed revision control system. Git's main strength is it's support for distributed, non-linear workflows[46], a feature we took advantage of during implementation.

In our workflow, The repository has a develop branch which always reflects a state with the latest delivered development changes. When a new user story / feature is to be implemented, we create a new branch for this feature. When the task is completed we merge the feature branch back into develop. In the occurrence that the development of a feature has to be paused to prioritize another feature's development, we simply switch branches and the workflow continues.

5.4.2 Travis CI - Continuous Integration

Travis CI[6] is an continuous integration service used to build and test projects hosted at GitHub. Once configured with a repo, Travis CI automatically detects when a commit has been made and pushed. Each time this happens, it will try to build the project and run the test suite. If the build fails or a test case fails, Travis sends an email to the containing the test results to the developer.

```
language: objective-c
xcode_workspace: Slang.xcworkspace
xcode_scheme: Slang
xcode_sdk: iphonesimulator
```

Figure 5.6: Travis Config File

[5]http://git-scm.com/
[6]http://en.wikipedia.org/wiki/Travis_CI

Travis CI is configured by adding a file named '.travis.yml' to the root directory of the GitHub repo. The travis configuration file for our project is shown in Figure 5.7. Although the project is written in swift, Travis CI documentation sats that the language should be set set to 'objective-c' in the config file.

The integration Travis and Github provides a very powerful workflow as the developer will easily be aware of any commits that either don't build or have failed tests. The most recent status of a branch can also be seen which is useful when the developer is revisiting an old branch and doesn't remember what state it is in.

5.4.3 Xcode - iOS Development Environment

Xcode[7] is an integrated development environment (IDE) containing a suite of software development tools developed by Apple for developing software for OS X and iOS. The current version is the 6.x series. Along with writing the application, Xcode includes a Mac-based iOS Simulator which the developer can build, install, run, and debug apps on.

5.4.4 Cocoapods - Dependency Management

```
source 'https://github.com/CocoaPods/Specs.git'
platform :ios, '8.0'
use_frameworks!

def import_pods
    pod 'pop'
    pod 'Cartography'
    pod 'SwiftyJSON'
    pod 'AHKBendableView', :git => 'https://github.com/fastred/AHKBendableView.git'
end

target :"Slang" do
    import_pods
    pod 'Reveal-iOS-SDK', :configurations => ['Debug']
end

target :"SlangTests" do
    import_pods
    pod 'Quick'
    pod 'Nimble'
end
```

Figure 5.7: Slang's Podfile

Cocoapods[8] is the dependency manager for for OS X and iOS development. It helps developers install dependencies (e.g. libraries) for an application by specification of dependencies rather than manually copying source files. The process goes as so; the developer creates a list of dependencies(using a podfile), cocoapods downloads the required sources, and configure the existing Xcode project in such a way to be able to use the libraries. Developers can also by this process, specify specific versions of libraries to be used or even update to the latest version.

An added benefit of using the cocoapods was it enabled us to share code between the application and the test suite. This can be seen in Figure 5.8 which is the podfile used in our implementation.

[7] http://en.wikipedia.org/wiki/Xcode
[8] http://en.wikipedia.org/wiki/CocoaPods

5.4.5 Reveal - Visual Debugger

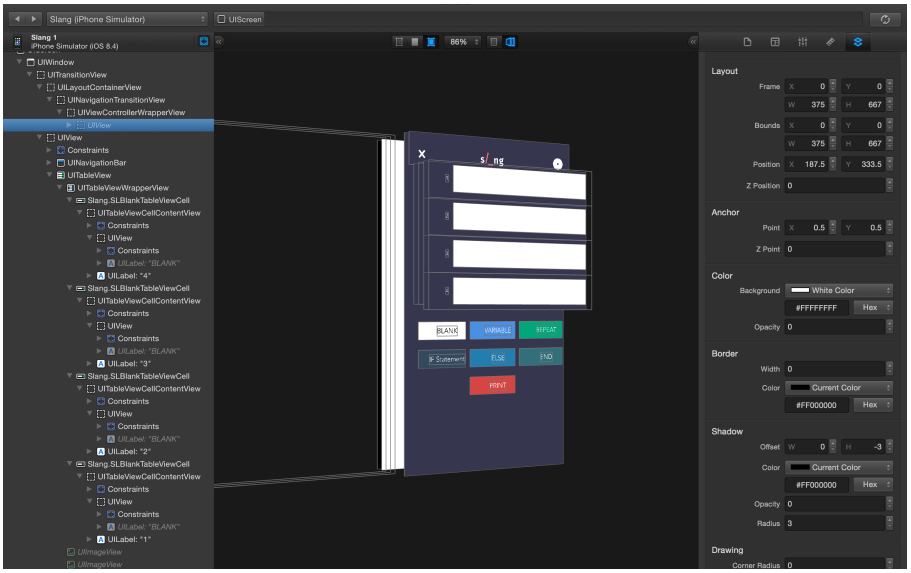


Figure 5.8: Reveal App Screenshot

A big pain point when developing on iOS is debugging views and visual layouts. To help with this, we used Reveal[9], it's an extremely powerful visual debugger.

Reveal, is an extremely powerful visual debugger. With it, once can visually explode the layers of the current screen giving you a birds-eye view of every UI element. Figure 5.9 shows Reveal applied on the VPL Interface. Each layer is selectable giving you quick access to the hierarchy and constraints that positioned that view.

5.4.6 Trello - Task Manager

Details about this tool have been mentioned in Section 5.2, Implementation Methodology.

5.4.7 Parse - BackEnd As A Service

Details about this technology have been mentioned in Section 4.5, System Design.

5.4.8 Firebase - BackEnd As A Service

Details about this technology have been mentioned in Section 4.5, System Design.

[9]http://revealapp.com/

5.5 Third Party Libraries

Listed in this section are the libraries used in the development of the project. For each library, I explain their purpose and detail why we decided to use it in the development of the application.

5.5.1 Pop - Animation Library

Animations are an important part of any application. Well executed animations not only delight a user but also help reinforce a user's mental model of your application. The iOS SDK provides a View Animation API that includes a method for spring animations, Figure 5.10. Spring animations aim to recreate animations that behave like a spring. They are by far the most realistic animation type.

```
class func animateWithDuration(_ duration: NSTimeInterval,
                         delay delay: NSTimeInterval,
         usingSpringWithDamping dampingRatio: CGFloat,
          initialSpringVelocity velocity: CGFloat,
                        options options: UIViewAnimationOptions,
                     animations animations: () -> Void,
                     completion completion: ((Bool) -> Void)?)
```

Figure 5.9: UIView Spring Animation Method API

Unfortunately, the spring animation api provided by Apple is rather subpar. The animations created do not move as smoothly or as organically as they should. The issue lies in the method parameters. As we can see from Figure 5.11, to create a spring animation, a duration must be specified. In the underlying implementation the properties of the animation are configured to suit the specified duration. In other to get the best animation effect, one would have to play with different durations as well as the initial values of damping and velocity. It's either that or solve the mathematical equation that models the movement of a spring.

Neither of those options are ideal which is why Facebook open sourced Pop[10], their animation library. By looking at Figure 5.11 we can see that with Pop, we don't need to set a duration. We simply modify the spring properties and duration is computed from that.

```
let anim = POPSpringAnimation()
anim.springBounciness = 2
anim.springSpeed = 5
```

Figure 5.10: POP Library Code Snippet

[10]https://github.com/facebook/pop

5.5.2 Cartography - Declarative Auto Layout

The standard way to positions views is with AutoLayout. AutoLayout provides a flexible and powerful system that describes how views and their content relate to each other and to the superviews they occupy.[36] AutoLayout works by the developer setting up constraints on views. Constraints are rules that describe how one views layout is limited with respect to another. For example, a view might occupy only the left half of the screen, or two views might always need to be aligned at their bottoms. At run time, the constraints are transformed into a set of simultaneous linear equations then solved and the results applied to views by setting the view's frame[11]

```
containerView.addConstraints(NSLayoutConstraint.
    constraintsWithVisualFormat("V:|[nameInfoLabel]|",
    options: nil,
    metrics: nil,
    views: views))
```

Figure 5.11: Adding Constraints using the Standard API

The standard way of adding a constraint can be seen in Figure 5.12. There are two issues with the above code snipper. Firstly, the code is quite verbose and it isn't immediately clear what view layout the constraint is describing. The other issue is that, the constraint is defined using a string; for that reason, we will be unable to detect any errors in that line of code at compile-time.

```
constrain(nameInfoLabel) { view in
    view.centerX == view.superview!.centerX
    view.height == view.superview!.height
}
```

Figure 5.12: Adding Constraints using Cartography

Figure 5.12 shows the same constraints in Figure 5.11 but written with Cartography[12] instead. As you can see, with cartography the code is much more concise and any errors with the layout will be caught at compile-time.

[11] A view's frame is it's x & y origin as well as it's size
[12] https://github.com/robb/Cartography

5.5.3 SwiftyJSON - JSON Data Extraction

Swift is very strict about types. But although explicit typing is good for avoiding bugs, it becomes painful when dealing with JSON and other areas that are, by nature, implicit about types. This results from the fact although we are using swift, the iOS SDK and its libraries are still in Objective-C which is not statically typed.

```
let JSONObject: AnyObject? = NSJSONSerialization.JSONObjectWithData(data,
options: nil, error: nil)

if let statusesArray = JSONObject as? [AnyObject],
   let status = statusesArray[0] as? [String: AnyObject],
   let user = status["user"] as? [String: AnyObject],
   let username = user["name"] as? String {
    println(username)
}
```

Figure 5.13: Extracting Data from JSON using the Standard API

In Figure 5.14, we attempt to retrieve a user's name from a JSON file. Due to type system, we have to use Optional Chaining[13] in order to safely extract the data.

```
let json = JSON(data: dataFromNetworking)
if let username = json[0]["user"]["name"].string{
  println(username)
}
```

Figure 5.14: Extracting Data from JSON using SwiftyJSON

In Figure 5.15, we perform the same action as in Figure 5.14 but written using the SwiftyJSON[14] API. SwiftyJSON does the underlying optional chaining so that we can write much clearer code while ensuring we extract data from the JSON safely.

[13]Optional chaining is a process for querying and calling properties, methods, and subscripts on an optional that might currently be nil

[14]https://github.com/SwiftyJSON/SwiftyJSON

5.6 Implementing The Visual Programming Language

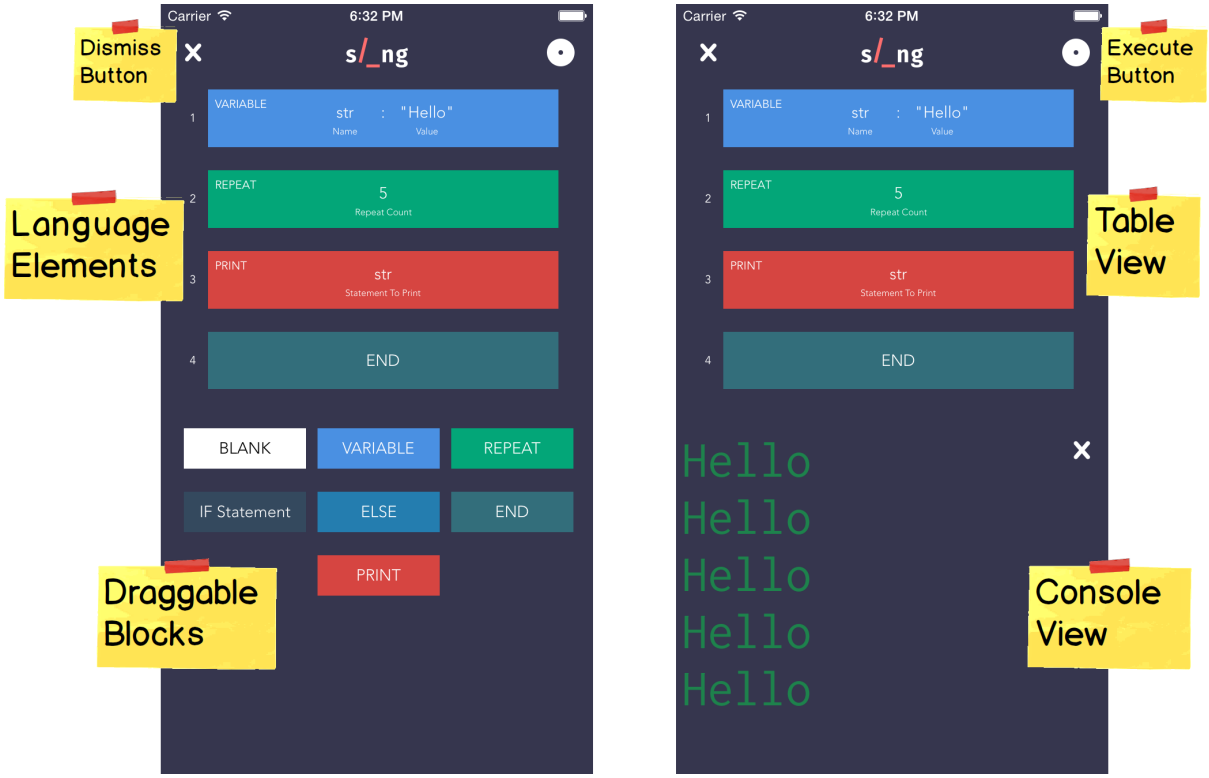


Figure 5.15: VPL Interface Implemented

Figure 5.16 shows the implemented VPL interface. This was the most challenging aspect of the app implementation. There are many individual components involved with this interface. Below, we will take each component in turn and detail how it works. At the end, we will then explore how the components work together to form a cohesive system.

5.6.1 Design Changes

The execute button is at the top rather than next to the draggable blocks (as shown in Figure 4.6) as during implementation, we realized that when the console view is shown, the user cannot click on the execute button (to run any changes) without closing the console view.

Another design change is that we added a 'blank' language element. Initially we designed the language elements to be swipepable and users could use that action in order to remove a language element but we decided that this was not required for the minimum viable product.

5.6.2 Handling State

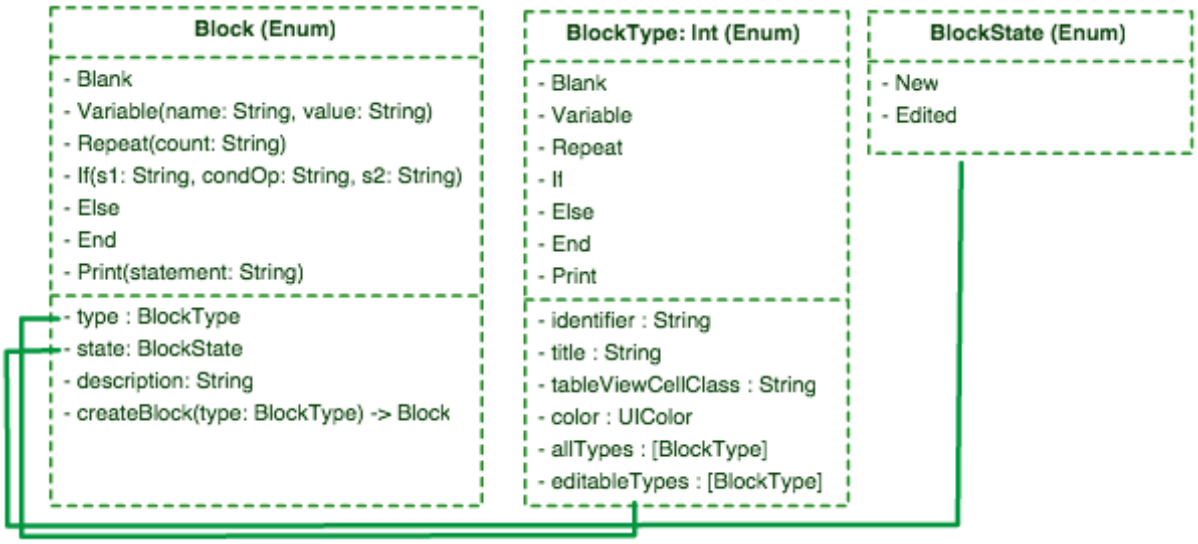


Figure 5.16: Enumeration Objects

Handling state properly is an important thing to do in order have an easily maintainable and extendable class or codebase. In Figure 5.15, we can see that we have different types of draggable blocks and language elements. How can be best keep track of this different states in a clean an understandable way?

We decided that the best way would be to use Enums. In Swift, Enums represent a finite number of 'states of being', which fits our finite number of language elements. Another benefit of using Enums is that we can take advantage of the fact that can define Swift enumerations to store associated values of any given type, and the value types can be different for each member of the enumeration if needed[15]. An example of creating a Block of type 'Variable' and giving it its associated values (language element arguments) can be seen in Figure 5.17

```
let block = Block.Variable(name: str, value: "Hello")
```

Figure 5.17: Example use of Associated Values

The finite nature of enums and being able to set associated values allows us to best handle each possible type of language elements and their individual properties very easily. As can be seen in Figure 5.16, we have the main enum, Block, which has the language elements as cases and properties such as type, which links to the BlockType Enum and state which links to the BlockState Enum.

[15]Also known as discriminated unions, tagged unions, or variants in other programming languages.

5.6.3 Draggable Blocks

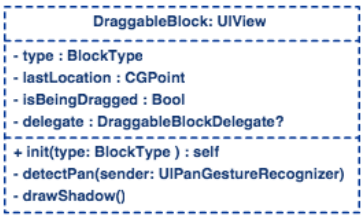


Figure 5.18: Draggable Block Class

The purpose of the draggable blocks are to allow the user to place language elements onto the tableview by dragging and dropping a block onto the tableview. Each block has a 'blockType', which holds all the properties such as the title and color. We use the type value to configure the content of the view.

The most important aspect of Draggable Blocks is the ability for it to be dragged around the screen. This was achieved using the UIPanGestureRecognizer. UIPanGestureRecognizer is a concrete subclass of UIGestureRecognizer that looks for panning (dragging) gestures. A panning gesture is continuous. It begins (UIGestureRecognizerStateBegan) when a finger has moved enough to be considered a pan. It changes (UIGestureRecognizerStateChanged) when a finger moves while pressed down. It ends (UIGestureRecognizerStateEnded) when the fingers is lifted.[3]

```
func detectPan(sender: UIPanGestureRecognizer) {
    if sender.state == .Ended {
        delegate?.draggableBlock(
        panGestureDidFinishWithDraggableBlock: sender.view as!DraggableBlock)
        isBeingDragged = false
        return
    }
    var translation  = sender.translationInView(self.superview!)
    self.center = CGPointMake(lastLocation.x + translation.x,
    lastLocation.y + translation.y)
    isBeingDragged = true
}
```

Figure 5.19: 'detectPan' method of the DraggableBlock Class

In the detectPan method of the DraggableBlock class, we handle what the different actions that should occur at each gesture state. While the state is either UIGestureRecognizerStateBegan or UIGestureRecognizerStateChanged, we create a new CGPoint (x,y) using the 'panGesture' translation and the 'lastLocation' of the block; this will be the new center of the block and the block will move accordingly.

When the gesture state becomes UIGestureRecognizerStateEnded, we call a method on the delegate and send the block as an argument. This is essentially how the block communicates to its delegate that its pan gesture has ended.

5.6.4 Language Elements

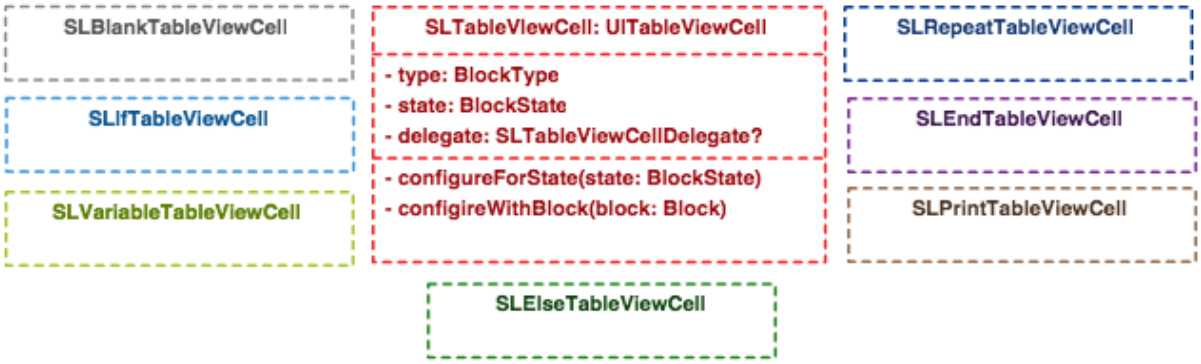


Figure 5.20: SLTableViewCells

The Language Elements in Figure 5.15 are cells in a UITableView[16]. We can customize the cells in a table view by subclassing UITableViewCell. Before we create individual subclasses for each language element, we create an abstract class called 'SLTableViewCell' that all the language element cells will inherit from. Having an abstract class was essential as it helps us encapsulate the common features of the language elements as well as prevents us from violating the 'Don't Repeat Yourself' principle[17].

SLTableViewCell contains two important methods namely 'configureForState' and 'configure-WithBlock'.

ConfigureWithBlock

The 'configureWithBlock' method takes a block argument of type Block (Enum shown in Figure 5.16). The method performs a switch on the block given and pre-fills the cell with content from the block. Figure 5.21 shows a snippet of the code in method.

```
func configureWithBlock(block: Block) {
    switch block {
        case .Variable(let name, let value):
            let cell = self as! SLVariableTableViewCell;
            cell.name = name
            cell.value = value
    }
}
```

Figure 5.21: 'configureWithBlock' method snippet

ConfigureForState

A Block can calculate its state based on its associated values. In the 'configureForState' method, we check what the state value is and perform any needed changes to the view. The implemen-

[16] A table view displays a list of items(cells) in a single column.
[17] A principle of software development, aimed at reducing repetition of information of all kinds.

tation of this method is left empty in SLTableViewCell, the subclasses are meant to override the method and provide their own implementation.

5.6.5 Communication Protocols

SLTableViewCellDelegate

Some of the language elements have arguments that can be edited by the user. When the user has finished editing an argument, the SLTableViewDelegate is used to inform other classes that a user has finished editing.

```
protocol SLTableViewCellDelegate: class {
    func tableViewCell(tableViewCellAtIndex index: Int,
    didUpdateWithBlock block: Block)
}
```

Figure 5.22: SLTableViewCellDelegate

As you can see from Figure 5.22, we send the delegate, the index of the cell as well as the Block that encapsulates the data (arguments) for that particular cell.

DraggableBlockDelegate

As mentioned earlier in the 'Draggable Blocks' subsection, the draggable block have a delegate protocol that is used to inform other classes when the drag gesture has been completed.

```
protocol DraggableBlockDelegate: class {
    func draggableBlock(panGestureDidFinishWithDraggableBlock
    draggableBlock: DraggableBlock)
}
```

Figure 5.23: DraggableBlockDelegate

As you can see from figure 5.23, we send the delegate the draggableBlock, so it can perform any actions on the view. For example, returning the block back to its original position.

5.6.6 DraggableBlocks + Language Elements

SlangViewController

Figure 5.24: SlangViewController Class

The Controller that manages the Visual Programming Language Interface is the 'SlangViewController'. The SlangViewController positions the TableView as well as the draggable block. It also acts as the delegate for the protocols mentioned above. For the SlangViewController to act as a delegate, it means that it provides an implementation for the functions shown in Figure 5.22 and 5.23.

```
func draggableBlock(panGestureDidFinishWithDraggableBlock
draggableBlock: DraggableBlock) {
    let visibleCells = tableView.visibleCells()
    for cell in visibleCells {
        let cellframe = tableView.convertRect(cell.frame, toView: self.view)
            if CGRectIntersectsRect(cellframe, draggableBlock.frame) {
                animateDraggableBlockReturn(draggableBlock)
                let iP = tableView.indexPathForCell(cell as! UITableViewCell)!
                let block = Block.createBlock(draggableBlock.type)
                viewModel.updateBlock(atIndex: iP.row, withBlock: block)
                tableView.reloadData()
                break
            }
        }
    animateDraggableBlockReturn(draggableBlock)
}
```

Figure 5.25: Implementation of DraggableBlockDelegate in SlangViewController

Once a drag gesture has been completed, the method in the DraggableBlockDelegate protocol is called. In the implementation of that method in the SlangViewController, we check for which cell, the draggable block intersected with. If the block did intersect with a valid cell, we animate the return of the draggable block to it's original position then we create a new Block (using Block.createBlock(type: BlockType)) and we store it in the SlangViewModel.

SlangViewModel

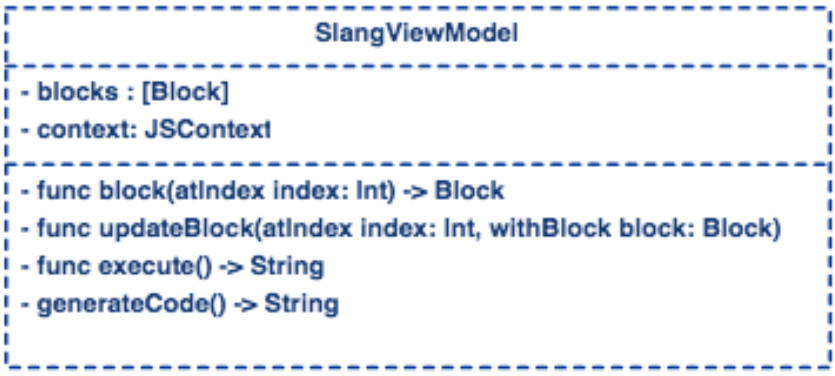


Figure 5.26: SlangViewModel Class

Guided by the MVVM pattern, mentioned is Section 5.1.2, we created a viewModel class to handle to the storage and retrieval of Blocks. As you can see from Figure 5.26, we have an array of Blocks and helper methods to access and update block in the array.

When the Table View is loaded, for each cell in the tableview, we query the viewModel for the block at the cell's index and use the SLTableViewCell 'configureWithBlock' method to set the content of the cell.

```
extension SlangViewController: SLTableViewCellDelegate {
    func tableViewCell(tableViewCellAtIndex index: Int,
    didUpdateWithBlock block: Block) {
        viewModel.updateBlock(atIndex: index, withBlock: block)
    }
}
```

Figure 5.27: Implementation of SLTableViewCellDelegate in SlangViewController

When the user edits a cell, the the method in the SLTableViewCellDelegate protocol is called. In the implementation of that method in the SlangViewController, we call the viewModel's 'updateBlock' method and the pass the index and block arguments.

5.6.7 Executing a Program

```
private func generateCode() -> String {
    var code = ""
    for block in blocks {
        switch block {
        case .Variable(let name, let value):
            code += "var \(name) = \(value);\n"
        case .Repeat(let count):
            code += "for (var i = 0; i < \(count); i++) {\n"
        case .If(let s1, let cond, let s2):
            code += "if ( \(s1) \(cond) \(s2) ) {\n"
        case .Else:
            code += "} else {\n"
        case .Print(let statement):
            code += "console.log(\(statement));\n"
        case .End:
            code += "}\n"
        case .Blank:
            code += ""
        }
    }
    return code
}
```

Figure 5.28: 'generateCode' method of SlangViewModel

The logic for executing a program is located in the 'generateCode' method of the SlangViewModel. For each block in the blocks array, we use a switch to get access to the blocks associated values and construct a string of javascript code that is equivalent to the language element logic that the block represents.

Using the Javascript bridge that iOS provides, we can pass the string of javascript and it will return the result from executing the javascript code. The result is then sent from the viewModel to the console view and then the console view appears from the bottom.

5.7 Problems Encountered

Creating the best way to handle communication between the draggable blocks, the tableview and storing the tableview state was rather challenging. I had to initially do a quick implementation at a hackathon to have an idea of all the requirements code wise. Then I rewrote that implementation from the ground up guided by the knowledge gained from the previous implementation.

Another huge challenge was my choice of using Swift, as I expected. Swift is still constantly being worked on and updated by Apple. Every few months, an update would be released,

I would update my Xcode accordingly and there would many errors all over the project due to language changes. I budgeted time for such a setback so it didn't put the project behind schedule.

Chapter 6

Testing and Evaluation

Testing and evaluation are vital parts of any software development project. It is important that the software built fulfills it's requirements and all the constraints placed on it. Testing was conducted at three levels. The first level was testing of the functional requirements using behavior driven testing and snapshot testing tools. The second, was testing against the non functional requirements and the third was user evaluation testing to evaluate whether Slang is an effective tool for learning to program.

6.1 Functional Testing

6.1.1 Behavior Driven Testing (BDT)

```
class SLBlankTableViewCellSpec: QuickSpec {
    override func spec() {
        let cell = SLBlankTableViewCell()
        let type = BlockType.Blank
        describe("type") {
            it("is SLBlank") { expect(cell.type) == BlockType.Blank }
        }
        describe("typeLabel") {
            it("should not be visible"){ expect(cell.typeLabel.alpha) == 0.0 }
            it("text should be the same as that of it's enum") {
                expect(cell.typeLabel.text) == type.title
            }
        }
    }
}
```

Figure 6.1: SLBlankTableViewCellSpec

BDT helps developers answer the question; what should I test? As the first word in BDT suggests, developers should no longer focus on tests, but should instead focus on behaviors.[17] This is an ideal method for us to conduct our testing as the in Chapter 3, we have a detailed list of requirements that specify how our application should behave. To conduct BDT, we used two libraries namely Quick[1] and Nimble[2]. Quick is a behavior-driven development framework for Swift and Objective-C. Quick comes together with Nimble a matcher framework for tests. A sample of BDT in use with Quick and Nimble can be seen in Figure 6.1.

Another benefit of BDT is that it creates very readable test specifications which helps easily identify what a failed test case means. For example, the test spec in Figure 6.1 can be described as follows:

- typeLabel__should_not_be_visible
- typeLabel__text_should_be_the_same_as_that_of_it_s_enum
- type__is_SLBlank

BDT helps a developer think more about how an object should behave (and how its interface should look) and less of how it should be implemented. By doing that, we end up with a more robust codebase, along with a great test suite.

[1] https://github.com/Quick/Quick
[2] https://github.com/Quick/Nimble

6.1.2 Snapshot Testing

Snapshot testing involves verifying that what the user sees is what you want the user to see. We conduct these tests using, FBSnapShotTestCase[3], a library released by Facebook.

FBSnapShotTestCase takes a UIView or CALayer subclass and renders it to a UIImage. This snapshot is used to create tests that compare a saved snapshot of the view/layer and the version generated by your test. When it fails, it will create a reference image of the failed test, and another image to show the difference of the two.[41] Figure 6.2 is an example of a snapshot test case.

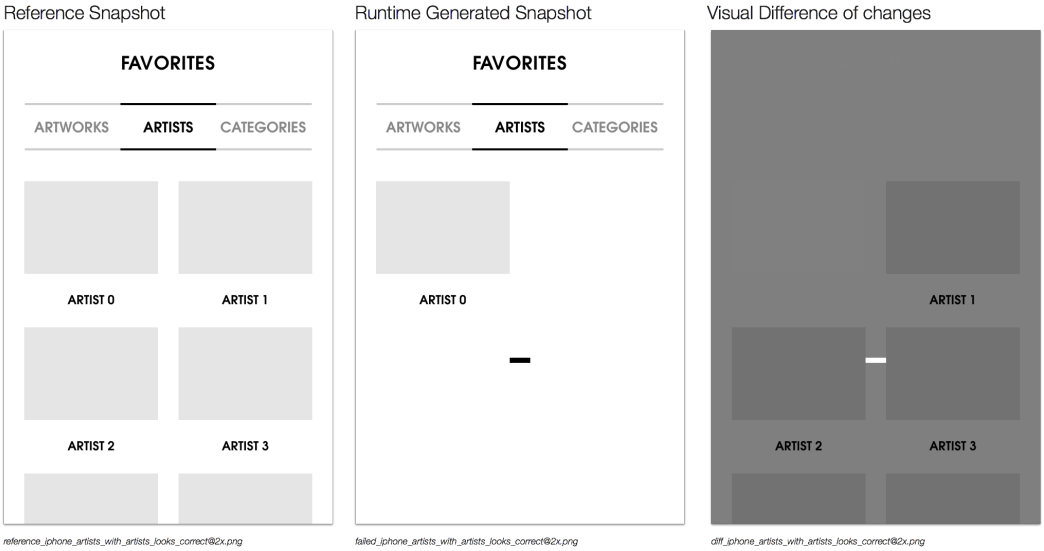


Figure 6.2: Snapshot Testcase Example[41]

The library makes the comparison by drawing both the view or layer and the existing snapshot into two CGContextRefs and doing a memory comparison of them with the C function memcmp() which makes it extremely quick.[41]

The use of snapshot tests also automates testing the user experience. An alternative would have been to create a list of actions and the expected results. This would then be given to testers to go through and ensure that expected outcomes were as expected in the app. That is a long winded task and cannot be done every time. Snapshot test cases help us continually do that set of tasks every time we test the app thereby ensuring that once a test case fails, we will know immediately.

[3]https://github.com/tomguthrie/SnapshotTestCase

6.2 Non-Functional Testing

Security

As mentioned in Chapter 3, without some sort of obfuscation, strings like API keys and other sensitive data can be extracted by utilizing various command-line tools. In order to this, we used a library called UAObfuscatedString. UAObfuscatedString[4] is a simple NSString category to hide sensitive strings from appearing in the binary.

The library works by only ever storing single characters in the binary and combines them into a string at runtime. It's highly unlikely that these single letters will be discoverable in the binary as they will be interjected at random places in the compiled code. Thus, they will appear to be randomized code to anyone trying to extract strings.[12]

Software Quality

To minimize the occurrence of bugs and improve code quality and maintainability, we took several precautions. Some of these are listed below;

- **Using a style guide** - We used the Github Swift Style Guide[5] during the implementation stage of the project. The style guide is beneficial as its rules helped ensure rigor, decreased the likelihood of programmer error, increased clarity of intent and reduced verbosity.
- **Extensive Testing** - An entire suite of test cases was created ranging from behavioral tests to user acceptance testing using snapshots.

Performance

The application's performance was tested using Xcode Instruments[6]. Instruments is a performance-analysis and testing tool for dynamically tracing and profiling OS X and iOS code. It can track user events, CPU activity, memory allocation, memory leaks and network activity.[44] We used instruments specifically to check that no methods heavily utilized the CPU and also to ensure there were no memory leaks in the application.

6.3 User Feedback Testing

The main aim of this project was to evaluate if Slang is an effective way to teach coding. In order to do this, I sourced 30 participants from the Tech@NYU organization. All 30 chosen confirmed on a short questionnaire that they had no previous programming knowledge but did have an interest in learning to code.

The participant pool of 30 individuals was split into three groups. The first group would learn to code using Codecademy's Code Hour App, the second would use Swifty and the third group

[4]https://github.com/UrbanApps/UAObfuscatedString
[5]https://github.com/github/swift-style-guide
[6]https://goo.gl/HqROoI

would use Slang. After completion, participants would then be given a short list of problems to solve / solutions to modify in the language taught be the app they used.

To keep things fair, we ensured that the participants learnt the same concepts in their individual apps. The lesson plan was constructed so as to teach the students how to print strings as well as how to add logic to programs by using if statements. Also as the questions were on the much simpler side of things, we prevented the participants from reviewing their lessons while completing the tasks.

The list of problems to solve was chosen from the Simple Programming Problems website[7]. The list is as follows;

1. Write a program that prints Hello World to the screen.
2. Modify the previous program to print your name
3. Modify the FizzBuzz program so that for multiples of three you print Fizz instead of the number and for the multiples of five you print Buzz. For numbers which are multiples of both three and five print FizzBuzz

[7]http://adriann.github.io/programming_problems.html

6.3.1 Results & Analysis

The results of the study are shown below. It is to be noted that the sole purpose of this study was to capture how effective this tools are at teaching code hence we ignored the time spent and other factors. We solely wanted to find out given the app and no further help or any changes, could the participants correctly solve the problem set given. Trying to glean any other kind of information from such a small number of people would be futile and misleading. This basic research was fundamental as no study has been compiled yet to study the effectiveness of apps for learning to code.

The Code Hour app performed the worst in this study. Only 2 participants were able to solve the first two problem sets compared to 5 in Swifty and 9 in Slang. As the third problem set was much harder, the number of people dropped for each app. Slang had the least percentage of dropout rate for problem 3.

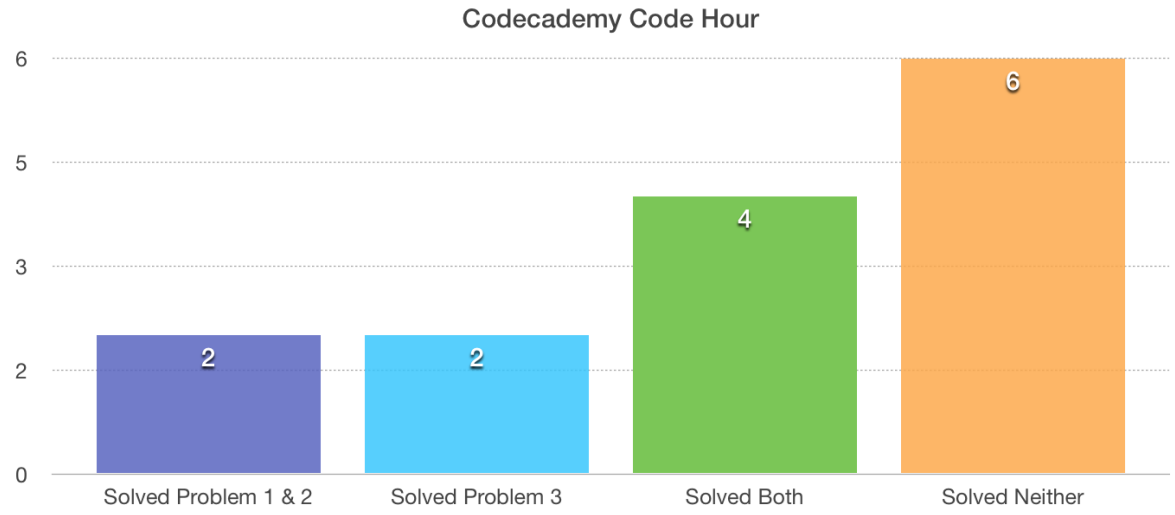
When candidates were asked later about how they performed, the code hour participants felt the content was simply too passive and as such they had an idea of what they needed to do but without being able to review, they were stuck.

Swifty participants were delighted with the app. It explained the concepts well and required some involvement from them before moving on to the next concept. From the results we see that this the more active learning process increased the success rate of completing the problem set compared to Code Hour. With Problem 3, many struggled as they understood the concept but were not sure how to fix it. This is likely due to the debugging aspect of the question which they had no avenue of learning as Swifty never required them to write any code.

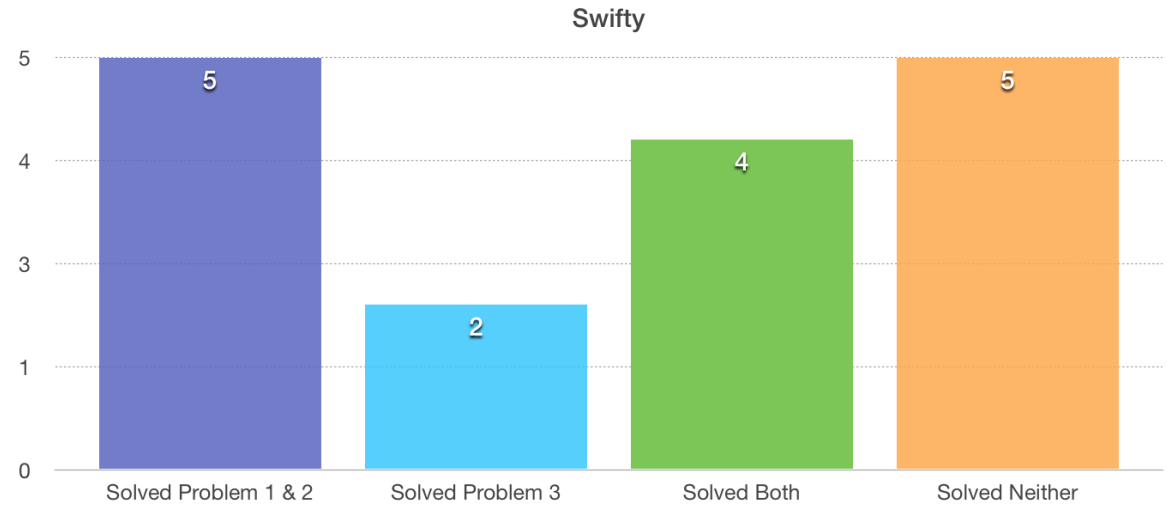
Slang participants felt well prepared for the problem 1 and 2 (shown by the high success rate - 9/10) as due to the inclusion of the VPL language they were already familiar with writing code. They felt the block system was an easy paradigm for them to understand. As expected, performance in problem 3 dropped significantly. Some participants mentioned they only got it right by experimenting and weren't entirely sure what they were doing. The others gave up. This fits the pattern we mentioned in the literature review, Chapter 2, where we noted there are two types of learners that we will need to cater for.

Conclusion

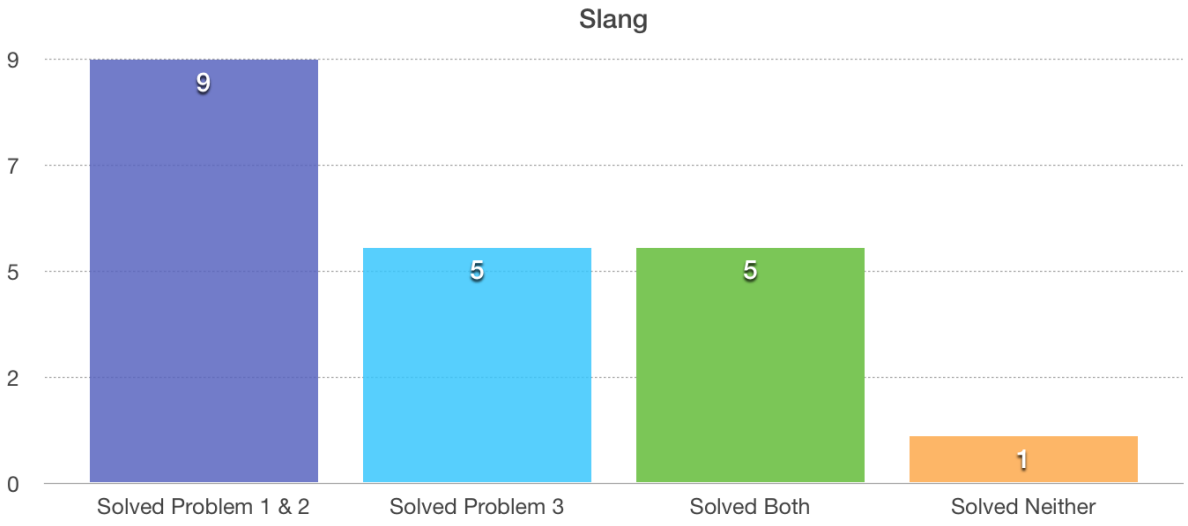
Due to our comprehensive literature, with Slang, we clearly identified a lot of the pain points that people learning to code face and thus were able to come up with requirements to help alleviate this issues. This can be seen from the high performance in problem 1 and 2 and the satisfactory performance in problem 3. There is still a lot of work to be done and more research to be done further but I believe our minimum viable product has shown that learning to code on mobile isn't impossible, but of course, special attention must be made to the little details and secondly, learners should be able to start writing their own code within the app in order for them to really learn.



(a) Code Hour Results



(b) Swifty Results



(c) Slang Results

Figure 6.3: Study Analysis

Chapter 7

Further Work & Reflections

7.1 Further Work

As the implementation stage of this project only built the minimum viable product, there is much more work to be done in building the final product. Apart from that, there are three main tasks that I believe should focused on when working on this project further.

- **Help** - Learning to program is very hard thus at every step of the way, we need to always aim to reduce the cognitive overhead of the application. We also need to look out for problem areas and provide a clear way for the user to get help.

- **From VPL to Real Code** - I believe that the mobile platform is actually a great way to teach people how to code but unfortunately teaching solely through a VPL will be rather limiting. After a few lessons, it will be much more beneficial to introduce an actual Javascript editor and teach students Javascript. Typing code on mobile isn't ideal but we can improve the experience by, for example, creating a custom keyboard that puts some of the characters like '', '', ""' within easy reach.

- **Education Consultants** - We have developed a simple education framework but we will need to conduct much more research how best to deliver our lessons to our users. Education consultants will help conduct focus groups and the like in order to identify where users get stuck or identify the best way to teach a certain concept to the user.

- **Gamifying the Process** - It is also important that our users have fun while they learn to code. In our requirements we introduce this, by creating badges that users can collect by completing lesson and challenges but we also need to look further. We need to create unique coding environments where users can create mini games, for example, using a turtle to draw both simple and complex shapes much like logo programming[1].

[1] http://www.bbc.co.uk/schools/gcsebitesize/ict/measurecontrol/1logocontrolrev1.shtml

7.2 Reflections

This project was a very hard and challenging project. A lot of thought was required beforehand and it was especially important that I validate my thoughts and choices with my users. Overall, I am happy with the progress made and the validation I gained from my users. Solving the challenges involved in teaching people how to code through an app can have a big impact on improving code literacy on our world today and I am excited to continue development on this project and hopefully eventually release it on the iOS AppStore.

Bibliography

[1] K. Academy, "Intro to js: Drawing & animation," March 2015. [Online]. Available: https://www.khanacademy.org/computing/computer-programming/programming

[2] Adobe, "Photoshop," March 2015. [Online]. Available: http://www.adobe.com/uk/products/photoshop.html

[3] Apple, "Uipangesturerecognizer," May 2015. [Online]. Available: https://developer.apple.com/library/prerelease/ios/documentation/UIKit/Reference/UIPanGestureReco

[4] ——, "Using design patterns," May 2015. [Online]. Available: https://goo.gl/ahk0On

[5] Balsamiq, "Balsamiq," March 2015. [Online]. Available: https://balsamiq.com/products/mockups/

[6] D. Bolton, "Some reasons why swift is better than objective-c," Jan 2015. [Online]. Available: http://thenewstack.io/some-reasons-why-swift-is-better-than-objective-c/

[7] B. Boulay, "Some difficulties of learning to program."

[8] J. Cao, "How to use mockups in the ux design process," February 2015. [Online]. Available: http://www.creativebloq.com/web-design/use-mockups-21514108

[9] L. Cerejo, "Design better and faster with rapid prototyping," June 2010. [Online]. Available: http://www.smashingmagazine.com/2010/06/16/design-better-faster-with-rapid-prototyping/

[10] Codecademy, "Codecademy hour of code app," March 2015. [Online]. Available: http://www.codecademy.com/hour-of-code/iphone

[11] Code.org, "K-12 curriculum philosophy and goals," March 2015. [Online]. Available: https://code.org/educate/curriculum-philosophy

[12] M. Coneybeare, "Uaobfuscatedstring," May 2015. [Online]. Available: https://github.com/UrbanApps/UAObfuscatedString

[13] N. Cook, "Javascriptcore," January 2015. [Online]. Available: http://nshipster.com/javascriptcore/

[14] D. Crow, "Why every child should learn to code," February 2014. [Online]. Available: http://www.theguardian.com/technology/2014/feb/07/year-of-code-dan-crow-songkick

[15] C. Davidson, "Cathy davidson to present on the 4th r: algorithm," 2012. [Online]. Available: http://www.cathydavidson.com/appearances/cathy-davidson-to-present-on-the-4th-r-algorithm-mozilla-fireside-virtual-chat-feb-1-open-to-the-public/

[16] V. Driessen, "A successful git branching model," Jan 2010. [Online]. Available: http://nvie.com/posts/a-successful-git-branching-model/

[17] P. Dudek, "Behavior-driven development," August 2014. [Online]. Available: http://www.objc.io/issue-15/behavior-driven-development.html

[18] T. Dunn, "Focus groups vs. usability testing - what, when and why?" July 2015. [Online]. Available: http://www.smashingmagazine.com/2010/06/16/design-better-faster-with-rapid-prototyping/

[19] Duolingo, "Duolingo," March 2015. [Online]. Available: http://www.duolingo.com

[20] H. edX, "Harvard introductory programming course," March 2015. [Online]. Available: https://www.edx.org/course/introduction-computer-science-harvardx-cs50x

[21] P. Fox, "Programming curriculum overview," March 2015. [Online]. Available: https://www.khanacademy.org/coach-res/reference-for-coaches/teaching-computing/a/programming-curriculum-overview

[22] A. Furrow, "Introduction to mvvm," June 2014. [Online]. Available: http://www.objc.io/issue-13/mvvm.html

[23] ——, "Model-view-viewmodel for ios," Jan 2014. [Online]. Available: http://www.teehanlax.com/blog/model-view-viewmodel-for-ios/

[24] P. Guo, "Why python is a great language for teaching beginners in introductory programming classes," May 2007. [Online]. Available: http://pgbovine.net/python-teaching.htm

[25] S. Hurff, "How to design for thumbs in the era of huge screens," October 2014. [Online]. Available: http://www.smashingmagazine.com/2010/06/16/design-better-faster-with-rapid-prototyping/

[26] M. M. Lab, "Scratch homepage," March 2015. [Online]. Available: https://scratch.mit.edu/

[27] M. C. Linn and J. Dalbey, "Cognitive consequences of programming instruction."

[28] A. Maurya, "Minimum viable product," March 2015. [Online]. Available: http://practicetrumpstheory.com/minimum-viable-product/

[29] Mixpanel, "ios 8 adoption," May 2015. [Online]. Available: https://mixpanel.com/trends/

[30] J. Nielsen, "10 usability heuristics," January 1995. [Online]. Available: http://www.nngroup.com/articles/ten-usability-heuristics/

[31] M. OCW, "Mit introductory programming courses," March 2015. [Online]. Available: http://ocw.mit.edu/courses/intro-programming/

[32] B. U. of California, "Learning goals/outcomes," March 2015. [Online]. Available: http://teaching.berkeley.edu/learning-goalsoutcomes

[33] D. Perkins, C. Hancock, R. Hobbs, F. Martin, and R. Simmons, "Conditions of learning in novice programmers."

[34] D. Radigan, "A brief introduction to kanban," May 2015. [Online]. Available: https://www.atlassian.com/agile/kanban

[35] J. Rogalsku and R. Samurcay, "Acquisition of programming knowledge and skills."

[36] E. Sadun, *iOS Auto Layout Demystified*, 2nd ed. Addison-Wesley Professional, 2013.

[37] Sketch, "Sketch," March 2015. [Online]. Available: http://www.bohemiancoding.com/sketch/

[38] ——, "Sketch features," March 2015. [Online]. Available: http://www.bohemiancoding.com/sketch/features/

[39] W. Staff, "Infographic: Sketch vs. photoshop," March 2015. [Online]. Available: http://www.webdesignerdepot.com/2015/03/infographic-sketch-vs-photoshop/

[40] Swifty, "Swifty," March 2015. [Online]. Available: http://www.swifty-app.com/

[41] O. Therox, "Snapshot testing," August 2014. [Online]. Available: http://www.objc.io/issue-15/snapshot-testing.html

[42] Udacity, "Udacity - learn programming in html, css, javascript, python, java & more," March 2015. [Online]. Available: https://itunes.apple.com/gb/app/udacity-learn-programming/id819700933?mt=8

[43] R. Vesselinov and J. Grego, "Duolingo effectiveness study," December 2012. [Online]. Available: http://static.duolingo.com/s3/DuolingoReport_Final.pdf

[44] Wikipedia, "Instruments (application)," August 2014. [Online]. Available: http://goo.gl/kGN1w6

[45] ——, "Experiential learning," May 2015. [Online]. Available: http://en.wikipedia.org/wiki/Experiential_learning

[46] ——, "git (software)," May 2015. [Online]. Available: http://en.wikipedia.org/wiki/Git_(software)

[47] ——, "Visual programming language," March 2015. [Online]. Available: http://en.wikipedia.org/wiki/Visual_programming_language