# Automated Cross-Platform Code Synthesis from Web-Based Programming Resources

Antuan Byalik

Thesis submitted to the Faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of

Master of Science
in
Computer Science and Applications

Eli Tilevich, Chair
Dhruv Batra
Sharath Raghvendra

July 2, 2015
Blacksburg, Virginia

Keywords: Recommendation Systems; Code Synthesis; Mobile Computing; Android; iOS; Java; Swift

# Automated Cross-Platform Code Synthesis from Web-Based Programming Resources

Antuan Byalik

(ABSTRACT)

For maximal market penetration, popular mobile applications are typically supported on all major platforms, including Android and iOS. Despite the vast differences in the look-and-feel of major mobile platforms, applications running on these platforms in essence provide the same core functionality. As an application is maintained and evolved, programmers need to replicate the resulting changes on all the supported platforms, a tedious and error-prone programming process. Commercial automated source-to-source translation tools prove inadequate due to the structural and idiomatic differences in how functionalities are expressed across major platforms.

In this thesis, we present a new approach—*Native-2-Native*—that automatically synthesizes code for a mobile application to make use of native resources on one platform, based on the equivalent program transformations performed on another platform. First, the programmer modifies a mobile application's Android version to make use of some native resource, with a plugin capturing code changes. Based on the changes, the system then parameterizes a web search query over popular programming resources (e.g., Google Code, StackOverflow, etc.), to discover equivalent iOS code blocks with the closest similarity to the programmer-written Android code. The discovered iOS code block is then presented to the programmer as an automatically synthesized Swift source file to further fine-tune and subsequently integrate in the mobile application's iOS version. Our evaluation, enhancing mobile applications to make use of common native resources, shows that the presented approach can correctly synthesize more than 86% of Swift code for the subject applications' iOS versions.

This thesis is based on research accepted for publication to GPCE'15.

This research is supported by the National Science Foundation through Grant CCF-1116565

# Dedication

*I dedicate my thesis to my family and close friends,*
*especially my parents, Rita and Vladimir, my brother, Yan, and my grandfather Michael.*
*This thesis is a realization of your immeasurable effort instilling strong life values in me.*
*"If I have seen further, it is by standing on the shoulders of giants."*

# Acknowledgments

If my journey to this master's thesis would have been a sole venture, I have no doubt it would have ended before ever getting started. Many people of varying technical and non-technical expertise provided much needed assistance, some at times when I was sure failure was on the horizon. I would like to take a moment to individually thank some of the most influential people who have spent an enormous amount of time helping me by facilitating a thirst for knowledge and learning, instilling valuable life principles, and teaching me the skills needed to succeed.

First and foremost, I would like to thank my parents, Rita Gorochovskaya and Vladimir Byalik. The sacrifices you made, and continue to make, to see both my Brother's and my success do not go unnoticed. Early on, you both instilled the most basic and valuable of principles unto me: family. Regardless the circumstance or current dilemma, family is above all else. But you didn't stop there. You moved on to perseverance, self-motivation, the value of intellect, and naturally, to ambition. It wasn't until some half-way through my undergrad here at Virginia Tech that I realized the potential I had because of what you had done for me. To me now there is but one central goal, that no matter what I do in life, I do it such that: I always strive for excellence, I always aim to make you proud, and I always remember my family that was there for me at every step.

I would like to thank my brother, Yan Byalik, and grandfather, Michael Byalik. You both solidified principles I didn't quite get on the first try from my parents and instilled new ones. With my family moving to the United States due to the impending collapse of the former Soviet Union, I was born a first generation American and could have never learned the culture of my family. You both, along with my parents, helped me grow to understand where I came from while also assimilating into a brand new culture. I would not be the man I am today without your guidance and I am forever in awe of the time and effort you put towards making me succeed. Know that I recognize your sacrifices and support, and I stand here incredibly grateful. Alongside my parents, I aim to make you proud above all others.

I would like to thank my closest friend since before I ever imagined writing a program, Brendan Avent. While there are a lot of positive things to say about you and what you've done for me, I will stick to the presently relevant ones here. I knew we were destined for greatness since the first science project we did together in 6th grade. You supported me

when I needed it and challenged me when I needed it, but one of the most important things you did is shatter my concept of greatness. Rather than comparing myself to the average CS student, I found myself largely comparing myself to what Mr. four majors was doing; that made me work harder, ultimately helping me strive not for mediocrity but for excellence and greatness at all stages my life. I look forward to our next adventures and also to what you say about me in your PhD dissertation acknowledgments.

I would like to thank Sean Butler and Igor Janjic. While there is not enough space to thank all of my close friends who have supported me through undergraduate and graduate studies, you two have always been there for me to vent or share a night out. You kept me disciplined to working out and helped me release frustration at academics or general stresses. Arguably most important though, was your ability to remind me that excelling academically while maintaining a social life of occasional de-stress is not impossible.

Among the many faculty who guided me along my research studies, my adviser, Dr. Eli Tilevich, has been a plethora of information and guidance. Even as an undergrad taking your classes, I realized you had an aura of excellence around you. Stemming from similar roots, you never settled for average, but instead challenged yourself for greatness. This quickly made me respect you both as a researcher and as a person I could turn to for advice. Among your arsenal of skills, I would like to point two unrelated talents. First, your ability to write a technical manuscript or edit existing ones. You command an amazing vocabulary and general knowledge of how to make English sound brilliant, made only more impressive by your native language being Russian. After many excellently and last minute fluffed grade school and collegiate reports, I assumed I had a gift for turning an thoughts into beautiful words. As it turns out, I was an amateur after watching you rebuild some of my initial research papers. Secondly, your ability to sell an idea. Writing it nicely alone isn't enough, you must figure out how to sell the idea to your intended audience. Again, I was in shock at how you managed to spin ideas such that they looked like the greatest concept ever. We have shared many late nights at the CRC working on various research projects or simply talking about new and cool ideas. This thesis' underlying foundation began as one of your cool ideas that I tried to make a reality. Without your guidance and expertise, I would never have made it this far in my post-graduate studies.

I would like to thank the two members of my committee, Dr. Dhruv Batra and Dr. Sharath Raghvendra. Dr. Batra has been an inspiration for my intellectual growth since I first took that intro Machine Learning class. I quickly realized just how smart other people were and it only solidified my thoughts for graduate studies and excelling as high as I could. Dr. Raghvendra helped me reach this same realization in the field of online algorithms only a short while later. I again felt myself dipping my feet in what was a challenging and interesting field. I was beginning to really enjoy research and the notion of more advanced studies. Both members of my committee have been incredible resources for information, side research projects that I didn't have time for but wanted to do anyways, and general guidance in career choices. I am forever grateful for your contributions and general assistance towards helping me succeed.

I would like to thank Dr. Calvin J. Ribbens, Dr. T. M. Murali, Dr. Edward A. Fox, Dr. Kwok Ping Tsang, who were also supportive and influential during both my undergrad and graduate studies here at Virginia Tech. I would like to thank Sanchit Chadha who has been on board for various projects during my graduate research studies and was excellent in some of the underlying coding for the implementation in this thesis. Also, I would like to thank Mr. Greg Farris, Mrs. Terry Arthur, Ms. Libby Bradford, and Mrs. Sharon Kinder-Potter who were outstanding academic advisers, mentors, and friends throughout my time at Virginia Tech. From industry experience, I would like to single out David C. Morris, who even back in high school, helped me learn and excel in the industry side of computer science.

Thank you to my family, friends, various professors, industry members, and staff here at Virginia Tech that have helped me grow and learn as the years went by. I am very thankful to have had your support throughout the years and know that I take the following quote to heart: "If I have seen further, it is by standing on the shoulders of giants."

# Contents

# List of Figures

# List of Tables

# Chapter 1

# Introduction

## 1.1 Realities of Modern Mobile Software Developer

The mobile application market remains fragmented, with several major platforms, including Android, iOS, and Windows Phone, competing to dominate market share. Nevertheless, shrewd mobile software vendors commonly support their popular applications on all major platforms to maximize their customer base. On each supported platform, an application replica essentially delivers an identical set of functionality, albeit within the conventions and formats of the platform at hand. For example, a mobile application with a map component would use Google Maps on Android and Apple Maps on iOS Devices. Even though from the end-user's perspective, the provided map feature provides identical functionality on both platforms, from the software engineering perspective, implementing the same feature on different platforms requires the use of vastly dissimilar languages, APIs, and software architectures. For instance, Android applications are written in Java using the Android standard library, in which UI events are expressed by means of callbacks; meanwhile, iOS applications are written in Swift using the iOS standard library, in which UI events are expressed by means of delegates.

As a mobile application evolves with new features and functionalities, the mobile developers must replicate the changes on all supported platforms. Having overcome the challenges of ascertaining the correct program logic and implementation details on one platform, the developer has no choice but to repeat the same tedious and error-prone process on all the remaining supported platforms. In other words, the developer is unable to leverage the expertise gained by undertaking a programming task on one platform to facilitate the performance of that same task on other platforms. What if it were possible to automatically glean the knowledge acquired by adding a feature to an application on a source platform to semi-automatically synthesize the code required to add the same feature on target platforms?

## 1.2 Native-2-Native

### 1.2.1 Design

In this thesis, we present `native-2-native`—a novel approach that applies a code synthesis algorithm to discover publicly available Swift code blocks whose semantics are equivalent to a Java code block written for the Android platform. The approach starts with the programmer adding a feature to an application's Android version. The approach then logs the manually written Java/Android code and uses it to search the web for the available Swift/iOS code that implements the same functionality. A ranking algorithm applies a high-dimensional feature vector to select the code block whose functionality is the closest to the original Java/Android code. The selected code then parameterizes a code generator that synthesizes a semantically correct Swift source file that can be included into the the application's iOS version, requiring minimal manual fine-tuning in most cases. Our approach focuses on mobile applications' native resources, such as sensors and services. Because the majority of mobile applications nowadays need to make use of such native resources, our approach aims at facilitating one of the most common programming tasks undertaken by the modern mobile application developer.

### 1.2.2 Implementation

Our reference implementation of `native-2-native` makes use of the Eclipse IDE to provide a plugin for the Android development environment. The plugin captures and analyzes the token frequency in the Java code block that accesses a resource by means of some native API. Based on the captured code, the plugin forms a meta-data query to search popular Web-based programming resources for Swift code blocks accessing equivalent native APIs on the iOS platform. The highest ranked discovered Swift code blocks are provided to the developer who can further refine them with extra functionality, such as fault tolerance capabilities or strengthened security. Our evaluation shows that the automatically discovered Swift code for accessing subject native APIs ends up fit as is for the task at hand in more than 86% of test cases. These results indicate that the presented approach can become a pragmatic and powerful tool in the toolset of developers charged with the challenges of supporting mobile applications on multiple platforms.

## 1.3 Research Contributions

By presenting my work, this thesis makes the following contributions:

1. Three underlying insights that enable the use of web-based programming resources to automatically synthesize code blocks for cross-platform mobile development
2. A theoretical model and architectural design of `native-2-native` approach
3. A reference implementation of approach in the form of an Eclipse IDE plugin that is publicly available for experimentation and enhancement
4. An evaluation of the approach through the reference implementation that indicates a concrete practical benefit to the professional cross-platform mobile developer

## 1.4 Thesis Roadmap

The rest of this thesis is organized as follows. First, Chapter 2 gives a brief overview of major terms used throughout the thesis. Then, Chapter 3 provides a detailed background to this line of research and the various recent major developments in this field. Chapter 4 compares the presented approach with the related state of the art. Chapter 5 presents a running example that will be referenced throughout the thesis. Chapter 6 explains the underlying insights that enable our model and reference implementation. Then Chapter 7 details the approach overview with sections 7.1, 7.2, 7.3, and 7.4 going into detail about the core components. Chapter 8 details our evaluation while Chapter 9 discusses the strengths and limitations of our approach. Chapter 10 presents future work directions and 11 provides the concluding remarks.

# Chapter 2

# Terminology

- *native resources* → hardware sensors and software libraries customized to the host platform
- *document* → text/code file either mined from web-based resources or provided to the system as input
- *source document* → input Android Java code block
- *target document* → output iOS Swift code block
- *query* → extracted semantics from the input Android code block, used to search for the target document's constituent components
- *token* or *element* → any single document element at the level of individual strings
- *transpiler* → source-to-source translation from one language to another
- *meta-data* → data that describes other data
- *semantics* → core functionality of a code block

# Chapter 3

# Technical Background

This chapter presents a technical foundation for the research described in this thesis. The major sections include an overview of the current mobile development ecosystem, the various approaches, along with their corresponding shortcomings, in programming language translation, and the field of information retrieval. Although this chapter does not intend to describe every relevant technical idea used in this research, it does aim to ensure the reader can sufficiently comprehend the various concepts and technical arguments made throughout the remainder of the thesis.

## 3.1 Modern Mobile Software Development

The modern mobile device user enjoys the luxury of various phones, tablets, smart watches, smart glasses, and other devices. Many of these devices are developed by multiple companies each trying to outdo the other in terms of the offered functionality. This competitive nature leads to very similar products readily available on the mobile marketplace and a constant flow of new functionality driving users to remain engaged. There are currently three major platforms that produce the aforementioned devices on the market today. Figure 3.1 shows a high-level feature comparison between these three major platforms and demonstrates the massive amount of feature crossover. Many of these features were released in back-to-back version updates on one platform after being announced on one of the others. Figure 3.1 reinforces the fact that while native look-and-feel varies, and is shown below, the functionality delivered across platforms is largely similar.


Figure 3.1: Feature comparison for Android, iOS, and Windows Phone. [30] - D. I. Ginny Mies, Armando Rodriguez. Apple ios 6 vs. android vs. windows phone (comparison chart), `http://www.techhive.com/article/257363/apples_ios_6_vs_android_and_windows_phone.html` 2012. Used under fair use, 2015

The three major platforms—Android, iOS, and Windows Phone—are discussed in detail in the following subsections, including quick examples of native look-and-feel.

### 3.1.1 Android

The Android operating system is currently owned by Google and available open-source to the public for use and enhancement. Android is based on the Linux kernel [1] and allows for the development of mobile applications in Java with the Android Standard Library package. The incredibly widespread use of Java developers and familiarity with the language makes the Android platform the most widely used mobile development platform [62, 35]. The Google Play store, Android's mobile market for uploading and downloading Android applications, contains over 1 million published applications and more than 50 billion applications downloaded [34, 62].

Since the initial release of the first Android OS in 2008 [62, 35, 36], the platform has transferred owners, become open-source, and currently at release 5.1.1 "Lollipop" [34]. Some of the more notable mobile devices running an Android OS include a plethora of mobile phones, many tablets, Google Glass, and Android watches. The open-source nature of this platform facilitates the mobile application development more than other major platforms and helps keep Android as a dominant player on the market. A lock screen and app view of an Android Lollipop OS device is shown in Figure 3.2

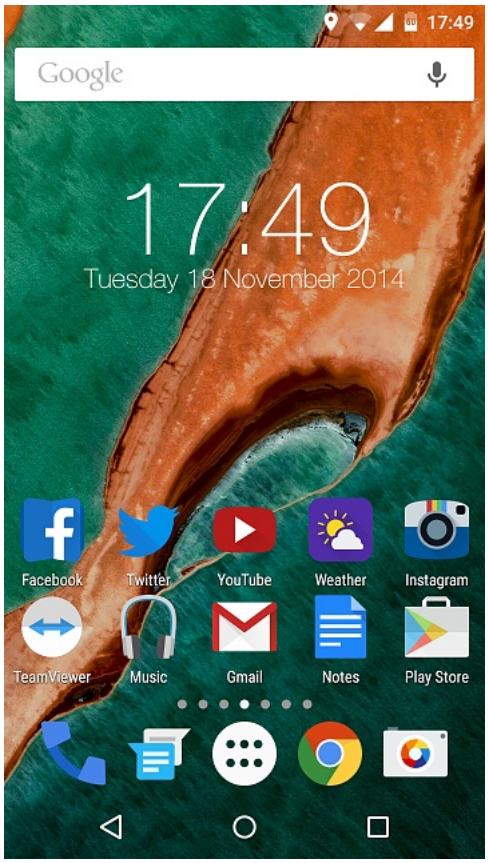


Figure 3.2: Android Lollipop OS device screen shot. [40] - harrygooz33. Share your nexus 5 screenshots! `http://forums.androidcentral.com/attachments/google-nexus-5/148907d1416334124t-share-your-nexus-5-screenshots-screenshot_2014-11-18-17-49-51.jpg,2014.` Used under fair use, 2015

### 3.1.2 iOS

The iOS operating system is a closed-source platform owned and developed by Apple. Originally launched in 2007 to support the first generation iPhone, Apple's first major smart phone, this platform has since been incorporated in many future iPhone generations, iPad, and Apple TV [44, 79]. This platform has a similar application store called Apple' App Store [48, 79], which houses close to 1.4 million applications with over a 100 billion downloads [48]. Even with a larger application base and higher number of downloads, Apple's iOS is a very close second to Android in market share of mobile applications and devices in use worldwide [79].

An iOS application, up until just over a year ago, was written in Objective-C, an Applie in-house designed programming language [28]. However, in June 2014, Apple launched a new

language to succeed Objective-C that is called Swift [3]. Swift is currently the dominant language for developing iOS applications for the Apple App Store. In Apple's latest press conference, an announcement was made saying Swift's coming releases would be open-source [3]. This marks the first time an Apple operating system in the mobile market will be developed by an open-source language and likely facilitating the growth of iOS applications on the market. A view of the iOS 7 OS home screen is shown in Figure 3.3

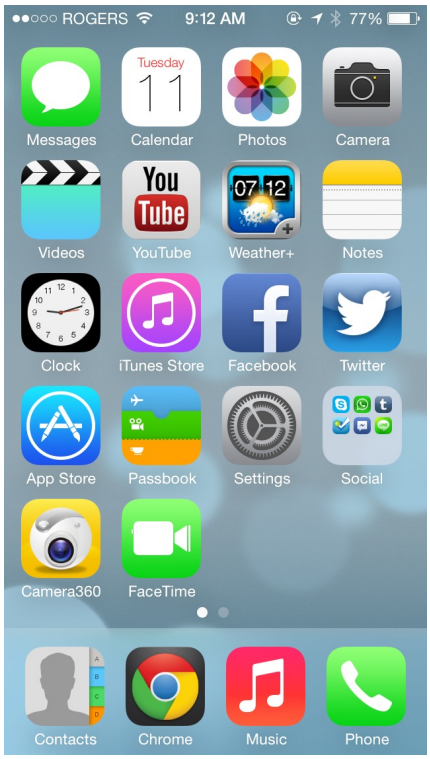


Figure 3.3: iOS 7 OS device screen shot. [109] - J. Yep. ios 7 (beta 1): A change worth the wait. `https://jordanstechstop.files.wordpress.com/2013/06/ios-7-home-screen.jpg?w=640`, 2013. Used under fair use, 2015

### 3.1.3 Windows Phone

Microsoft developed Windows Phone as a competitor for Android and iOS mobile operating systems and launched the first version in late 2010 [41]. Windows Phone has a very low percentage of market share. This can be attributed to Microsoft being a primarily desktop and tablet software provider and only recently attempting to gain entrance into mobile software operating systems. However, Microsoft has put forth substantial effort towards their latest Windows 8 "tile" style display as well as pushing cloud-based license ownership for software in their next release, Windows 10 [67]. This push is mirrored on their mobile OS as well. It is possible that with the coming seamless transition between desktop, tablet,

and mobile devices and corresponding Microsoft accounts, Windows Phone could incur a higher market share. The focus of this thesis is aimed at Android and iOS automated code synthesis and acknowledges Windows Phone as a potential future supported platform. Figure 3.4 shows the home screen of Windows 8 version of Windows Phone OS as a final demonstration of the various modern, major mobile platforms.

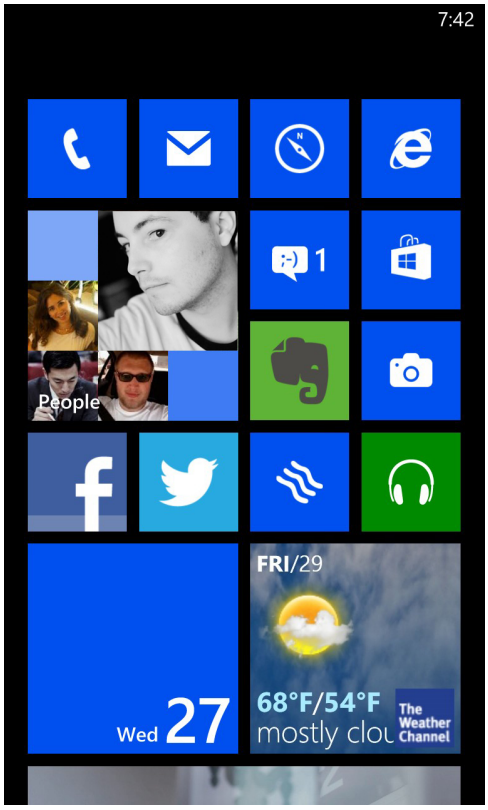


Figure 3.4: Windows Phone home screen on Windows 8 version. [37] - W. Gordon. Ask lh: Is windows phone ready to replace my iphone or android? `http://img.gawkerassets.com/img/18isacnob15c9jpg/original.jpg`, 2013. Used under fair use, 2015

### 3.1.4 Other Mobile Vendors

On a final note towards the various major mobile platforms, Research in Motion (RIM) and their primary device the BlackBerry [4], was a major player in the mobile market. However, they have fallen to newer platforms in the last decade and mainly hold only a small market share in company issued mobile devices. Many of these companies are switching over towards Android or iOS as well and BlackBerry's market share will likely continue to fall. Tables 3.1 and 3.2 display market share changes in units shipped. First Table 3.1 shows the total changes in the 2013-2014 years. It is clear that Android and iOS are squeezing out the other OS vendors and gaining ground for their platform. Similarly, Table 3.2 shows the same

| OS | 2014 UV | 2014 MS | 2013 UV | 2013 MS | % Year Change |
|---|---|---|---|---|---|
| Android | 1059.3 | 81.5% | 802.2 | 78.7% | 32.0% |
| iOS | 192.7 | 14.8% | 153.4 | 15.1% | 25.6% |
| Windows Phone | 34.9 | 2.7% | 33.5 | 3.3% | 4.2% |
| BlackBerry | 5.8 | 0.4% | 19.2 | 1.9% | -69.8% |
| Others | 7.7 | 0.6% | 2.3 | 0.2% | 234.8% |
| Total | 1300.4 | 100.0% | 1018.7 | 100.0% | 27.7% |

Table 3.1: Market Share (MS) for mobile OS in Unit Volume (UV) 2013-2014: [75] - M. S. R. Llamas, M. Chau. Android and ios squeeze the competition, swelling to 96.3smartphone operating system market for both 4q14 and cy14, according to idc,2015. Used under fair use, 2015

| OS | 2014-Q4 UV | 2014-Q4 MS | 2013-Q4 UV | 2013-Q4 MS | % Year Change |
|---|---|---|---|---|---|
| Android | 289.1 | 76.6% | 228.4 | 78.2% | 25.6% |
| iOS | 74.5 | 19.7% | 51.0 | 17.5% | 46.1% |
| Windows Phone | 10.7 | 28% | 8.8 | 3.0% | 21.6% |
| BlackBerry | 1.4 | 0.4% | 1.7 | 0.6% | -17.6% |
| Others | 1.8 | 0.5% | 2.3 | 0.8% | -21.7% |
| Total | 377.5 | 100.0% | 292.2 | 100.0% | 29.2% |

Table 3.2: Market Share (MS) for Q4 OS Unit Volume (UV) 2013-2014: [75] - M. S. R. Llamas, M. Chau. Android and ios squeeze the competition, swelling to 96.3smartphone operating system market for both 4q14 and cy14, according to idc, 2015. Used under fair use, 2015

information temporally localized to the fourth quarter sales in both 2013 and 2014. Again, we see that the other vendors are being forced out as Android and iOS are strengthening their position as the two titans of the mobile software development market.

As the main focus is Android and iOS, no further background or details will be provided about Windows Phone or BlackBerry.

### 3.1.5 Resource Sharing

In recent years there has been a large push towards effective resource sharing among similar devices [13, 19]. Some work even focuses on sharing resources across the major platforms [42]. The notion of resource sharing either via cloud offloading or nearby-resource sharing through Bluetooth/WiFi, presents an interesting component relevant to this thesis. With various sensors and services common across the major platforms, resource sharing is a viable and efficient method of utilizing similar functionality from other devices. This is made possible because of the availability of similar native resources across major platforms and

an ability to create middleware frameworks that can be heterogeneous [56, 24]. With API changes being referred to as a threat to Android success [61], the major mobile platforms have more inertia to stay course with similar resource naming conventions and core functionality.

## 3.2 Programming Language Translation

In this section we will discuss relevant background information in the field of programming language translation. While the notion of translating from one programming language to another is by no means novel, this thesis builds upon a native-2-native translation methodology. Figure 3.5 shows a high-level overview of the programming language translation process. All programming language translations discussed in the following sections can be simplified down to the three steps in Figure 3.5 In the following sections, we first build on the various types of language translation and then present a brief overview of native-2-native, explaining how it differs.


Figure 3.5: Simplified example of programming language translation

### 3.2.1 Traditional Compilation

The traditional compilation process is a specific instance of the more general set of programs called translators [70]. Most commonly, traditional compilation is used, in the modern setting, to translate high-level programming languages into assembly code which is then processed by an assembler to produce a binary executable. Traditional compiling developed as programmers at the dawn of computer science grew tired of programming in assembly and released the first ever high-level programming languages. The full history of programming languages is far from relevant to this thesis, however it is important to note that there is a considerably large set of programming languages in use today, each with its own syntax, architecture, and intended purpose.

### 3.2.2 Source-to-source Translation

With the evolution of programming languages, developers realized an automatic translation of one language to another could be very beneficial in many cases. There are a few chief reasons for needing to conduct a source-to-source translation (sometimes referred to as a transpilation [10]). Some tasks are simply easier in one language than another and it becomes more efficient to write code in one language and convert it to another. However, this process is far from perfect in many of the existing transpilers and is in many cases, a tool for producing a majority of the target languages' code but not to complete the process in its entirety.

While the details of some notable transpilers, like J2ObjC [49], are discussed in detail in the next chapter, it is important to mention another major advantage to automatically translating source from one language to another. Refactoring is a large sub-field of software engineering and involves the reorganizing of a code base to increase efficiency or readability [64, 69]. There is a strong research foundation to refactoring techniques via automated tools, but this is usually focused around improving one language and not to cross-platform improve from one language to another. Consider now a major version update for a language. Python's 2to3 tool refactors code operational in Python2 but deprecated in Python3 into the Python3 equivalent [26]. The 2to3 tool operates like a transpiler by converting between two languages even though it is a major version change and not a completely different programming language.

Source-to-source translation plays an important role in modern software engineering due to the various languages available and their corresponding applications. In this thesis, we tackle a different underlying issue in transpilation: accomplishing identical tasks on two different platforms, with two different programming languages. The underlying principles remain very similar to standard translations, but as Chapters 5 and 6 describe later in this thesis, the concept of transpilation falls short in capturing the native APIs and their functionality. However, these transpilers like J2ObjC [49] and Python 2to3 [26] prove incredibly valuable in their respective tasks.

### 3.2.3 Native Translation

The notion of native translation refers to extracting the underlying functionality of some code block, irrespective of the host platform and language, and expressing it in another language natively. Native translation represents a broad category of programming language translation that intends to capture the essence of a code block and make it portable to a fleet of relevant languages. The underlying goal is to ease developer effort at repeating identical functionality on multiple platforms. While the details of our approach are covered in Chapter 7 and relevant state of the art research is covered in 4, this section highlights the differences in traditional compilation, source-to-source translation, and native translation.

## 3.3 Information Retrieval

In this section we discuss the field of information retrieval and some of the canonical models developed to facilitate the process. Information retrieval, in the context of digital documents from web-based resources, refers to the gathering of data, in various forms, from an assortment of web-based resources and facilitating some key functionality [85, 84, 55]. The important functionality needed typically includes a standardization of the query document and all documents found into a meta-data object, efficient storage of results into a relational database or similar structure, and a metric for document similarity. The goal being that with a large collected document corpus that contains all documents in some easily search-able format that uses a clearly defined similarity measure. Figure 3.6 demonstrates the concept of information retrieval in the context of this thesis. Input query is used to search, either a database or some external web-based resource, and returns a populated list of document results.

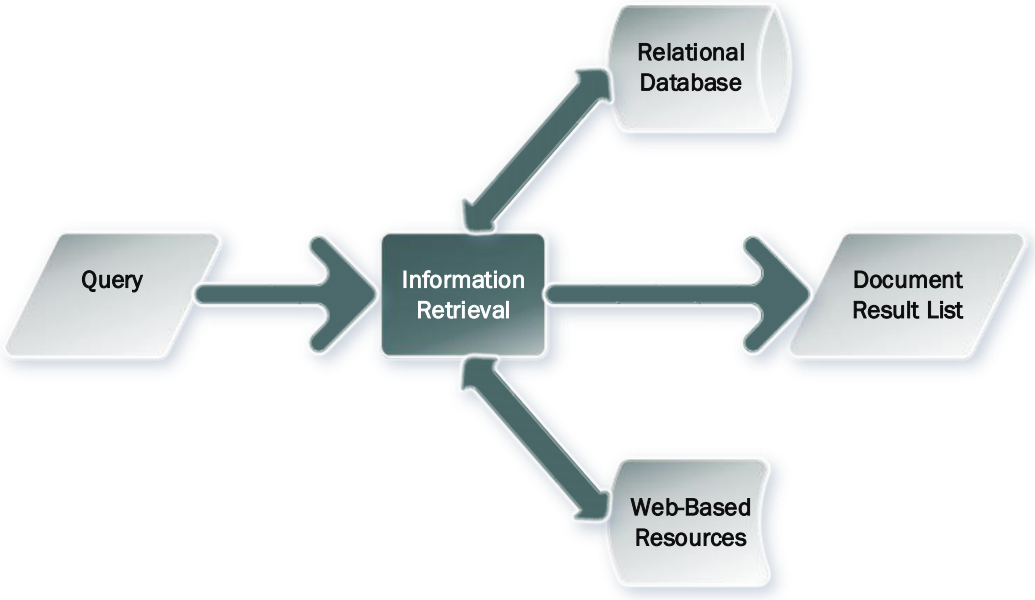


Figure 3.6: Simplified example of information retrieval process on input query

In the following sections, we discuss some key components in information retrieval as they relate to the primary thrust of this thesis and its future works. We first examine data mining and document similarity as the most relevant components of information retrieval incorporated in this thesis. Next, we discuss clustering and common applications to document similarity and information retrieval. Lastly, we comment on the nature of storing such

documents in a relational database.

### 3.3.1 Data Mining

Before any analysis can occur or any models run, the documents have to be acquired. The first step in the information retrieval component of this thesis involves using keywords to search or *mine* for the relevant documents. Traditional methods of searching for web resources include web crawlers and other scraping techniques [18]. However, typical web scraping only has to search a small finite amount of times and then the overarching analysis is performed extensively. In our approach, we instead must continually pull information as the user requests it, across potentially a large amount of simultaneous users. In effect, this prevented us from developing a single-minded web crawler and unleashing it to pursue needed documents. Instead, we needed to focus on available APIs for wrapped web scraping via authorization keys, effectively registering the application as *not* spam to the search engines.

The structure of the query is also crucial when dealing with data mining for text documents [98, 58]. Various state-of-the-art mechanisms for text mining exist [11, 108], and some can account of the specifics of programming resource-based documents for the ultimate goal of recommendation systems [54]. The mining of programming resources involves more extensive algorithms for intelligently filtering documents who's semantics are not inline with the query document [17]. Thus, some more advanced machine learning based approaches are well known and apply sophisticated filtration techniques while sifting through large document sets [105, 38].

### 3.3.2 Document Similarity

Document similarity is the underlying process by which documents in the overall corpus can be compared to each other. This is a core component of information retrieval as it determines what set of documents are returned for a given query document. There are many different approaches that can be used to compare documents, with most including two key components. First, there is an underlying model that is typically algebraic or probabilistic in nature, which operates on a tokenization of the documents [89]. Second, some sort of weighting scheme must be applied to the features of the model being used. Both components are equally important as a poorly chosen model for the given dataset will prove problematic during constructing the final ranking and a poorly chosen weighting scheme will render final results largely in-accurate. We focus our discussion on algebraic and probabilistic models as they are the most popular and relevant to this thesis work, however other approaches including set-theoretic, and feature-based models can often be implemented in this context [22, 59, 31].

### Algebraic Models

The first category are algebraic models that use an underlying vector or matrix structure to represent the documents. In this set of models, the resulting similarity is often stored as a $NxN$ matrix of pairwise comparisons, with each value being a scalar. Thus, given two documents $d_i, d_j$, the scalar value of the matrix's $[i, j]$ position would hold the pre-computed similarity index between those two documents.

In this thesis, we use an algebraic model called the vector space model [86, 76, 76], which, as the name suggests, holds a vector for each document. Every dimension in the vector represents a different feature. The size of the vector corresponds to all unique elements in the entire document corpus such that some value determined by the weighting algorithm is present or zero when the element is present or absent, respectively. The similarity is determined by the cosine between two two documents' respective vectors. This model will be discussed in more detail as it is used in Chapter 7, however similar algebraic approaches are commonly used depending on the dataset.

### Probabilistic Models

Probabilistic models operate on an underlying probability distribution that uses probabilistic inference to assign conditional probabilities to the similarities of documents. Typically Bayesian beliefs are used to reason out conditional probability tables which describe likelyhood of similarity [111]. While algebraic models store scalar similarity indexes derived from cosines of vectors or other properties, probabilistic models instead derive a measure of confidence in similarity between each document, represented as a conditional probability table. The largest probability from one document to all others would show the document most similar to this query.

Figure 3.7 demonstrates the different types of models along with their respective similarities and differences. Notice, a central distinction is with term inter-dependencies throughout the model's flow. The subcategory of set-theoretic models is very loosely defined but does contribute the *Fuzzy Set* model. The details of these approaches are omitted as they are not relevant to the core work of this thesis. Similarly, the numerous other algebraic and probabilistic models are left as omitted details. While they do detail interesting and varying views to the ranking of documents, the focus in this thesis will remain with the vector space model and no term-inter-dependencies.

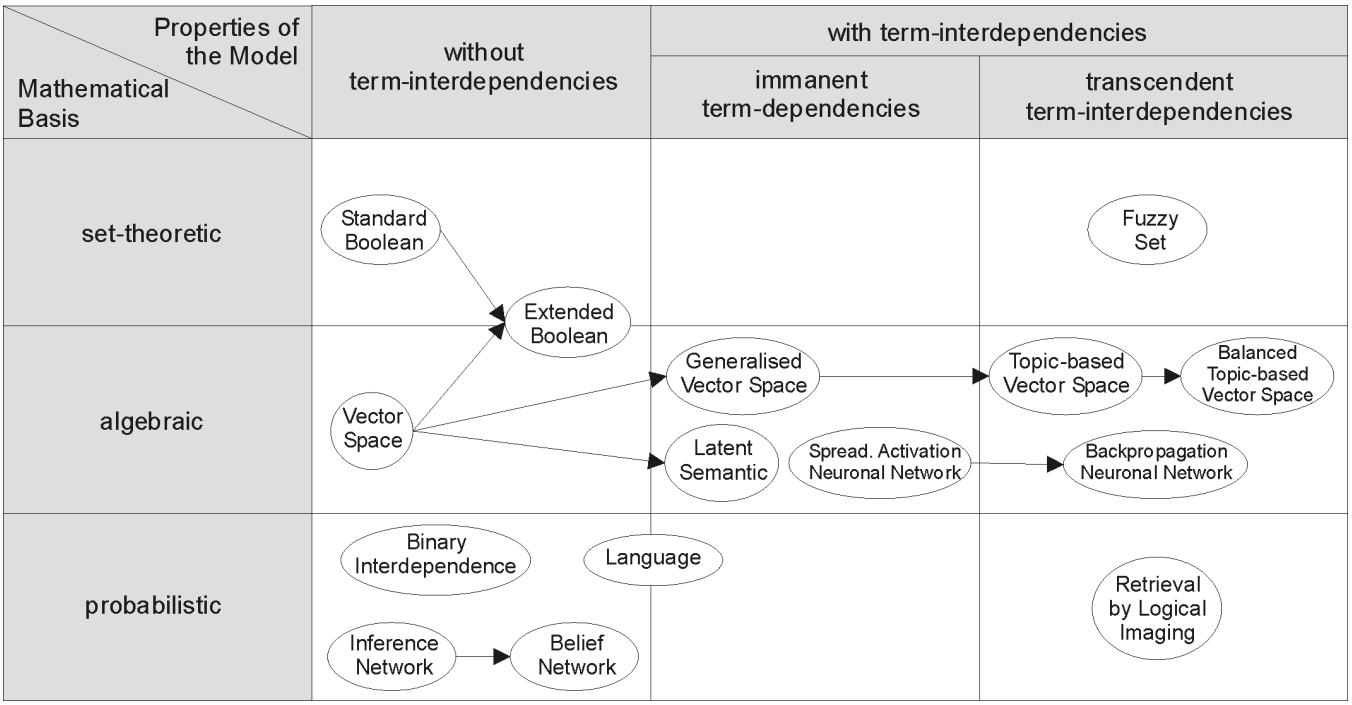


Figure 3.7: A comparison of the different types of information retrieval models. [23] - en.wikipedia. Information-retrieval-model, 2015. Used under fair use, 2015

#### Weighting Algorithms

The weighting algorithm determines how important various features are to the both a specific document and the entire document corpus. The approach taken to weight the features is important and very dataset specific. In this thesis, the underlying weighting algorithm used is term-frequency inverse-document-frequency (`tf-idf`) [84, 16, 20]. The most important aspect of `tf-idf` is that common elements across the entire document corpus are seen as less relevant because they are present everywhere. The "inverse" portion of this weighting scheme focuses on ensuring such common elements are partially marginalized out so as not to affect elements with smaller frequencies but rarer in the context of a query and target document [78]. Other metrics exist but are not relevant to the core algorithms the `native-2-native` approach uses and thus are not discussed here.

### 3.3.3 Hierarchical Clustering

Figure 3.8 displays a basic document clustering example. We see that two groups of documents are formed and separated into two clusters. The center of each cluster is some arbitrary document. Clustering in document retrieval expedites the process of searching

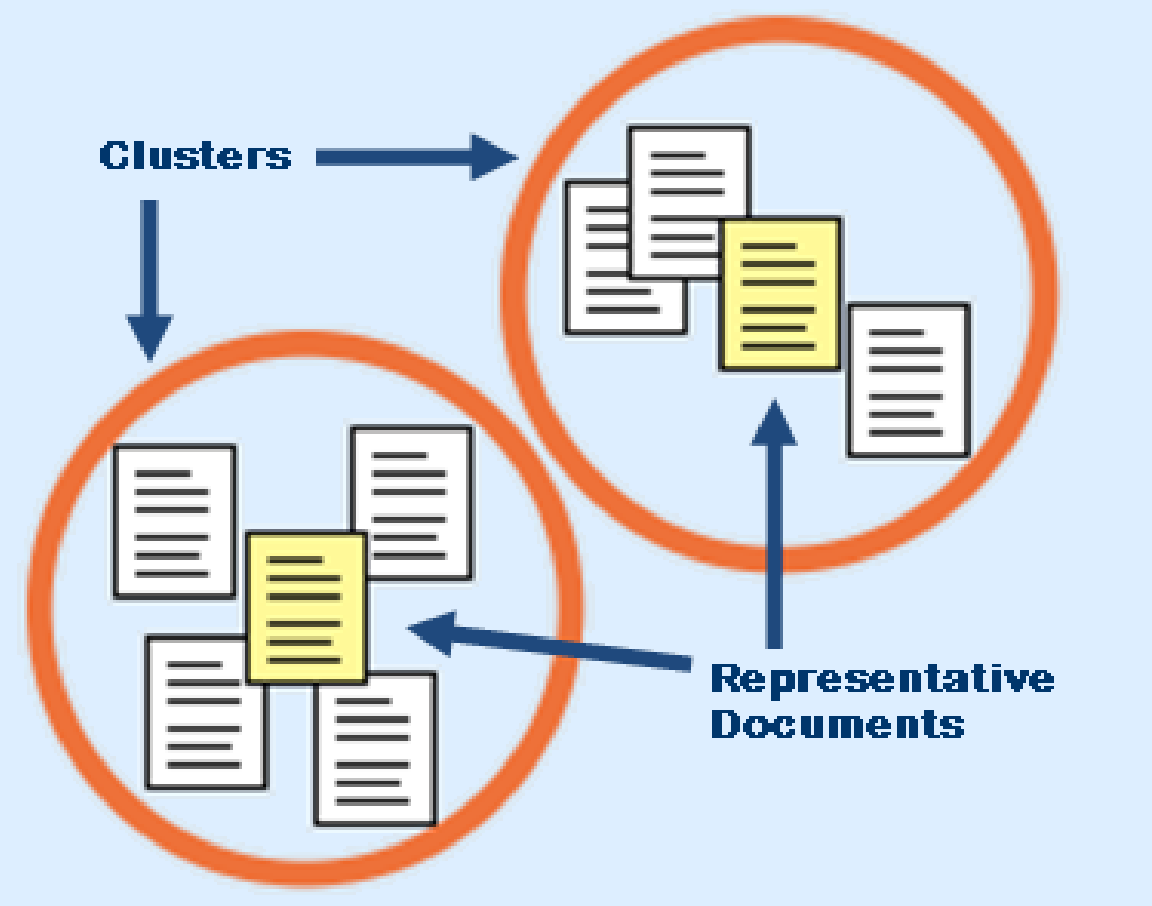


Figure 3.8: Document clustering example. [6] - D. Austin. ediscovery best practices: Cluster documents for more effective review, 2011. Used under fair use, 2015

by localizing the search to a single cluster or two rather than the entire document corpus [94]. Specifically, the interesting aspect of clustering for information retrieval is by creating a hierarchy for fast, chain like access [27]. In this thesis, clustering is mentioned only in the primary future work direction, however this notion is important to information retrieval. It is important to understand how the cluster centers are organized and updated.

There are two options for supplementing the document retrieval process with clustering. Either in an offline setting, we store all the documents previously accessed and build one time clusters for readily available access, or in an online setting, we add to the clusters at each new document located. In the latter case we must either add the new document to an existing cluster or, if it so vastly different from those currently available, we can start a new cluster. However, starting a new cluster usually involves merging two previous clusters in order to maintain a fixed number of clusters. The details of which approach is taken are omitted in this thesis but would be an important contribution to future work and extensions of this design [104, 47].

### 3.3.4 Document Storage in Relational Databases

In information retrieval, the data storage becomes increasingly more important as your model gets more complex. Typically, a relational database is used to store the meta-data of the object [110]. This way when a search is done, the objects meta-data is actually being compared in a standardized format to other documents. This facilitates the document similarity component and effectively makes the whole process possible. Of course, storing the documents as meta-data in a relational database is not as important when doing repeated web-based queries, as in this thesis. Furthermore, it is the meta-data representation that is actually the most important component. Nevertheless, without proper storage for larger datasets, this process becomes computationally difficult and in most cases, impractical for real world applications. Figure 3.9 displays the terminology and associations used in a relational database model.

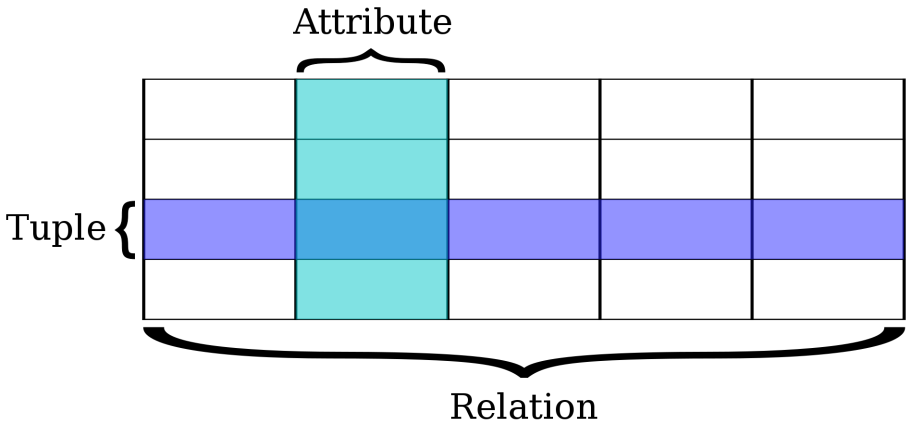


Figure 3.9: Relational Database Example. [12] - Booyabazooka. Relational database terms. Wikipedia, 2008. Used under fair use, 2015

# Chapter 4

# Related Work

`Native-2-native` is representative of a broad class of software engineering applications known as recommendation systems [80, 81]. Several examples of recommendation systems synthesize code snippets from web-based programming resources [73, 72, 71, 112] or build an intelligent code search engine [53]. We will discuss how our approach differs from or improves over these examples.

## 4.1 IDE Plugins

Prompter [73] is an Eclipse plug-in that given the current working code context automatically identifies relevant StackOverflow discussions. Its uniqueness is in providing a user-controlled confidence threshold to suggest only discussions that surpass this threshold. Compared to `native-2-native`, Prompter also makes use of the StackOverflow API, albeit as the only source for relevant discussion and code snippets. Our approach's larger search space [101], which includes Google Search, Google Code, and third-party resources in `native-2-native`), can achieve the level of precision required for cross-language and platform translation, a feature not supported by Prompter [73] or [112].

## 4.2 Static Caching of Web-Based Programming Resources

Some related approaches make use of statically cached programming resources to accelerate data retrieval. For example, the approaches presented in [73, 72] rank output code blocks by normalizing a sigmoid function of the average StackOverflow vote count from a June 2013 static data dump. Selene [97] recommends equivalent code blocks by searching a repository

of 2 million example programs to provide usage examples for a given input code block. Sourcerer [9] is another code search engine for a large-scale code repository (SourceForge). Strathcona [43], similarly to the systems above, also uses a repository based search corpus and provides the user with a structural overview of relevant code rather than actual code examples and discussions. A recommendation system developed by Bacchelli et al. [7] utilizes a vector space model with *tf-idf* as its frequency weighting model along with a singular query corpus source of StackOverflow. Similarly to other systems, Seahawk [71] also uses a static and publicly available dump of StackOverflow questions and lacks support for Swift.

As mentioned above, StackOverflow receives close to 6000 new questions a day. A static data snapshot from 2 years ago may be sufficient to mine for information about established language ecosystems and environments. However, the focus of `native-2-native` is mobile computing with rapidly evolving programming environments and language ecosystems. By combining vector space and linear models on live StackOverflow data, `native-2-native` returns suggestions that are more relevant, up-to-date, and better geared toward newer languages, such as Swift. Seahawk motivates the necessity of using the static dump to be able to search by means of the *Apache Solr* search system to achieve high performance efficiency. Although it would be unrealistic trying to match the performance efficiency of searching against a static local snapshot, we discovered that carefully calibrating weights and feature sets for the *tf-idf* analysis and vector space model not only provides complete and relevant results for both StackOverflow posts and other code sources, but also yields performance levels sufficient for practical use. Lastly, [53] focuses on providing useful documentation that supersedes standard API usage documentation. While an important aspect of the developer's ability to create mobile applications can at times include understanding necessary documentation, `native-2-native` focuses instead on the code synthesis and not just on supplemental documentation for the developer.

## 4.3 Machine-Based Language Translation

Before the notion of translating programming languages was explored, there was (and is) a large focus on automatic general-speaking language conversion. In many cases, the popular WMT'14 dataset is used for evaluating English to French programmatic conversions [66]. A popular technique used is to funnel raw input data, represented as text in English, into a Deep Neural Network in order to facilitate the translation process [96]. While the focus of this thesis is to automatically convert between programming languages, there is a fundamental token analysis which shares some similarity with standard machine-based language translation models used in [5, 8].

## 4.4 Transpiling Emerging Languages

The recentness of the Swift language's entry into the mobile computing space renders mainstream native transpilation (i.e., source-to-source compilation) systems inapplicable. For example, Google's J2ObjC [49] converts pure Java source code into Objective-C source code. Although a powerful and practical tool used by Google internally, J2ObjC lacks support for Android APIs and the controller component of the MVC design pattern, as well as converting to Objective-C rather than the new standard of Swift. By contrast, `Native-2-native` embraces the native mobile APIs such as Android and iOS. While it remains unclear whether rule-based compiler translation is even capable of bridging the differences between the platforms as architecturally dissimilar as Android and iOS, in the meantime `Native-2-native` provides a practical solution for deriving working Swift code analogous to its Android counterpart. Figure 4.1 shows sample usage of the J2ObjC plugin developed by Google and [88].

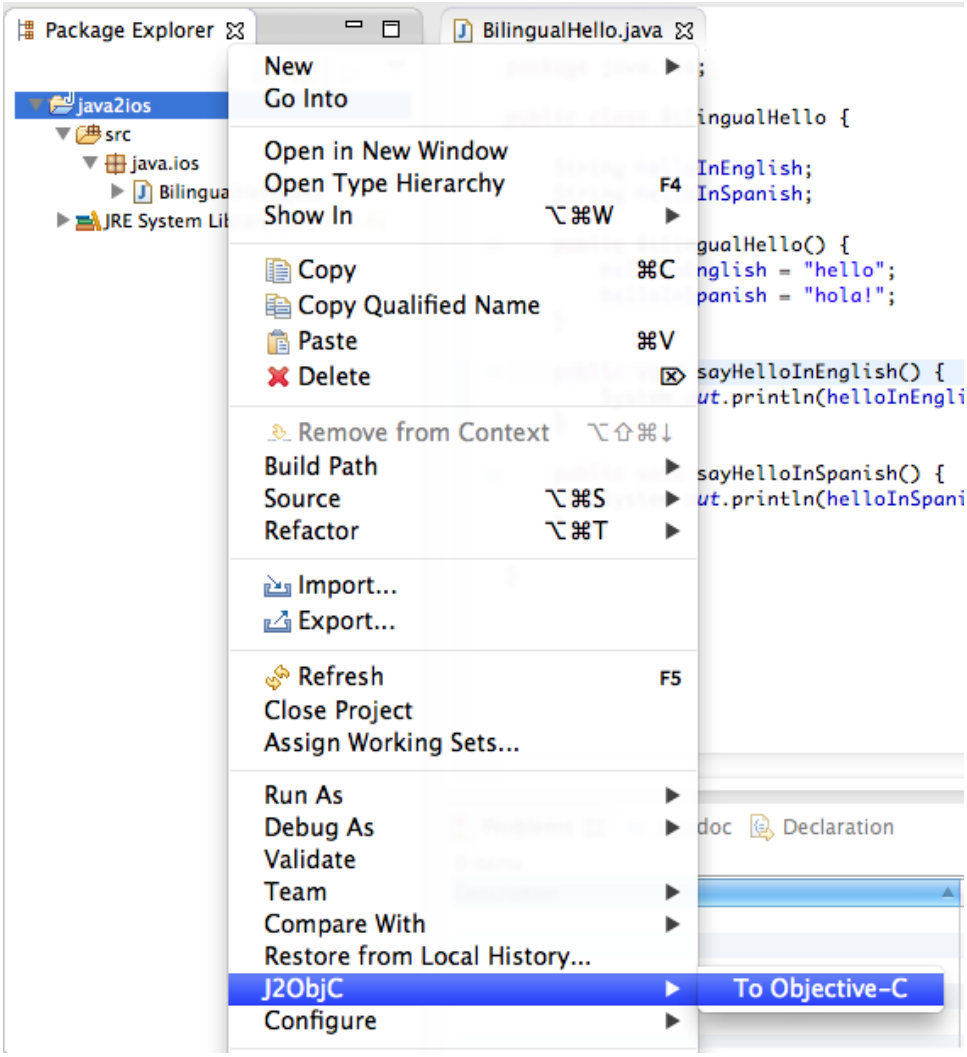


Figure 4.1: J2ObjC use case. [88] - H. Sapkota. j2objc-eclipse-plugin, 2015. Used under fair use, 2015

## 4.5 Incorporating Gleaned Insights

`Native-2-native` utilizes common themes and features across various prior recommendation systems, but it applies them to a problem that arises from the realities of modern mobile development—the necessity of supporting popular mobile applications on all major platforms, despite the inherent dissimilarities in the platforms' languages, APIs, and architectures. Potentially, this problem may be addressed by state-of-the-art cross-platform source-to-source translation, with some inroads already in place [74]. Nevertheless, precise source-to-source compilation capable of translating Java Android to Swift iOS functionality for native APIs remains a futuristic vision.

By using sources other than user-driven StackOverflow posts, `Native-2-native` mines a wider feature net without being restricted to API mappings [113]. This feature enables our approach to provide potentially more valuable and relevant search results based on the developer's code context. Given that the overriding goal of `native-2-native` is to aid the developer by suggesting and synthesizing the iOS/Swift equivalent of Android/Java code, any new suggested Swift code, as long as it correctly implements some iOS API, is likely to prove useful to the mobile developer.

Many of the related works discussed in this Chapter prove to be effective tools in their respective domains. However, it is by building upon their respective use cases that brings `Native-2-native` to a strong competitive stance. We combine the different web-based resources to utilize a fuller spectrum of potential target documents and avoid missing key data objects by not limiting ourselves to a static cache. Similarly to Prompter [73], we understand the primary delivery system as an Eclipse IDE plugin and thus provide an implementation open-source, allowing developers to use the plugin as is or enhance existing functionality.

# Chapter 5

# Running Example

Next we provide a concrete example that motivates the need for `native-2-native`. Consider a mobile application that determines if any of the user's friends are in the vicinity. Hence, the application is in essence a person proximity locator that continuously retrieves and processes GPS location information from the requesting device. Let us assume that the application is supported on both Android and iOS.

```
func startLocationUpdate() {
    locManager.requestWhenInUseAuthorization()
    locManager.startUpdatingLocation()
}

func locationManager(manager: CLLocationManager!,
  didUpdateLocations locations: [AnyObject]!) {
    var location = locations.last as! CLLocation
    var lat:Double = location.coordinate.latitude as
      Double
    var long:Double = location.coordinate.longitude
      as Double
    var result = NSString(format: "%.5f, %.5f",
      location.coordinate.latitude,
      location.coordinate.longitude)
    self.location = result as String;
}

func locationManager(manager: CLLocationManager!,
  didFailWithError error: NSError!) {

    DLog("Location Error: " + error.description);
}
```

Figure 5.2: iOS get Location basic functionality

```
public static LocationModel getLocation(Context context) {
    locationManager = (LocationManager)context.
      getSystemService(Context.LOCATION_SERVICE);
    Criteria criteria = new Criteria();
    String bestLocation = locationManager
      .getBestProvider(criteria, false);
    Location location = locationManager.
      getLastKnownLocation(bestLocation);
    LocationListener loc_listener = new LocationListener()
    {
        public void onLocationChanged(Location l) {}
        public void onProviderEnabled(String p) {}
        public void onProviderDisabled(String p) {}
        public void onStatusChanged(String p, int status,
          Bundle extras) {}
    };
    locationManager.requestLocationUpdates
      (bestLocation,0,0,loc_listener);
    location = locationManager.
      getLastKnownLocation(bestLocation);
    LocationModel loc = new LocationModel
      (location.getLatitude(),location.getLongitude());
    return loc;
}
```

Figure 5.1: Android get Location basic functionality

## 5.1 GPS Location

Figure 5.1 shows the code block that retrieves GPS location in Android; Figure 5.2 shows equivalent functionality in iOS. Even though both code blocks accomplish the same task, they are structured quite dissimilarly. In particular, the Android version creates a `locationManager` object from the passed context object, so that the initialized object can be queried for the GPS information. The iOS version creates a `CLLocationManager` object, whose initialized state contains the latitude and longitude variables. The code blocks on both platforms are relatively short, following similar coding idioms (i.e., creating an object to query its state) to retrieve the GPS information. Nevertheless, having written the code block in Android would not equip the programmer with the required knowledge to replicate this basic functionality in Swift.

Despite the popularity of source-to-source translators, they would be inapplicable if one wanted to automatically derive the iOS version of the code. The reason for the ineffectiveness of source-to-source translation in this instance is that native API access is always domain-specific, a complication that cannot be tamed with syntax-directed translation. By contrast, `native-2-native` extracts domain-specific semantics from the Android code block to be able

to search for equivalent Swift code. By automating the processes of extracting a code block's semantics and of discovering existing Swift examples with the closest equivalent functionality, the presented approach can alleviate some of the most tedious and error-prone programming tasks in modern mobile development.

## 5.2 Other Sensors

This running example will be referenced throughout the remainder of this thesis. Location via the GPS sensor serves as a central example of replicating functionality across major mobile platforms. However, many sensors are shared across Android and iOS. Table 5.1 shows the availability of various sensors across Android and iOS. Almost all are available on both, albeit by different names, and the few that are not will likely be adopted in coming releases.

| Sensor | Android | iOS |
|---|---|---|
| Proximity | Yes | Yes |
| Ambient Light | Yes | Yes |
| Accelerometer | Yes | Yes |
| Magnetometer | Yes | Yes |
| Gyroscopic | Yes | Yes |
| Camera | Yes | Yes |
| Temperature | Yes | Yes |
| Touch | Yes | Yes |
| Gravity | Yes | No |
| Air Pressure | Yes | No |
| Radio | Yes | Yes |

Table 5.1: Sensor availability across Android and iOS

# Chapter 6

# Enabling Insights

Heretofore, the discussion in this paper has focused on the *what* component of our approach—our end goal of automatically synthesizing native code for a target platform from equivalent code on a source platform, thereby enabling a cross-platform translation of natively implemented features. Before explicating the *how* component, which will provide our approach's algorithmic and implementation details, next we will focus on the *why* component, which will identify our approach's enabling insights.

The presented approach is enabled by a confluence of the following insights, derived from observing the realities of modern mobile software development: the peculiarities of the mobile software market, the working preferences of the modern mobile programmer, and the nature of platform-specific mobile APIs. We next describe each of these insights in turn.

## 6.1 Excessive Shared Core Functionality

Major mobile platforms have been competing with each other for market domination. Mobile hardware vendors have embraced the competitive mindset, which results in a so-called "arms race" when it comes to the devices' features and capabilities. As soon as one platform introduces a new hardware enhancement, the competitors feel compelled to introduce the equivalent or improved enhancement on their platform, lest they were to lose a major marketing advantage. Consider the GPS sensor, which provides the foundation for our running example. If this sensor and its corresponding capabilities were to be introduced to the Android platform first, the iOS platform would have no choice not only to mimic this feature, but also to add extra enhancements in the closest release feasible. Subsequently, Android would be compelled to match the latest iOS progress in this area without delay. This continuous competitive cycle, although driven exclusively by market forces, unveils the first enabling insight of our approach: major mobile platforms necessarily share an excessive amount of core native features.

Although their implementation languages and the corresponding native API's may differ vastly, their underlying feature sets from the end user's perspective enjoy a remarkable degree of similarity, which in turn leads to a high level of correlation in the vocabulary used to express the native APIs for these corresponding features. Assuming that API designers aim at creating intuitive-sounding and easy-to-understand names, reading in GPS information can be expressed in a finite, reasonably sized number of ways. It is also highly likely that the ways the GPS APIs are expressed on different mobile platforms will overlap in non-insignificant ways. Going back to the code blocks for this feature in Figures 5.1 and 5.2, one can see that the tokens *location* and *location manager* are both heavily used in both Java/Android and Swift/iOS. It is these shared tokens that make it possible to design a cross-platform translation mechanism for native APIs for shared features. Furthermore, the number of analogous features and their corresponding API shared tokens will continue increasing unless some platform definitively wins the competition for market share.

Consider the current notable features across the Android and iOS platforms. While not an exhaustive list by any standard, a short list for Android and iOS is shown below. First, the Android examples:

1. Google Drive
2. Google Hangouts
3. Google Now
4. Google Maps
5. In-App sharing through G+/third party apps

The first item on the list is a resource sharing application called Google Drive, that is Android's custom version of Dropbox. Next, we have a messaging application that integrates local phone contacts as well as Google's social network experiment, Google+. There is of course, Google's voice activated phone assistant: Google Now, and their in-house map applications. Lastly, we show their ability to share images and posts in existing apps through an integrated tool, posting them directly to Google+ or third party locations like Facebook. Now we show some major features on iOS, not surprisingly demonstrating overlap in features.

1. iCloud Drive
2. iMessage
3. Siri
4. Apple Maps
5. In-App sharing through third party apps

A virtually identical feature set is available on iOS devices currently. They contain similar naming conventions to the host platforms respective methodology with the exception of Google Now and Siri. However, in the scheme of total features on iOS, Siri is in the very small minority of applications that does not follow the typical naming convention. This is just a short list of notable major features discussed in [52, 32]. Many of the existing features overlap. Furthermore, as discussed above, it is unlikely for this trend exemplified here to change in the foreseeable future [100, 102]. Hence, we conclude that a methodology capable of leveraging such shared features and vocabulary conventions would have great potential benefit to become a practical and influential aid for modern mobile developers.

## 6.2 Reliance on Web-Based Programming Resources

Based on the growing number of online programming resources, the modern programmer increasingly relies on the Web to answer questions that come up during their day-to-day operations ranging from simple bug fixes to complex refactoring [57]. For example, StackOverflow [90] receives 2.65 new questions per minute and 4.41 new answers per minute on average, reaching close to 6,000 questions a day in peak sessions [92]. Indeed, StackOverflow and similar online question/answer discussion forums have become the modern programmer's primary medium of information exchange. One can attribute this strong shift toward online programming documentation to sheer demographics—StackOverflow reports that the average age of their user is 28.9 year old, based on surveying over 26K developers across 157 countries [91], which places them strongly within Generation Y, also known as the first digital generation, used to rely on the Web for all kinds of information. At any rate, it is indisputable that the Web has become an invaluable information sharing and acquisition platform for mobile developers. An additional draw of programming resources web sites is serving as reputation builders. Users of these websites commonly have the ability to rank the quality of provided information, with top providers earning high degrees of prestige and notoriety. These digital rankings can also be leveraged to automatically assess the quality of the available programming information.

In addition to average developers incorporating web-based programming resources into their daily routines, academia is slowly being pushed towards a similar mindset. There is notable research in adopting major blogs, podcasts, and various crowd-sourced discussion boards into the standard curriculum of programming courses in higher-education [83]. This argument stems from the increasing reliance on the Web for day-to-day programming questions and learning concepts. The next step seems to be to provide formal training in the form of unit teaching, to incorporate mainstream social discussion boards and successful blogs into collegiate introductory courses [82]. In general, it is this notion of being able to teach "anyone anything anywhere anytime" [46] that fuels web-based resources to grow and mature.

## 6.3 Accidental Complexity

Finally, we argue that figuring out how to express a native API on some platform, having just expressed an equivalent API on another platform, constitutes *accidental complexity*, and as such is a promising candidate for automated treatment via software engineering innovation [15]. Consider the process by which some native API becomes used in a typical mobile program. A developer decides that some feature needs to be added to a program, and that feature will make use of some native resource. Designing the feature is *essentially* complex, while discovering which API one must invoke is *accidentally* complex. Furthermore, this discovery process remains accidentally complex, irrespective of the implementation platform. Removing accidental complexity is a key driving force behind reducing the programmer workload. There is another peculiarity that arises when translating native APIs between Java and Swift. While Java remains the most commonly used programming language [91, 107], Swift according to StackOverflow is now regarded as "most loved" language. Hence, one can expect an abundance of web-based programming resources for both languages.

## 6.4 Summary of Insights

By leveraging these three main insights, we were able to create a simple but powerful approach to automatically translating native APIs between Java/Android and Swift/iOS. First, noticing the realities of modern mobile software development provides a foundation from which a system can be designed. Next, realizing the paradigm shift towards web-based programming resources provides direction in how the proposed system could function. Lastly, understanding the differences in *accidental complexity* and *essential complexity* as discussed in [15], supports our claims of candidacy for automation. It is only after combining these three core beliefs can we begin to discuss the details of our approach.

In the next Chapter, we provide the algorithmic and implementation components of our approach. In Chapter 8, we report on the results of applying our approach to real-world examples of using native APIs.

# Chapter 7

# The Native-2-Native Approach

Figure 7.1 shows a high-level overview of our approach. A mobile application developer first implements a new feature using some native API of the Android platform in Java. Once the developer deems the feature's implementation completed, the feature's code block is passed as input into the `Native-2-Native` IDE plugin. The plugin performs the following tasks in sequence: generalize the Java code block to form a search query, execute the query against popular web-based programming resources, summarize and rank the results, and finally present a synthesized Swift code block for the iOS platform back to the developer. The presented code block is typically partially complete, and implements the same feature as the input Java code, but by means of the equivalent native Swift/iOS API. It is the ability to automatically discover a code block that natively implements a feature in Swift/iOS for an equivalent code block implemented natively in Java/Android that gives our approach the name of `native-2-native`.

In the following sections, we detail the constituent parts of the native-2-native approach. Section 7.1 describes the process of extracting semantics from the input Android/Java code block. This process includes multiple techniques in tokenization, filtration, and frequency analysis to generate an important set of query keywords. Next, section 7.2 explains how the approach ascertains a similarity index between the initial Java code block and all mined Swift code blocks. The meta-data objects serve as the central representation of platform and language-independent semantics for all mined documents as well as the input Java source code. Finally, section 7.3 delves into the underlying process by which the approach is able to produce an equivalent Swift code block. This process makes use of two algorithms: (1) a searching algorithm that discovers the relevant resulting set of documents, and (2) a ranking algorithm that operates on a set of mined documents to produce a platform- and language-independent semantic ranking, which synthesizes the output Swift code block.

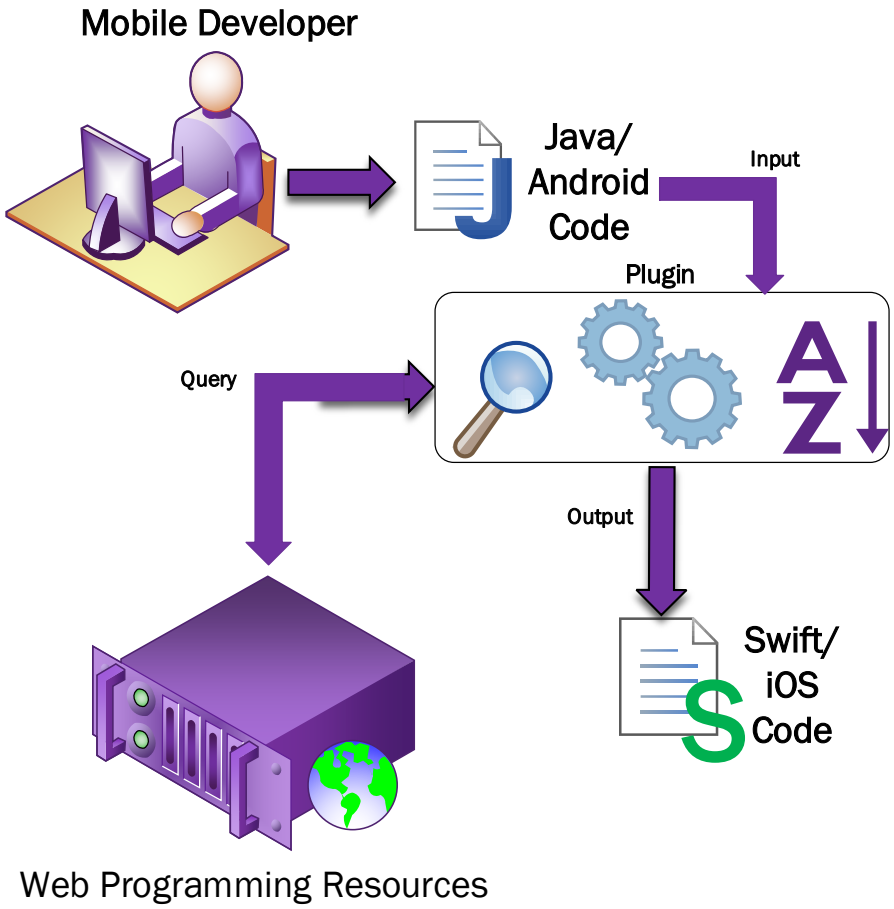


Figure 7.1: Native-2-Native: High-level Approach Overview

## 7.1 Extracting Core Functionality from Java/Android Code

This section describes the process of translating input Android Java code blocks into web queries. This process extracts the semantics of the code block by means of tokenization, filtering, and frequency analysis, described in turn next.

### 7.1.1 Tokenizing and Filtering

The flowchart in Figure 7.2 details the process. The Java source input is lexically tokenized, with the superfluous tokens filtered out. The tokenizing makes use of the canonical bag-

of-words model [50]. This model also separates camel case variables, title case variables, method declarations/invocations, and also removes non-alphanumeric characters attached to strings.

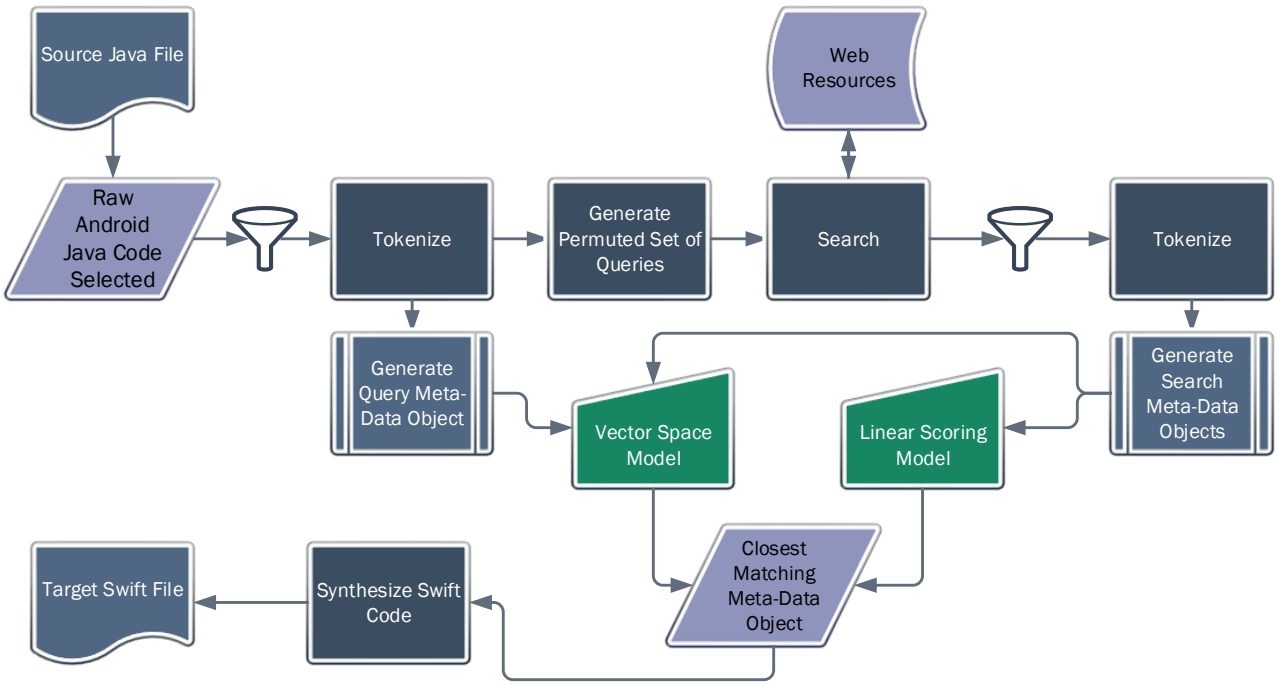


Figure 7.2: Full approach program flow

The resulting tokens are then first filtered by applying stemming in the form of suffix stripping to eliminate any verb-tense discrepancies that could arise while searching web resources for the target document's constituent components. Then tokens that happen to be substrings of other tokens are filtered as well. For example, tokens 'location' and 'loc' are assumed to possess related semantic intent. Tokens that would weaken the precision of the frequency analysis are filtered out as well (e.g., comment designators, stop words, etc.)

### 7.1.2 Frequency Analysis to Generate Web Queries

The next step calculates the document frequency for each extracted token that has not been filtered. The frequency analysis produces a sorted list of the most used tokens in the input. The $k$ most used tokens become the query keywords for the search routine in Figure 7.4. The default value of $k$ is 3 but can be customized at will. The query keywords generated for the GPS location used throughout this thesis are shown below as an example.

| 1st Highest Frequency | 2nd Highest Frequency | 3rd Highest Frequency |
|---|---|---|
| "`location`" | "`get`" | "`user`" |

Table 7.1: GPS location query keywords extracted from input Java source code

## 7.2 Meta-Data Objects

Meta-data, data that describes other data, has been used to facilitate the search and discovery of related data objects [14, 87]. The presented approach uses meta-data to describe source and target documents as a means of streamlining similarity comparison. The source document's meta-date object is composed upon the completion of the tokenizing and filtering steps as described above. The target documents go through a parsing process, after they are found via the searching algorithm demonstrated in Section 7.3, that is shown at a high-level in Figure 7.3. All target documents are converted into their meta-data object equivalents and then the ranking, also described in 7.3, occurs.

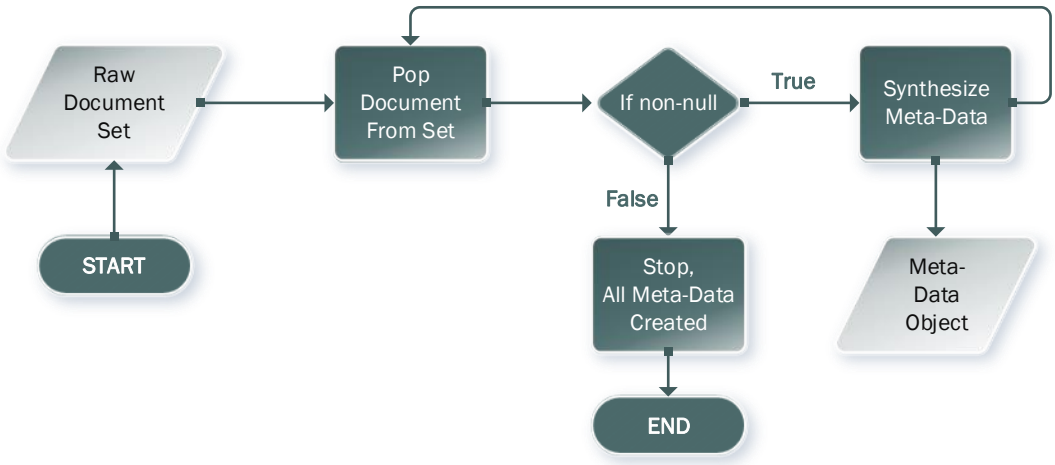


Figure 7.3: Target document's meta-data construction process

### 7.2.1 Meta-Data Fields

Meta-data fields serve as the abstraction that captures all pertinent information from web-based programming resources and the source document in an easily search-able and comparable format. The meta-data objects use their fields to store features mined from all the searched web resources as well as those extracted from the source document. A `generate` routine processes every search result to populate the fields defined by a given meta-data

object. Representing the search results via meta-data objects makes them easily amenable to similarity evaluation with various ranking models.

Some features are deliberately supplemented to indicate the functionality to search for. For example, a preferred StackOverflow response would be a so-called "accepted answer," a code snippet check-marked by its originator as having solved the posed question.[1] In contrast, the source document lacks this property, as it simply represents the input Android code block. To prevent supplemented features from unduly skewing the final ranking, the source document's missing features are defaulted in the meta-data object as their ideal functionality. Similarly, web search results returned from random programming resources will also contain missing features, as well their own supplemental features, to be steered toward the searched for functionality. The meta-data objects' fields are the union of all features across all mined web resources and the source document.

## 7.3 Searching and Ranking

In this section, we first present a searching algorithm that queries web-based programming resources, in permuted orders, for relevant data pertaining to the query keywords. Then, we present a ranking algorithm that incorporates two different models for determining similarity between the source document and all mined potential target documents.

It is worth pointing out that both the searching and ranking algorithms disregard the control flow constructs present in the source document, operating solely on the extracted keywords described in the previous section. This design decision renders our approach largely independent of the specifics of the business logic of the native code blocks at hand, focusing exclusively on the native API used. Nevertheless, the disregarded control flow constructs are fully restored during the final code synthesis phase.

### 7.3.1 Searching Algorithm

Figure 7.4 shows the searching algorithm for mining the relevant data. The algorithm's input is the generated query keywords discussed above and the output is a full list of the resulting searches stored in a custom-made, answer-wrapper object. Before the core of the searching algorithm is executed, the `initKeywords` subroutine first initializes the full query keyword set. Although the input is just the set of query keywords, various permutations and subsets of the original query keywords must be searched to locate all relevant results. The `initKeywords` subroutine in Figure 7.4 explains how the full set of keywords involves three components: 1) transcribe the original keywords as individual queries, 2) identify the first and second (by frequency-based importance) keywords, in both orders, as subset queries,

[1] The StackOverflow website uses green check-marks to denote accepted answers.

and 3) permute the complete set of the original keywords. Although some flexibility in the number of query keywords is allowed for user fine-tuning, the hard limit of 5 separate keywords for the permutation component ensures that the input is computable in practical space and time boundaries.

```
/* INPUT: query keywords to search */
/* OUTPUT: full list of resulting searches */
DEF Search(keywords)
    resultList ← ∅
    keywordSet ← initKeywords(keywords)
    FOREACH keyword ∈ ∀keywordSet DO
        queryResultsSOF = execStackOverflow(keyword)
        FOREACH result ∈ ∀queryResultsSOF DO
            Ans ← result, rank
            resultList ← resultList∪ {Ans}
        END FOREACH

        queryResultsGoogle = execGoogle(keyword)
        FOREACH result ∈ ∀queryResultsGoogle DO
            temp ← execStackOverflow(result)
            Ans ← temp, rank
            resultList ← resultList∪ {Ans}
        END FOREACH

        queryResultsElse = execElse(keyword)
        FOREACH result ∈ ∀queryResultsElse DO
            Ans ← result, rank
            resultList ← resultList∪ {Ans}
        END FOREACH
    END FOREACH
    RETURN resultList
END Search

DEF initKeywords(keywords)
    fullSet ← keywords
    fullSet ← fullSet ∪ {key[0] + key[1]}
    fullSet ← fullSet ∪ {key[1] + key[0]}
    fullSet ← fullSet ∪ perm(keywords)
    RETURN fullSet
END initKeywords
```

Figure 7.4: Search Algorithm Pseudo code

Although the algorithm can be configured to search any web-based programming resources, we next describe it by means of three representative categories: (1) a live API call to StackOverflow with the given query keywords, (2) a Google search on the current query keywords that specifically limits to StackOverflow websites only to catch discrepancies in a StackOverflow search internally as well as an external Google search on the same keywords; the

Google search results are funneled to StackOverflow to directly data-dump that post.[2] (3) a Google search that excludes StackOverflow posts to include less popular but potentially also relevant online blogs and other web resources to the list of relevant search results. All three methods' results are standardized into the answer-wrapper object and stored for later ranking and analysis. This entire process is conducted for each of the permuted list of query keywords to generate the final list of web-based search results.

### 7.3.2 Ranking Algorithm

The ranking algorithm determines the degree of similarity between the source document and all potential target documents to present the most relevant Swift code blocks to the user. Figure 7.5 details the algorithm. The input is the list of search results generated by the searching algorithm, as well as the original source document (also in the answer-wrapper format), and a $k$ value for the number of results to be returned. The output is the $k$-top results of the ranking. The algorithm's main components produce the vector space model and linear model with their respective feature sets. The standardization into answer-wrapper objects described above facilitates the retrieval of this information for each document in the results list. The standard format for the answer-wrapper objects is used to generate the target set of meta-data objects that produce their respective vector space and linear model scores. Next, the models are combined, sorted, and the top $k$ of those are returned.

Figure 7.5 also includes the subroutine for calculating the `cosine` of the angle between the current potential target document and the source document, as required by the central logic of the vector space model [7]. The cosine determines similarity from a constructed $n$-dimensional sphere, containing all the $n$-dimensional feature vectors. Each vector's dimension is calculated using a term-frequency inverse-document-frequency (`tf-idf`) weighting function, which accounts for a term's frequency without over fitting by accounting for common terms across the document corpus, where $n$ is the number of total unique tokens available in the current document corpus. Each value in the vector is represented by the modified indicator function seen in equation 7.1 that incorporates the `tf-idf` weights. Here $d \in D$ represents some document in the full $k$ set of documents, including all potential targets and the source, with weight $w$.

$$v_{d_k}(i) = \begin{cases} 0 & \text{if token } i \notin d_k \\ w_i & \text{if token } i \in d_k \end{cases} \tag{7.1}$$

The cosine value is calculated using the underlying vector space model shown in Equation 7.2 [87]. The norm of each vector is the square root of the summation of each dimension's squared value. Given that the dot product of two vectors is a scalar and this quantity is

[2]Searching through Google and StackOverflow frequently yields dissimilar complementary results in differing orders.

```
/* INPUT: source/targets MetaData, k top results */
/* OUTPUT: k closest ranked MetaDatas */
DEF Rank(Source, TargetMDs, k)
    ranking, topK ← ∅
    FOREACH target ∈ ∀TargetMDs DO
        C ← getCos(target, sourcelate)
        L ← getLin(target)
        temp ← bag(C, L)
        ranking ← ranking∪ (temp, target)
    END FOREACH
    sort(ranking)
    FOR i IN k DO
        topK ← topK ∪ ranking(i)
    END FOR
    RETURN topK
END Search

DEF getSimilarityUsingCos(t, s)
    N_t, N_s, cos ← ∅
    FOREACH d ∈ ∀t DO
        N_t ← N_t, d^2
    END FOREACH
    FOREACH d ∈ ∀s DO
        N_s ← N_s, d^2
    END FOREACH
    N_t ← sqrt(N_t)
    N_s ← sqrt(N_s)
    cos ← (t · s) / (N_t · N_s)
    RETURN cos
END getCos
```

Figure 7.5: Rank Algorithm Pseudo code

divided by the product of two norms, the resulting value is also a scalar. The variables $s$ and $t$ are used to denote the source and target documents, respectively.

$$cos_s(t) = \frac{s \cdot t}{||s||_2 \quad ||t||_2} \tag{7.2}$$

Note that the linear model subroutine is not shown in figure 7.5, as it is simply the linear combination of all relevant features in each meta-data object of the results documents shown in equation 7.3. The model's weights are derived from a combination of tf-idf values and normalization by average feature quantities available from the StackOverflow API.

$$\text{lin}(t) = \sum_{i=1}^{n} (w_i * t_i) \tag{7.3}$$

### 7.3.3 Complexity Analysis

In this section, we briefly comment on the computational complexity incurred by the more exhaustive procedures of the `native-2-native` approach. The ranking algorithm and following model combination process is the most computationally intensive component, on which we hence concentrate our analysis. Recall Equations 7.1 and 7.2, presented in the previous section, that outline the vector space model approach, as well as Equation 7.3 that explains the linear model used. Equation 7.4 shows the complete model combination, along with the weight calculation, as a function on all potential target documents. Note the parameters $\alpha$ and $\beta$ that are constrained such that $\alpha + \beta = 1,\ \alpha \geq 0,\ \beta \geq 0$ and represent the respective magnitude of the two models to each other.

$$\max_{t \in T} \left\{ \frac{\alpha(s \cdot t_j)}{\sqrt{\sum_{i=1}^{n} s_i^2 \cdot \sum_{i=1}^{n} t_{i,j}^2}} + \frac{\beta \sum_{i=1}^{k} w_{i,j} f_{i,j}}{\max_{a \in A} \left( \sum_{i=1}^{k} w_{i,a} f_{i,a}) \right)} \right\} \tag{7.4}$$

This equation is derived by combining the vector space model with a normalized linear model and using the weighting function shown in Equation 7.5. In Equation 7.5, note the $I(t)$ indicator function that is 0 if the current term is not present in the current document and 1 otherwise. Also of importance to this equation is $T$ which represents the total number of target documents mined by the searching algorithm.

$$s, \forall t_j \in T : t_f \cdot \log \left( \frac{|T| + 1}{I(t)} \right) \tag{7.5}$$

Given that $n_i$ is the total number of tokens in some document $i$, we bound the maximum number of tokens on the overall approach as $dim = \{\sum_{i=1}^{|T|+1} \{n_i\}\}$. This bound being placed in set notation to represent the elimination of duplicate tokens across not just a single document but all documents, and accounting for the full set $T$ of target documents in addition to the single source document. Lastly, $d$ is the upper bound on the feature vector used for the vector space model as we take all unique tokens. In essence, we have $|<d_{1,i}, d_{2,i}, ..., d_{n,i}>| \leq dim$, where the vector represents some document $d_i$'s $n$ features; that will be equal across all targets and the single source document. Hence, the Big O of the approach's primary component in terms of documents is $\mathcal{O}(T + 1)$, which runs in linear time, $\mathcal{O}(n)$. However, accounting only for documents fails to accurately reflect the true computational complexity, as for certain operations one must measure their more-numerous token operations to evaluate their complexity. Thus, the complexity measured in tokens is $\mathcal{O}(dim^2 + dim * d_i + dim * f + f)$,

where $f$ is the number of linear features, a number strictly smaller than $dim$, and where $d_i$ is the current document's token count, also strictly smaller than $dim$ as shown above. If $d_i \leq dim$, then $d_i * dim \leq dim^2$. Finally, we can simplify the overall complexity in terms of tokens as $\mathcal{O}(dim^2 + dim^2 + dim^2 + f) = \mathcal{O}(3dim^2) = \mathcal{O}(n^2)$.

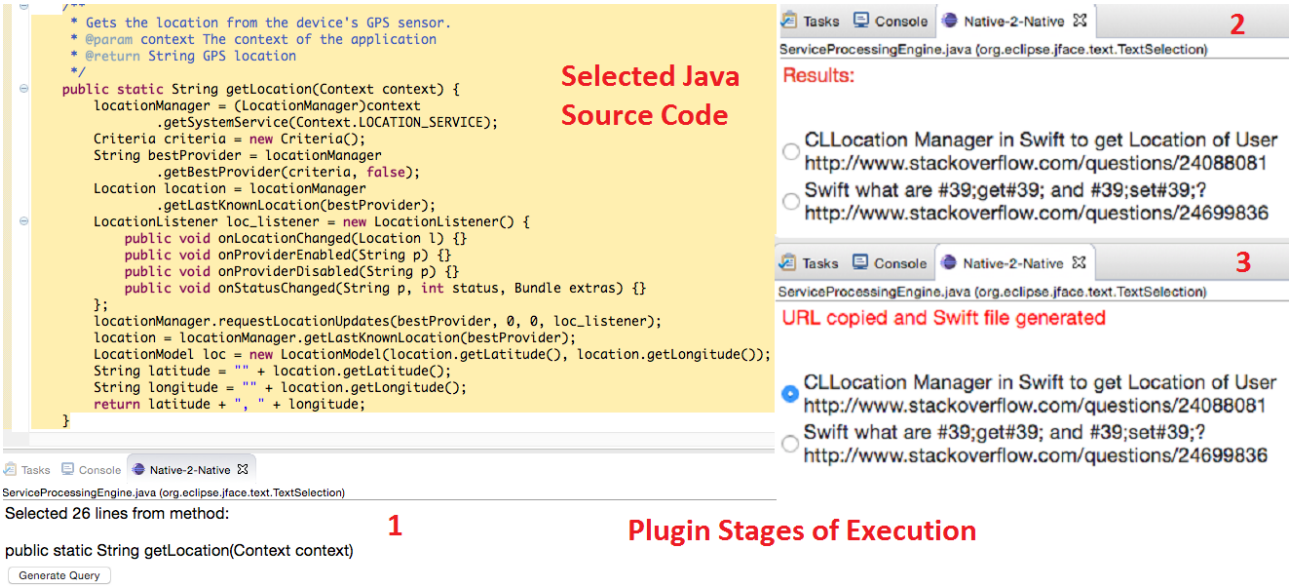


Figure 7.6: Plugin showing user selecting GPS Location code and subsequent results returned

## 7.4 Programming Interface and Code Synthesis

The approach is concretely realized as an Eclipse IDE plugin publicly available and open-sourced for future improvements and enhancements and located on a publically available github: `https://github.com/antuanb/Native-2-Native`. Figure 7.6 (left half) displays the initial plugin view from the end mobile developer's perspective. The developer first highlights a code block to be rendered in Swift and then clicks the generate button. Figure 7.6 (right half) highlights how the developer selects which of the top two presented results they desire to be presented as a Swift source file. Note the URL of the corresponding result is copied to the clipboard, so that the developer can refer to the originating web-resource for further information. The generated Swift source file is saved in the current working directory of the Android/Java project.

The left side of Figure 7.7 shows a sample generated Swift file in our running example of GPS location. The control flow of the Android/Java file is replicated to create a skeleton Swift source file first. Then, the developer-selected result is incorporated into the Swift file

to finalize the synthesized code block. In this example, the synthesized Swift file has the correct iOS/Swift protocol for instantiating and utilizing the `CLLocationManager` native API to access a user's GPS location along with a logic flow outline. The remaining fine-tuning left for the developer is to format and return the contained location information in the style desired. The next section focuses on the evaluation of the presented approach, detailing the native APIs used, the evaluation process, and the precision levels obtained.

```
//N2N-getLocation.swift
/**
* Gets the location from the device's GPS sensor.
* @param context The context of the application
* @return String GPS location
*/
func getLocation(/*unknown token*/) -> String {
    //Insert Native-2-Native result
    var latitude:String = "" /*unknownToken*/
    var longitude:String = "" /*unknownToken*/
    return latitude + ", " + longitude
}

/* NATIVE-2-NATIVE RESULT
override func viewDidLoad() {
    super.viewDidLoad()
    // Do any additional setup after loading the view,
       typically from a nib.
    locationManager = CLLocationManager()
    locationManager.delegate = self
    locationManager.desiredAccuracy = kCLLocationAccuracyBest
    locationManager.requestAlwaysAuthorization()
    locationManager.startUpdatingLocation()
}
*/
```

Figure 7.7: Synthesized Swift code snippet for GPS Location (left) and HashMap (right)

# Chapter 8

# Evaluation

In this section, we present the results of evaluating our approach. To that end, we applied our approach to various Android/Java native APIs to automatically synthesize analogous functionality in iOS/Swift, and then examined the synthesized code blocks for their fitness in expressing the functionality at hand. We evaluated the Native APIs, including sensors (e.g., GPS, accelerometer, etc.), network interfaces (e.g., WiFi, Bluetooth Low Energy (BTLE), etc.), and canonical library classes/data structures (e.g., `String`, `ArrayList`, `HashMap`, etc.).

## 8.1 Analysis Categories

We evaluated all the synthesized functionality by hand, which included compiling the code with the Swift compiler and testing its runtime behavior. Future work will investigate whether this tedious evaluation process can be automated. The evaluation placed each automatically synthesized code block into one of the following four categories: (1) the synthesized code block appears to be correct both in form and functionality (i.e., it can be used carry out equivalent native API functionality when integrated with an iOS application); (2) the synthesized code block is incorrect, either in form or in functionality (i.e., the code cannot be compiled without major modification or its runtime functionality fails to deliver the expected equivalent native API functionality); (3) the synthesized code was partially correct, requiring a reasonable amount of programming effort to carry out the expected functionality. We define "reasonable programming effort" if the synthesized code block can be used as a helpful reference point that would speed up the search for the right functionality rather than hinder progress. Notice that our definition is necessarily subjective, thus creating an internal threat to validity. (Fortunately, our evaluation did not yield many results as belonging to this category.); (4) the result is not a code block, thus providing negligible utility to the programmer.

## 8.2 Table Explanations

Tables 8.1, 8.2, and 8.3 display the results of the aforementioned evaluation. The four evaluation categories for the synthesized code blocks are designated as `YES`, `NO`, `1/2`, and `NaCB` (not a code block), respectively. The `TOTAL` column reports on the total number of test cases for each native API type. Recall that our implementation furnishes the top two-ranked code block suggestions to the programmer. Then it is up to the programmer's purview to select the suggestion to be integrated in a given iOS application. The tables list the results that our approach produced as both `Rank 1` and `Rank 2`. Table 8.3–`Rank 1 or 2` presents the `YES` results, which appeared in Rank 2 while missing in Rank 1; the relatively high number of cases in this column supports our design choice of the plugin furnishing the two top-ranked suggestions to the programmer.

| | | Rank 1 | | | |
|---|---|---|---|---|---|
| **Native APIs** | TOTAL | YES | NO | 1/2 | NaCB |
| **String** | 25 | 19 | 4 | 2 | 0 |
| **ArrayList** | 22 | 13 | 5 | 3 | 1 |
| **HashMap/Dictionary** | 13 | 10 | 3 | 0 | 0 |
| **GPS Location** | 5 | 4 | 0 | 1 | 0 |
| **Accelerometer** | 2 | 2 | 0 | 0 | 0 |
| **BTLE** | 5 | 4 | 1 | 0 | 0 |
| **Wifi** | 5 | 4 | 1 | 0 | 0 |
| **Overall** | 75 | **56** | 14 | 6 | 1 |
| **Overall Yes (Rank 1 Only)** | | **74.7%** | **78.9%(Normalized)** | | |

Table 8.1: Evaluation results–Rank 1 Only– for target native API code block synthesis as {yes, no, 1/2 suitable, NaCB (not a code block).}

The total evaluation results are summarized in the last row of each of the above tables. Specifically, they report the total percentage of the `YES` outcomes obtained from Rank 1 only, Rank 2 only, and from either Rank 1 or 2, respectively. While the first result is around 75%, the third one stands at more than 10 percentage points higher at almost 87%. One may wonder why we decided to limit our number of reported top suggestions only to two. This limitation is dictated by primarily practical considerations—for the majority of our test cases, considering more than two suggestions quickly proved impractical, taking additional time without a match in increased accuracy. The Rank 2 results alone indicate poor performance, however these being the second option, their main purpose is to supplement the first ranked suggestions. Thus, the important component lies in the combination of Rank 1 or 2 as described above and highlighted in Table 8.3.

| | | Rank 2 | | | |
|---|---|---|---|---|---|
| **Native APIs** | TOTAL | YES | NO | 1/2 | NaCB |
| **String** | 25 | 12 | 4 | 8 | 1 |
| **ArrayList** | 22 | 6 | 5 | 1 | 10 |
| **HashMap/Dictionary** | 13 | 3 | 2 | 3 | 5 |
| **GPS Location** | 5 | 3 | 1 | 1 | 0 |
| **Accelerometer** | 2 | 1 | 0 | 1 | 0 |
| **BTLE** | 5 | 3 | 1 | 1 | 0 |
| **Wifi** | 5 | 4 | 1 | 0 | 0 |
| **Overall** | 75 | **32%** | 14 | 15 | 16 |
| **Overall Yes (Rank 2 Only)** | | **42.67%** | **34.05%(Normalized)** | | |

Table 8.2: Evaluation results–Rank 2 Only– for target native API code block synthesis as {yes, no, 1/2 suitable, NaCB (not a code block).}

| **Native APIs** | Total | Rank 1 or 2 |
|---|---|---|
| **String** | 25 | 22 |
| **ArrayList** | 22 | 18 |
| **HashMap/Dictionary** | 13 | 11 |
| **GPS Location** | 5 | 4 |
| **Accelerometer** | 2 | 2 |
| **BTLE** | 5 | 4 |
| **Wifi** | 5 | 4 |
| **Overall** | 75 | **65** |
| **Overall Yes (Rank 1 or 2)** | **86.7%** | **84.9%(Normalized)** |

Table 8.3: Rank 1 or 2 Results

## 8.3 Limiting Example

The `native-2-native` approach is not universally applicable and may fall short of the programmer's expectations in the presence of the impedance mismatch between the Java and Swift language vocabularies. The right side of Figure 8.1 shows one such example. The example makes use of the Java standard library's `HashMap` class, parameterized with `String` types as its key and value. Unfortunately, `HashMap` is a Java-specific concept that has very different names in other languages. In Swift, this programming idiom is supported by the `Dictionary` class. Without special provisions, `native-2-native` would return the code block depicted on the right side of Figure 8.1. This code block contains a use case of the MD5 hashing algorithm in Objective-C embedded into the control flow of the source document's

Java code. This outcome would be a direct consequence of the extracted keywords (mapping, hash, and put) steering the search toward a hashing algorithm instead of a `dictionary` data structure. Because such vocabulary dissimilarities are inevitable, `native-2-native` special-cases several highly common instances of these dissimilarities by consulting a static mapping of vocabulary keywords between Java and Swift.

```
//N2N-putHashMap.swift
/**
* Associates the specified value with the specified key in this map. If the
* map previously contained a mapping for the key, the old value is
* replaced.
*/
func putHashMap(/*unknown token*/, B: String, C: String) -> /*unknown token*/ {
    //Insert Native-2-Native result
    var putHashMapTemp:/*unknown token*/ = A
    putHashMapTemp./*unknown token*/(B,C)
    return putHashMapTemp
}
/* NATIVE-2-NATIVE RESULT
extension String  {
    var md5: String! {
        let str = self.cStringUsingEncoding(NSUTF8StringEncoding)
        let strLen = CC_LONG(self.lengthOfBytesUsingEncoding(NSUTF8StringEncoding))
        let digestLen = Int(CC_MD5_DIGEST_LENGTH)
        let result = UnsafeMutablePointer<CUnsignedChar>.alloc(digestLen)
        CC_MD5(str!, strLen, result)
        var hash = NSMutableString()
        for i in 0..<digestLen {
        hash.appendFormat("%02x", result[i])
    }

    result.dealloc(digestLen)
    return String(format: hash)
    }
}
*/
```

Figure 8.1: Synthesized Swift code snippet for GPS Location (left) and HashMap (right)

## 8.4 Normalization

As a presentation choice, the final row of each of the aforementioned Tables treats every subject case first as equally important without normalizing across them, and then display the normalized counterparts. We argue that our presentation choice is well-founded. When it comes to needing assistance when supporting a piece of functionality on several platforms, the average mobile developer would likely care more about sensor APIs, such as `GPS` and

`BTLE`, than fundamental data structures APIs, such as `String`, `ArrayList`, and `HashMap`. By normalizing we present an equal per-category weight for each of the APIs, which benefits the smaller sensor APIs which understandably contain fewer subject evaluation cases than their counterparts in data structures.

The normalized results in the above tables demonstrate that a high accuracy in usable synthesized code is still present. Further experimentation utilizing other mobile-development-specific APIs along with an equal presence of various data structures will likely lead to similar results as we have depicted here. This is an important aspect towards future expansion and applicability outside of just the Android/Java to iOS/Swift domain. More applicability and the practicality of this solution is discussed in greater detail below, in Chapter 9.

# Chapter 9

# Discussion

Based on the results of our evaluation, one can conclude that the approach is effective enough to serve as a practical tool for mobile programmers supporting cross-platform applications. Because the automatically suggested code does not need to be perfect to provide a high degree of utility to the programmer, our algorithms proved surprisingly fit for the purposes intended. However, it is not only the fitness of our algorithms that explains the effectiveness of our approach. These algorithms work hand-in-hand with the realities of the mobile market, the availability of web-based programming resources, and the process of suggesting equivalent native APIs being manageable via tool automation.

## 9.1 Externalities

Despite its practical utility, our approach has several limitations. The model underlying the searching and ranking algorithms of our approach is bound by the dimensions of the feature vector. In other words, the accuracy of the algorithms is reversely proportional to the size of the total number of unique tokens comprising a given code block. As a result, mobile developers are likely to find our approach most effective in those cases when they need to find equivalent iOS code for small to medium (10-30 lines of code) Android native code blocks. The second limitation stems from the original closed development model for the iOS platform. Closed models traditionally result in reduced sharing of programming solutions. Exacerbating the conditions for evaluating the applicability of our reference implementation is a relative newness of the Swift language. In fact, we were surprised that our reference implementation was able to synthesize correct suggestions from a relatively limited set of web-based Swift programming resources. Nevertheless, several major technological trends are likely to address this limitation. For one, Swift will be open-sourced in coming releases, while the amount of available Swift code examples on the web seems to grow by leaps and bounds.

The generation of test cases was partially automated as well to facilitate running exhaustive tests on the data structures APIs. Since the process involved access various private members of the APIs in such a way as to generate run-able input Java source code, we leveraged Java's Reflection capabilities [25] to facilitate this process. Intermingling these subject API cases with sensor APIs proved a solid mechanism for evaluating breadth in the reference implementation, and thus the underlying approach.

## 9.2 Imperative vs. Declarative

Synthesizing Swift from given Java input is a necessarily difficult case of cross-language translation. Because Swift is much more declarative (i.e., concise) than Java, the translation must produce more declarative output from more loquacious input [21, 39]. As software engineering is becoming more declarative in terms of languages, specifications, and invariants, our approach holds a lot of promise for automatically transitioning current mainstream methods of expressing programming information into their declarative counterparts. For example, our approach can be used to get rid of the wordiness of anonymous inner classes in the pre-Java8 world, replacing them with code lambda expressions [33, 45].

The declarative vs. imperative language paradigm is nicely summarized in Figure 9.1 shown below. When receiving directions in today's GPS enabled world, we simply want the address (*what*) not the *how* provided in the comic by XKCD. The same *how* vs. *what* is present in Java and Swift, respectively and it becomes clear that the excessive language needed to represent identical issues creates additional complexity. Generating imperative Java source code from declarative and concise Swift is fundamentally simpler and thus could ensure the reverse of this approach also be a feasible and practical use case.

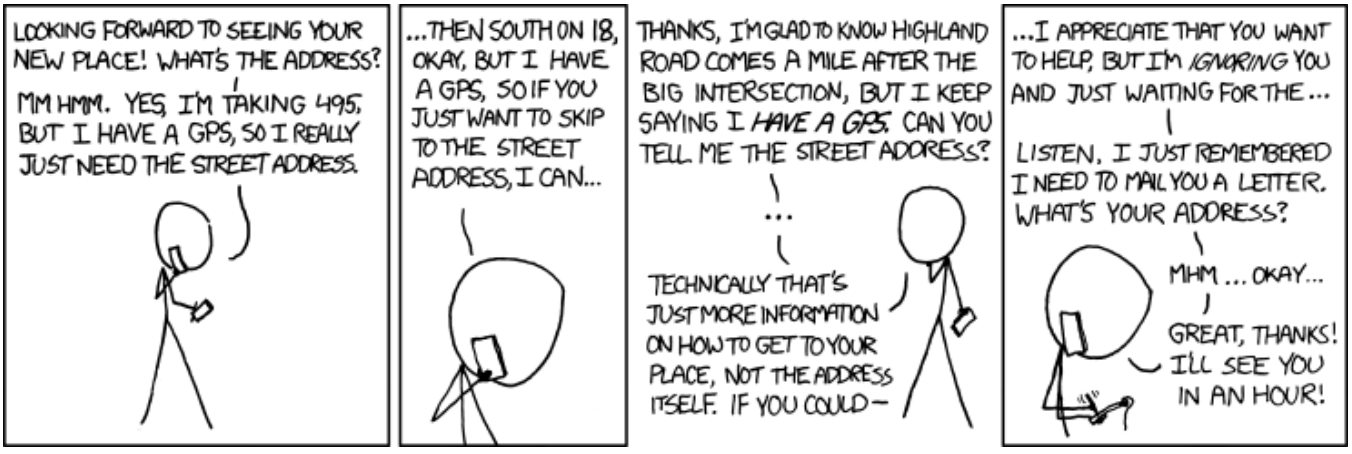


Figure 9.1: Comic illustrating declarative vs. imperative programming. [68] - R. Munroe. I dont want directions. `http://xkcd.com/783/`, 2015. Used under fair use, 2015

The Hover-over text for [68]: "Yes, I understand that the turn is half a mile past the big field, but my GPS knows that, too. This would be easier if you weren't about to ask me to repeat it all back to you [1]." This further entertains the absurdity of this situation being thought in terms of imperative rather than declarative.

## 9.3 Discussion Summary

While some minor externalities made evaluation of the approach more complicated than necessary, the results proved largely positive in nature. There are various underlying models that could be incorporated alongside the vector space and linear models currently in place, as well as additional open-source code samples from iOS for automated testing. However, the underlying beliefs about the approach are validated and the reference implementation demonstrates this more than adequately. In addition, by choosing to tackle this problem from Android/Java to iOS/Swift, an unintentional computational difficulty of translating from imperative to declarative was incurred. However, the results show that irregardless of this, it is still largely effective at automatic code synthesis for cross-platform mobile developers. Finally, the various externalities and programming language discrepancies demonstrate the cross-domain applicability of this approach. Given the extensible nature of the reference implementation alongside the generality of the approach, tailoring `native-2-native` to other code synthesis tasks via web-based resources proves a realistic next direction.

[1] In XKCD comics, there is always text when you hover-over the comic. It usually has an additional joke in the same manner or an extension of the original comic's underlying joke.

# Chapter 10

# Future Work

This chapter describes the various potential future work directions with their practicality and benefit. The following sections discuss the four major future work directions: preference server, clustering, model translation, and platform expansion.

## 10.1 Preference Server

One potential direction for improving this work entails reusing the search data in an intelligent manner. Our current approach relies on generating a document corpus upon each query, but these results are discarded for all future queries, not just locally for a particular user but across all users. If instead, these meta-data objects were saved globally to become accessible for all users, then our approach can be further optimized in the following two major ways. First, the potential use of a preference server would facilitate an ability to predict the semantics of future queries made by a particular user based on previous inquiries they made. Figure 10.1 demonstrates how the model would change from the original approach. The notion of a preference server facilitates access to relevant data and produces another medium of similarity in the ranking component. This predicted potential query could be weighted along with the newly generated meta-data object and form the user's next query.

The incorporation of a preference server fundamentally improves the user experience. Specifically, this style of improvement makes use of the developer's lazy programming style, a mentality many young developers maintain. Consider a code block used occasionally or periodically, in the same exact manner. For example, the process to read in a file in Java through a `BufferedReader`, is a universally used 10 line code block. However, many developers might have never memorized or really grasped this code block. Instead, they return to the same StackOverflow post every single time they need to write this particular code block. A preference server would quickly adapt to a search of this style and store the preferred StackOverflow post. Then all future queries would not require any data mining when the

developer is requesting information about reading in a file.

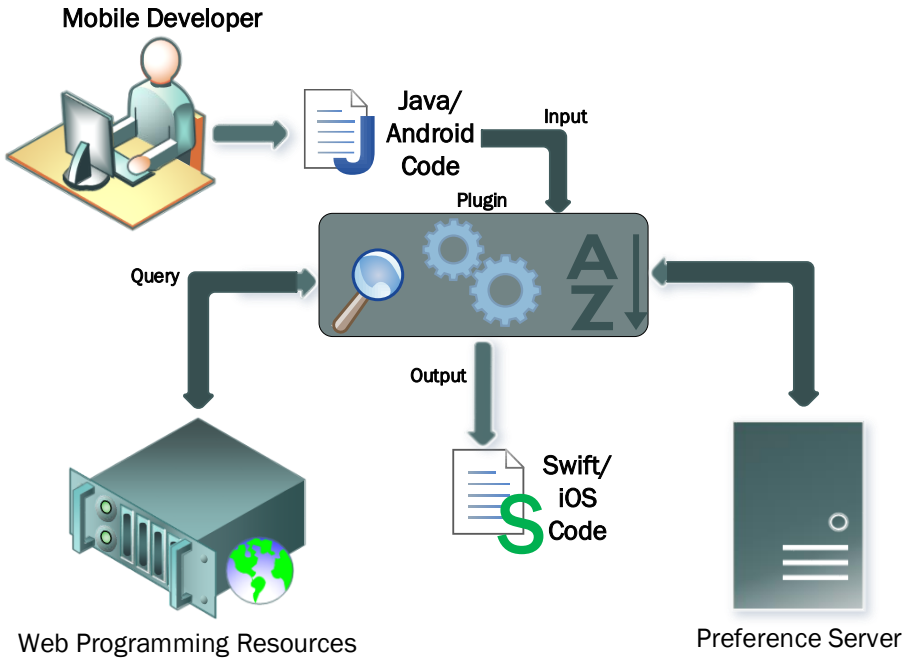


Figure 10.1: Augmented approach utilizing preference server

## 10.2 Incremental Clustering of Documents

The second optimization comes from storing all meta-data objects across all users in a set of online clusters. The clusters would be defined by some similarity measure related to the original ranking model. As described in the background information, the clusters would be arranged in a hierarchal manner as demonstrated in Figure 10.2. This optimization would facilitate a speedup when a user's query is within the cluster and within some threshold of similarity specified by the user, as one then would not need to continue a full web search for potential Swift code blocks.

The real advantage to utilizing a set of online clusters is that they would freely change and adapt as new queries were made by all users. This allows the model to incorporate specific user preferences, represented as a cluster, and similar users would also be stored in clusters. Essentially, by setting up an infrastructure of clusters, we allow various new features about user data and other preferences to improve the quality of results while also reducing the need to data mine new information at every step of the process.

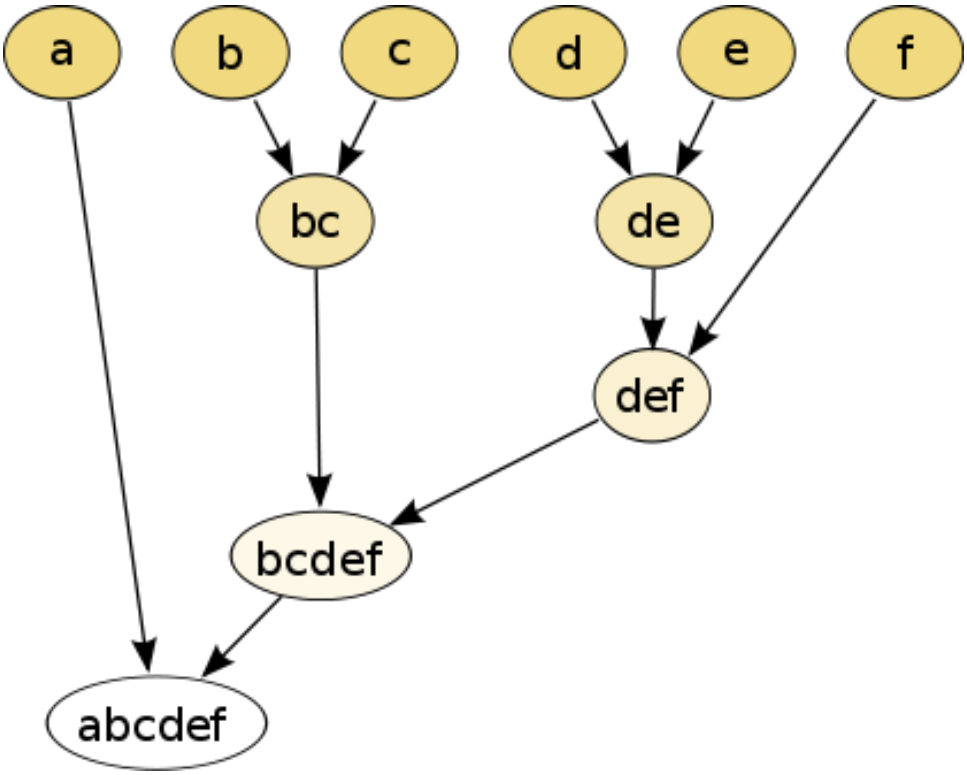


Figure 10.2: Dendrogram example of hierarchical clustering. [65] - Mhbrugman. Hierarchical clustering simple dendogram. Wikipedia, 2008. Used under fair use, 2015

## 10.3 Model Translation

Another potential avenue of improvement is to incorporate the translation of the "Model" component of an application's Model-View-Controller pattern. This enhancement could be easily accomplished by integrating with `native-2-native` the previously discussed [49] or another source-to-source translator of non-Android Java code. Similarly, we can further improve the translation and ranking by allowing users to rate our approach's returned Swift code blocks. These ratings would then be incorporated into future queries not just for this user but for all users making similar queries.

## 10.4 Platform Expansion

Yet another future work direction would expand the number of target platforms to Windows Phone and platform-independent JavaScript frameworks such as PhoneGap [29]. The developer would implement a new feature on one platform and then will get suggestions for

all the other supported platforms. Comparing the software metrics of the equivalent code blocks on different platforms can shed insights on various software engineering properties of different languages and architectures. Figure 10.3 demonstrates the design of Phonegap and how it could be integrated to further increase the marketability of the underlying work in this thesis.

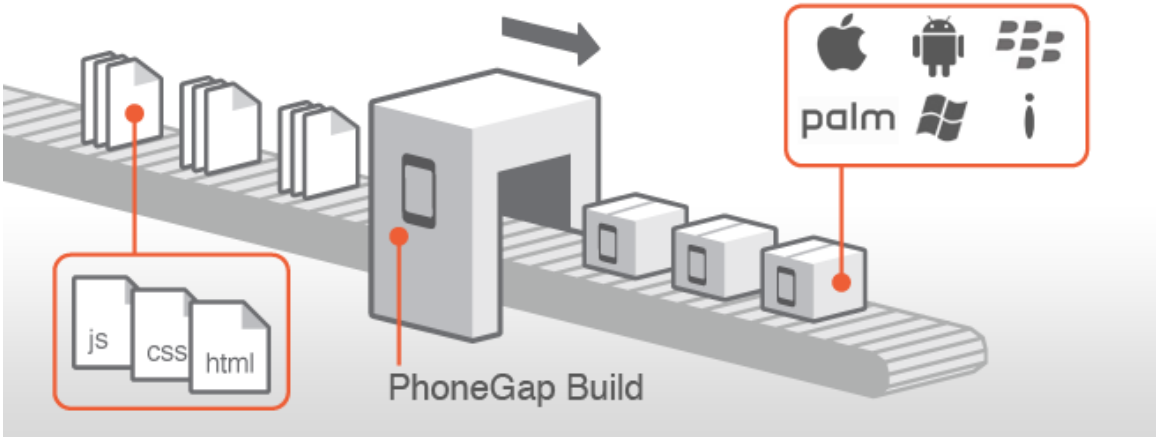


Figure 10.3: Phonegap usability example: [29] - R. Ghatol and Y. Patel. Beginning PhoneGap: Mobile Web Framework for JavaScript and HTML5. 2012. Used under fair use, 2015

In general, platform expansion is a necessary, but not sufficient form of improvement for any application geared towards mobile developers. While the two primary players in the current market are Android and iOS, many developers look forward to future models of the Windows Phone and also build HTML5 in-browser versions of their applications to penetrate more of the market. Similarly, it becomes obvious that *native-2-native* must adopt native translations in other languages.

# Chapter 11

# Conclusions and Availability

## 11.1 Conclusions

This thesis has presented a novel approach for automatic code synthesis from Android/Java to iOS/Swift by utilizing popular web-based programming resources. To enable our approach, we first gleaned several insights underlying the realities of modern mobile software development and the mobile computing market. These insights are summarized in three central points: (1) The modern mobile development marketplace ensues a constant struggle between major platforms for domination, forcing a large quantity of functionality necessarily sharing vocabulary across these major mobile platforms. (2) With the various new characteristics defining the modern mobile software developer, as well as the radical cultural shift in the field, the quantity and quality of web-based resources has grown exponentially. In turn, this makes many developers rely on these web-based resources as their primary source of reference information about a wide assortment of technical questions and programming issues. (3) The fundamental realization that ascertaining usage information about native APIs on one mobile platform, having just done this process on another yields negligible support. More so, this falls into the category of *accidental complexity*, as David Brooks discusses in [15], making this type of issue a prime candidate for automation.

Having understood these underlying insights, we then presented concrete realization of our approach as `native-2-native`. The approach includes first, extracting semantics from input java source code to produce an underlying source meta-data object as well as query keywords. Next, a searching algorithm is initiated on these keywords to mine various relevant documents from numerous web-based resources. These results are then funneled into corresponding meta-data objects and ranked alongside the source meta-data object to produce a final sorted list by document similarity. Finally, a code synthesis algorithm performs a control-flow copy by translating the input Java source logic, lacking the Android specific APIs, into output Swift source code and inserts the best ranked result. This synthesized code block is

returned to the requesting developer.

The evaluation of our approach and its reference implementation show that `native-2-native` produces useful and intended functionality in as many as 86% of subject native APIs. These results indicate that the presented approach can become a pragmatic and valuable programming tool in the arsenal of mobile software developers. A discussion of external domain applicability demonstrated that this approach is not restricted to Java and Swift translations, but can be tailored and expanded to other languages and corresponding APIs. Finally, we presented a concrete set of future directions from which this approach could be improved and optimized to undertake more difficult tasks.

## 11.2 Availability

`native-2-native` is available from `https://github.com/antuanb/Native-2-Native`. The site includes the full source code for the approach, including the integration with Eclipse, open-source license, detailed results, and additional evaluation use cases. Please refer any questions about source code use and issues to Antuan Byalik - `antuanb@cs.vt.edu`

# Bibliography


[1] V. Anand. *Multi-interface connectivity on Android*. PhD thesis, Faculty of the Graduate School of the University at Buffalo, State University of New York, 2014.

[2] B. Antunes, J. Cordeiro, and P. Gomes. An approach to context-based recommendation in software development. In *Proceedings of the Sixth ACM Conference on Recommender Systems*, RecSys '12, pages 171–178. ACM, 2012.

[3] Apple. Platforms state of the union, session 102. Apple Worldwide Developers Conference, 2014.

[4] C. Arthur. Ten things to know about blackberry – and how much trouble it is (or isn't) in. TheGuardian.com, 2014.

[5] M. Auli, M. Galley, C. Quirk, and G. Zweig. Joint language and translation modeling with recurrent neural networks. In *EMNLP*, volume 3, page 0, 2013.

[6] D. Austin. ediscovery best practices: Cluster documents for more effective review, 2011.

[7] A. Bacchelli, L. Ponzanelli, and M. Lanza. Harnessing stack overflow for the IDE. In *Proceedings of the Third International Workshop on Recommendation Systems for Software Engineering*, RSSE '12, pages 26–30. IEEE Press, 2012.

[8] D. Bahdanau, K. Cho, and Y. Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

[9] S. Bajracharya, J. Ossher, and C. Lopes. Sourcerer: An internet-scale software repository. In *Proceedings of the 2009 ICSE Workshop on Search-Driven Development-Users, Infrastructure, Tools and Evaluation*, SUITE '09, pages 1–4. IEEE Computer Society, 2009.

[10] J. Barnett. Building a cross-platform mobile game with html5.

[11] M. W. Berry and M. Castellanos. Survey of text mining. *Computing Reviews*, 45(9):548, 2004.

[12] Booyabazooka. Relational database terms. Wikipedia, 2008.

[13] A. Borriello, F. Melillo, and G. Canfora. Migrating android applications towards service-centric architectures with sip2share. In *Proceedings of the 17th European Conference on Software Maintenance and Reengineering (CSMR'13)*, pages 413–416. IEEE, 2013.

[14] L. Breiman. Bagging predictors. *Machine Learning*, 24(2):123–140, 1996.

[15] F. P. Brooks. No silver bullet: Essence and accidents of software engineering. *Computer*, 20(4):10–19, apr 1987.

[16] C. Buckley. The importance of proper weighting methods. In *Proceedings of the workshop on Human Language Technology*, pages 349–352. Association for Computational Linguistics, 1993.

[17] J. Callan. Document filtering with inference networks. In *Proceedings of the 19th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 262–269. ACM, 1996.

[18] C. Castillo. Effective web crawling. In *ACM SIGIR Forum*, volume 39, pages 55–56. ACM, 2005.

[19] E. Chen, S. Ogata, and K. Horikawa. Offloading android applications to the cloud without customizing android. In *Proceedings of 2012 IEEE International Conference on Pervasive Computing and Communications (PerCom'12)*, pages 788–793. IEEE, 2012.

[20] E. Chisholm and T. G. Kolda. New term weighting formulas for the vector space method in information retrieval. *Computer Science and Mathematics Division, Oak Ridge National Laboratory*, 1999.

[21] S. Efftinge, M. Eysholdt, J. Köhnlein, S. Zarnekow, R. von Massow, W. Hasselbring, and M. Hanus. Xbase: implementing domain-specific languages for java. In *ACM SIGPLAN Notices*, volume 48, pages 112–121. ACM, 2012.

[22] T. Elsayed, J. Lin, and D. W. Oard. Pairwise document similarity in large collections with mapreduce. In *Proceedings of the 46th Annual Meeting of the Association for Computational Linguistics on Human Language Technologies: Short Papers*, pages 265–268. Association for Computational Linguistics, 2008.

[23] en.wikipedia. Information-retrieval-model, 2015.

[24] C.-L. Fok, G.-C. Roman, and G. Hackmann. A lightweight coordination middleware for mobile computing. In *Coordination Models and Languages*, pages 135–151. Springer, 2004.

[25] I. R. Forman, N. Forman, and J. V. Ibm. Java reflection in action. 2004.

[26] P. S. Foundation. 2to3 - automated python 2 to 3 code translation. `https://docs.python.org/3.0/library/2to3.html`, 2014.

[27] B. C. Fung, K. Wang, and M. Ester. Hierarchical document clustering using frequent itemsets. In *SDM*, volume 3, pages 59–70. SIAM, 2003.

[28] C. Garling. iphone coding language now worlds third most popular. Wired, 2012.

[29] R. Ghatol and Y. Patel. *Beginning PhoneGap: Mobile Web Framework for JavaScript and HTML5*. 2012.

[30] D. I. Ginny Mies, Armando Rodriguez. Apple ios 6 vs. android vs. windows phone (comparison chart), 2012.

[31] J. Glass and E. Derr. Document similarity detection and classification system, Aug. 12 2004. US Patent App. 10/710,918.

[32] M. H. Goadrich and M. P. Rogers. Smart smartphone development: ios versus android. In *Proceedings of the 42nd ACM technical symposium on Computer science education*, pages 607–612. ACM, 2011.

[33] J. C. González-Moreno, M. T. Hortala-Gonzalez, F. J. Lopez-Fraguas, and M. Rodríguez-Artalejo. An approach to declarative programming based on a rewriting logic. *The Journal of Logic Programming*, 40(1):47–87, 1999.

[34] Google. Android market. `https://play.google.com/store/apps?hl=en`, 2010 (accessed December 30, 2014).

[35] Google. Android's google play beats app store with over 1 billion apps, now officially largest. Phonearena.com, 2013.

[36] Google. Introduction to the aidl interface. `http://developer.android.com/guide/components/aidl.html`, 2014 (accessed December 30, 2014).

[37] W. Gordon. Ask lh: Is windows phone ready to replace my iphone or android? `http://img.gawkerassets.com/img/18isacnob15c9jpg/original.jpg`, 2013.

[38] J. Han, M. Kamber, and J. Pei. *Data mining: concepts and techniques: concepts and techniques*. Elsevier, 2011.

[39] M. Hanus. Multi-paradigm declarative languages. In *Logic Programming*, pages 45–75. Springer, 2007.

[40] harrygooz33. Share your nexus 5 screenshots! `http://forums.androidcentral.com/attachments/google-nexus-5/148907d1416334124t-share-your-nexus-5-screenshots-screenshot_2014-11-18-17-49-51.jpg`, 2014.

[41] J. Herrman. What windows phone 7 could have been. Gizmodo. Gawker Media, 2010.

[42] E. Holder, E. Shah, M. Davoodi, and E. Tilevich. Cloud twin: Native execution of android applications on the windows phone. In *Proceedings of the28th IEEE/ACM International Conference on Automated Software Engineering (ASE'13)*, pages 598–603. IEEE, 2013.

[43] R. Holmes and G. C. Murphy. Using structural context to recommend source code examples. In *Proceedings of the 27th International Conference on Software Engineering*, ICSE '05, pages 117–125. ACM, 2005.

[44] M. Honan. Apple unveils iphone. Macworld, 2007.

[45] C. S. Horstmann. Lambda expressions in java 8. 2014.

[46] W. K. Horton. *Designing web-based training: How to teach anyone anything anywhere anytime*, volume 1. Wiley New York, NY, 2000.

[47] A. Hotho, S. Staab, and G. Stumme. Ontologies improve text document clustering. In *Data Mining, 2003. ICDM 2003. Third IEEE International Conference on*, pages 541–544. IEEE, 2003.

[48] N. Ingraham. Apple's app store has passed 100 billion app downloads. The Verge, 2015.

[49] J2ObjC.org. J2objc, 2015.

[50] P. Jackson and I. Moulinier. *Natural language processing for online applications. Text retrieval, extraction and categorization*, volume 5 of *Natural Language Processing*. Benjamins, 2002.

[51] S. Jobs. Third party applications on the iphone. Apple Inc, 2007.

[52] N. Khedekar. Six features apple borrowed from android and others for ios 8. `http://tech.firstpost.com/news-analysis/six-features-apple-borrowed-from-android-and-others-for-ios-8-225038.html`, 2014.

[53] J. Kim, S. Lee, S.-w. Hwang, and S. Kim. Towards an intelligent code search engine. 2010.

[54] M. Konchady. Text mining application programming (programming series). *Charles River Media*, pages 197–202, 2006.

[55] R. R. Korfhage. Information storage and retrieval. 2008.

[56] G. Kortuem. Proem: a middleware platform for mobile peer-to-peer computing. *ACM SIGMOBILE Mobile Computing and Communications Review*, 6(4):62–64, 2002.

[57] Y.-W. Kwon and E. Tilevich. Cloud refactoring: Automated transitioning to cloud-based services. *Automated Software Engg.*, 21(3):345–372, Sept. 2014.

[58] B. Larsen and C. Aone. Fast and effective text mining using linear-time document clustering. In *Proceedings of the fifth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 16–22. ACM, 1999.

[59] M. Lee, B. Pincombe, and M. Welsh. An empirical evaluation of models of text document similarity. *Cognitive Science*, 2005.

[60] H. Li, X. Zhao, Z. Xing, L. Bao, X. Peng, D. Gao, and W. Zhao. amassist: In-ide ambient search of online programming resources. In *Software Analysis, Evolution and Reengineering (SANER), 2015 IEEE 22nd International Conference on*, pages 390–398, March 2015.

[61] M. Linares-Vásquez, G. Bavota, C. Bernal-Cárdenas, M. Di Penta, R. Oliveto, and D. Poshyvanyk. Api change and fault proneness: A threat to the success of android apps. In *Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering (FSE'13)*, pages 477–487. ACM, 2013.

[62] F. Manjoo. A murky road ahead for android, despite market dominance. 2015.

[63] C. D. Manning, P. Raghavan, H. Schütze, et al. *Introduction to information retrieval*, volume 1. Cambridge university press Cambridge, 2008.

[64] T. Mens and T. Tourwé. A survey of software refactoring. *Software Engineering, IEEE Transactions on*, 30(2):126–139, 2004.

[65] Mhbrugman. Hierarchical clustering simple dendogram. Wikipedia, 2008.

[66] T. Mikolov. Statistical language models based on neural networks. *Presentation at Google, Mountain View, 2nd April*, 2012.

[67] M. D. N. (MSDN). Windows phone marketplace content policies, 2012.

[68] R. Munroe. I don't want directions. `http://xkcd.com/783/`, 2015.

[69] W. F. Opdyke. *Refactoring object-oriented frameworks*. PhD thesis, University of Illinois at Urbana-Champaign, 1992.

[70] M. P. Plezbert and R. K. Cytron. Does just in time=better late than never? In *Proceedings of the 24th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, pages 120–131. ACM, 1997.

[71] L. Ponzanelli, A. Bacchelli, and M. Lanza. Seahawk: Stack Overflow in the IDE. In *Proceedings of the 2013 International Conference on Software Engineering*, pages 1295–1298. IEEE, 2013.

[72] L. Ponzanelli, G. Bavota, M. Di Penta, R. Oliveto, and M. Lanza. Mining stackoverflow to turn the ide into a self-confident programming prompter. In *Proceedings of the 11th Working Conference on Mining Software Repositories*, MSR 2014, pages 102–111. ACM, 2014.

[73] L. Ponzanelli, G. Bavota, M. D. Penta, R. Oliveto, and M. Lanza. Prompter: A self-confident recommender system. In *Proceedings of the 2014 IEEE International Conference on Software Maintenance and Evolution*, ICSME '14, pages 577–580. IEEE Computer Society, 2014.

[74] A. Puder and O. Antebi. Cross-compiling android applications to ios and windows phone 7. *Mob. Netw. Appl.*, 18(1):3–21, Feb. 2013.

[75] M. S. R. Llamas, M. Chau. Android and ios squeeze the competition, swelling to 96.3smartphone operating system market for both 4q14 and cy14, according to idc, 2015.

[76] V. V. Raghavan and S. M. Wong. A critical analysis of vector space model for information retrieval. *Journal of the American Society for information Science*, 37(5):279–287, 1986.

[77] M. M. Rahman and C. K. Roy. Surfclipse: Context-aware meta-search in the ide. In *Proceedings of the 2014 IEEE International Conference on Software Maintenance and Evolution*, ICSME '14, pages 617–620. IEEE Computer Society, 2014.

[78] J. Ramos. Using tf-idf to determine word relevance in document queries. In *Proceedings of the first instructional conference on machine learning*, 2003.

[79] S. Ranger. ios versus android. apple app store versus google play: Here comes the next battle in the app wars. ZDNet, 2015.

[80] M. Robillard, R. Walker, and T. Zimmermann. Recommendation systems for software engineering. *Software, IEEE*, 27(4):80–86, July 2010.

[81] M. P. Robillard, W. Maalej, R. J. Walker, and T. Zimmermann, editors. *Recommendation Systems in Software Engineering*. Springer, 2014.

[82] N. Saeed and Y. Yang. Incorporating blogs, social bookmarks, and podcasts into unit teaching. In *Proceedings of the tenth conference on Australasian computing education-Volume 78*, pages 113–118. Australian Computer Society, Inc., 2008.

[83] N. Saeed, Y. Yang, and S. Sinnappan. Emerging web technologies in higher education: A case of incorporating blogs, podcasts and social bookmarks in a web programming course based on students' learning styles and technology preferences. *Journal of Educational Technology & Society*, 12(4):98–109, 2009.

[84] G. Salton and C. Buckley. Term-weighting approaches in automatic text retrieval. *Information processing & management*, 24(5):513–523, 1988.

[85] G. Salton and M. J. McGill. Introduction to modern information retrieval. 1986.

[86] G. Salton, A. Wong, and C.-S. Yang. A vector space model for automatic indexing. *Communications of the ACM*, 18(11):613–620, 1975.

[87] G. Salton, A. Wong, and C. S. Yang. A vector space model for automatic indexing. *Commun. ACM*, 18(11):613–620, Nov. 1975.

[88] H. Sapkota. j2objc-eclipse-plugin, 2015.

[89] R. Soricut and D. Marcu. Sentence level discourse parsing using syntactic and lexical information. In *Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology-Volume 1*, pages 149–156. Association for Computational Linguistics, 2003.

[90] StackOverflow.com. Stack overflow, 2015.

[91] StackOverflow.com. Stack overflow developer survey 2015, 2015.

[92] StackOverflow.com. Usage of /info - stack exchange api, 2015.

[93] Statista. Number of apps available in leading app stores as of may 2015, 2015.

[94] M. Steinbach, G. Karypis, V. Kumar, et al. A comparison of document clustering techniques. In *KDD workshop on text mining*, volume 400, pages 525–526. Boston, 2000.

[95] S. Subramanian and R. Holmes. Making sense of online code snippets. In *Proceedings of the 10th Working Conference on Mining Software Repositories*, MSR '13, pages 85–88. IEEE Press, 2013.

[96] I. Sutskever, O. Vinyals, and Q. V. Le. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112, 2014.

[97] W. Takuya and H. Masuhara. A spontaneous code recommendation tool based on associative search. In *Proceedings of the 3rd International Workshop on Search-Driven Development: Users, Infrastructure, Tools, and Evaluation*, SUITE '11, pages 17–20. ACM, 2011.

[98] A.-H. Tan et al. Text mining: The state of the art and the challenges. In *Proceedings of the PAKDD 1999 Workshop on Knowledge Disocovery from Advanced Databases*, volume 8, page 65, 1999.

[99] S. Thummalapenta and T. Xie. Parseweb: A programmer assistant for reusing open source code on the web. In *Proceedings of the Twenty-second IEEE/ACM International Conference on Automated Software Engineering*, ASE '07, pages 204–213. ACM, 2007.

[100] D. Tilson, C. Sørensen, and K. Lyytinen. Change and control paradoxes in mobile infrastructure innovation: the android and ios mobile operating systems cases. In *System Science (HICSS), 2012 45th Hawaii International Conference on*, pages 1324–1333. IEEE, 2012.

[101] B. Vasilescu, V. Filkov, and A. Serebrenik. Stackoverflow and github: Associations between software development and crowdsourced knowledge. In *Social Computing (SocialCom), 2013 International Conference on*, pages 188–195, Sept 2013.

[102] T. Vidas, C. Zhang, and N. Christin. Toward a general collection methodology for android devices. *digital investigation*, 8:S14–S24, 2011.

[103] M. Vizard. Mobile developer becomes redundant, 2013.

[104] P. Willett. Recent trends in hierarchic document clustering: a critical review. *Information Processing & Management*, 24(5):577–597, 1988.

[105] I. H. Witten and E. Frank. *Data Mining: Practical machine learning tools and techniques*. Morgan Kaufmann, 2005.

[106] E. Wong, J. Yang, and L. Tan. Autocomment: Mining question and answer sites for automatic comment generation. In *Automated Software Engineering (ASE), 2013 IEEE/ACM 28th International Conference on*, pages 562–567, Nov 2013.

[107] www.tiobe.com. Usage of tiobe language use statistics, 2015.

[108] N. Ye et al. *The handbook of data mining*, volume 24. Lawrence Erlbaum Associates, Publishers Mahwah, NJ/London, 2003.

[109] J. Yep. ios 7 (beta 1): A change worth the wait. `https://jordanstechstop.files.wordpress.com/2013/06/ios-7-home-screen.jpg?w=640`, 2013.

[110] M. Yoshikawa, T. Amagasa, T. Shimura, and S. Uemura. Xrel: a path-based approach to storage and retrieval of xml documents using relational databases. *ACM Transactions on Internet Technology*, 1(1):110–141, 2001.

[111] C. T. Yu and G. Salton. Precision weightingan effective automatic indexing method. *Journal of the ACM (JACM)*, 23(1):76–88, 1976.

[112] A. Zagalsky, O. Barzilay, and A. Yehudai. Example overflow: Using social media for code recommendation. In *Recommendation Systems for Software Engineering (RSSE), 2012 Third International Workshop on*, pages 38–42, June 2012.

[113] H. Zhong, S. Thummalapenta, T. Xie, L. Zhang, and Q. Wang. Mining api mapping for language migration. In *Proceedings of the 32Nd ACM/IEEE International Conference on Software Engineering - Volume 1*, ICSE '10, pages 195–204. ACM, 2010.

[114] A. Zwicky. Arnold zwicky's blog. `http://arnoldzwicky.org/2012/03/`, 2012.

# Appendix A

# Code Metrics: Flow

This appendix begins the code metrics group of appendixes. The focus here is to demonstrate various aspects of the code base used to build the reference implementation. Here specifically, we detail the flow of the entire source code. Figure A.1 details the code flow.

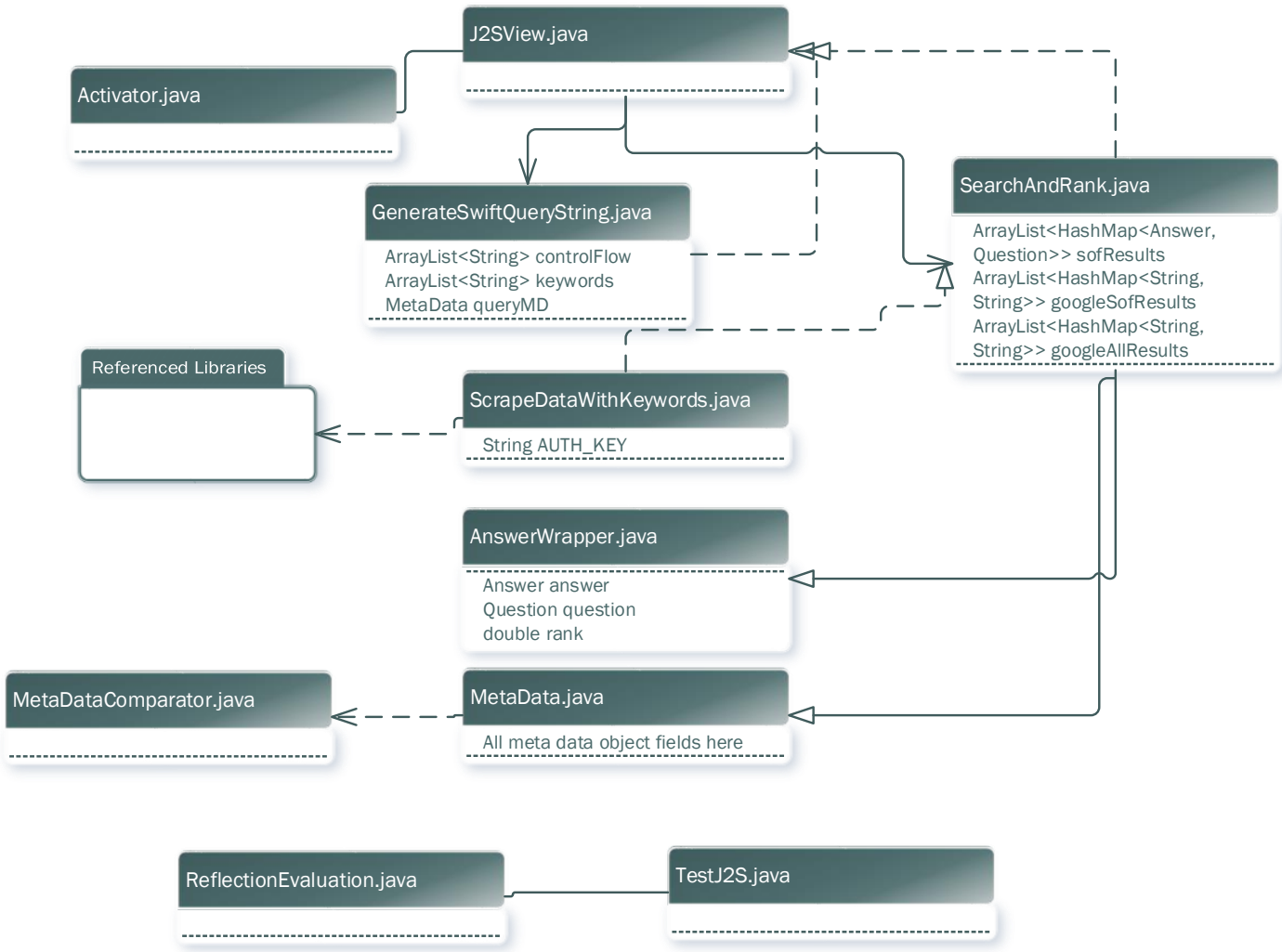


Figure A.1: Source Code Flow in UML

# Appendix B

# Code Metrics: ULoC

This appendix displays information about the total number lines of code used to build the reference implementation. Table B.1 details the total uncommented lines of code for the entire project and the sub-classes. This includes only code directly effecting run time, excluding comments and supporting materials. The test class shown displays information about the various unit testing methods.

| File | Number of Lines |
|---|---|
| SearchAndRank.java | 493 |
| ReflectionEvaluation.java | 471 |
| GenerateSwingQueryString.java | 348 |
| MetaData.java | 249 |
| J2SView.java | 210 |
| ScrapeDataWithKeywords.java | 139 |
| TestJ2S.java | 154 |
| AnswerWrapper.java | 28 |
| Activator.java | 24 |
| MetaDataComparator.java | 15 |
| Total | 2198 |

Table B.1: Uncommented lines of code (ULoC) in reference implementation

# Appendix C

# Code Metrics: LoC

This appendix displays information about the total number of lines of code used to build the reference implementation. Here, we focus on the entire code base, including comments, documentation, and supporting information necessary in running the reference implementation successfully. This information is shown in Table C.1.

| File | Number of Lines |
|---|---|
| SearchAndRank.java | 581 |
| ReflectionEvaluation.java | 753 |
| GenerateSwingQueryString.java | 521 |
| MetaData.java | 445 |
| J2SView.java | 278 |
| ScrapeDataWithKeywords.java | 178 |
| TestJ2S.java | 212 |
| AnswerWrapper.java | 35 |
| Activator.java | 32 |
| MetaDataComparator.java | 17 |
| Total | 3052 |

Table C.1: Lines of code (LoC) in reference implementation

# Appendix D

# Code Metrics: Referenced Libraries

This appendix displays information about the various referenced libraries necessary to complete the reference implementation. A brief explanation of how each package was important is shown with its corresponding jar.

1. stackexchange-java-core-2.2.1-SNAPSHOT.jar
2. stackexchange-java-schema-2.2.1-SNAPSHOT.jar
3. stackexchange-java-sdk-release-2.2.1-SNAPSHOT-release.jar
4. jsoup-1.8.2.jar
5. gson-1.4.jar
6. google-ajax-core-0.1.jar
7. google-ajax-schema-0.1.jar
8. google-ajax-release-0.1-release.jar

From the above list, the stackexchange jars (1, 2, and 3) provide access to the StackOverflow API. Jars 4 and 5 were used to convert objects into JSON and parse HTML for code tags and text tags, respectively. The final three jars, (6, 7, and 8) provided access to the Google API. All of the reference jars provided functionality during the data mining and parsing stages of the implementation. As there were access APIs and corresponding wrappers to parse the objects as needed, this allowed us to focus more on the core elements of the implementation rather than on the data scraping itself.

# Appendix E

# Subject Cases: BTLE

This appendix begins the subject cases that were used to facilitate the evaluation shown in Chapter 8. We show the BTLE subject cases which represent the java input for `native-2-native`.

```
/**
 * Write a value to a given characteristic.
 * @param characteristic The characteristic to write to
 * @return Value written successfully or not
 */
public boolean send(BluetoothGattCharacteristic characteristic) {
  if (taskQueue.isEmpty()) {
    System.out.println("empty queue");
    return false;
  }

  try {
    Thread.sleep(500);
  } catch (InterruptedException e) {
    // TODO Auto-generated catch block
    e.printStackTrace();
  }

  characteristic.setValue(taskQueue.get(0).getTaskQueueEntry());
  characteristic.setWriteType(BluetoothGattCharacteristic.WRITE_TYPE_DEFAULT);
  return mBluetoothGatt.writeCharacteristic(characteristic);
}
```

Figure E.1: BTLE subject case for sending data

Figure E.1 shows a basic example of sending information via a BTLE characteristic in Android/Java.

```
/**
 * Request a read on a given {@code BluetoothGattCharacteristic}. The read
 * result is reported asynchronously through the BluetoothGattCallback onCharacteristicRead
 * callback.
 *
 * @param characteristic The characteristic to read from.
 */
public void readCharacteristic(BluetoothGattCharacteristic characteristic) {
  if (mBluetoothAdapter == null || mBluetoothGatt == null) {
    Log.w(TAG, "BluetoothAdapter not initialized");
    return;
  }
  mBluetoothGatt.readCharacteristic(characteristic);
}
```

Figure E.2: BTLE subject case for reading data

Figure E.2 shows a basic example of reading information from a BTLE characteristic in Android/Java.

```
/**
 * Write data greater than 20 bytes to a specified characteristic.
 */
private void sendData(BluetoothGattCharacteristic characteristic) {
  final Intent intent = new Intent(BluetoothLeService.SEND_ID_BACK);
  TaskQueue tq = new TaskQueue();
  while (sendDataIndex < dataToSend.length) {
    int amountToSend = dataToSend.length - sendDataIndex > 20 ? 20 :
            dataToSend.length - sendDataIndex;
    byte[] currentSend = Arrays.copyOfRange(dataToSend, sendDataIndex,
            amountToSend + sendDataIndex);
    tq.addTaskEntry(currentSend);
    sendDataIndex += amountToSend;
  }
  sendDataIndex = 0;
  tq.addTaskEntry(new String("EOM").getBytes());
  BluetoothLeService.taskQueue.add(tq);
  intent.putExtra("id", tq.getId());
  sendBroadcast(intent);
  mBluetoothLeService.send(characteristic)
}
```

Figure E.3: BTLE subject case for constructing and sending the BTLE characteristic to the send function

Figure E.3 shows a basic example of constructing and sending information to the underlying BTLE send function in Android/Java.

In all three figures, the underlying task is some minute element of BTLE on Android/Java. The goal is to replicate this functionality on iOS/Swift such that we use the same protocol and characteristic-style of interacting with the protocol. These code blocks are tailored versions of a resource sharing project worked on by members of the team. All code represents actual use cases in real cross-platform applications.

# Appendix F

# Subject Cases: WiFi

In this appendix we display the two WiFi subject cases used. Similar to the previous BTLE subject cases, these are examples of real cross-platform applications attempting to replicate functionality in Android and iOS with Java and Swift. The goal is to establish a WiFi Direct connection to another nearby device and initiate a data transmission. The two examples used are the initialization and reset functions. Figure F.1 demonstrates the initialization while Figure F.2 displays the WiFi reset and its appropriate functionality.

```
/*
 * Initialization for all wifi-direct functionality Set global filename
 * variable to file for transfer and call method
 */
public void wifiDirectInitialization() {
  manager = (WifiP2pManager) getSystemService(Context.WIFI_P2P_SERVICE);
  channel = manager.initialize(this, getMainLooper(), null);

  final IntentFilter intentFilter = new IntentFilter();
  intentFilter.addAction(WifiP2pManager.WIFI_P2P_STATE_CHANGED_ACTION);
  intentFilter.addAction(WifiP2pManager.WIFI_P2P_PEERS_CHANGED_ACTION);
  intentFilter.addAction(WifiP2pManager.WIFI_P2P_CONNECTION_CHANGED_ACTION);
  intentFilter.addAction(WifiP2pManager.WIFI_P2P_THIS_DEVICE_CHANGED_ACTION);
  registerReceiver(wifiDirectReceiver, intentFilter);

  manager.discoverPeers(channel, new WifiP2pManager.ActionListener() {

    @Override
    public void onSuccess() {
      System.out.println("WIFI DIRECT DISCOVERY INITIATED");
    }

    @Override
    public void onFailure(int reasonCode) {
      System.out.println("WIFI DIRECT DISCOVERY FAILED: " + reasonCode);
    }
  });
}
```

Figure F.1: WiFi subject case for initializing the WiFi Direct connection

```
/**
 * Reset wifi service.
 */
public static void resetWiFi() {
  WifiManager wifiManager = (WifiManager) context.getSystemService(
       Context.WIFI_SERVICE);
  wifiManager.setWifiEnabled(false);
  try {
    Thread.sleep(1000);
  } catch (InterruptedException e) {
    e.printStackTrace();
  }
  wifiManager.setWifiEnabled(true);
}
```

Figure F.2: WiFi subject case for resetting the WiFi Direct connection

# Appendix G

# Subject Cases: Data Structures-String

This appendix shows the String API of the data structures category seen throughout the evaluation. Many of these cases present aspects of the Java programming language that are not commonly accessed and thus could be less useful than the above sensors in terms of application to cross-platform mobile software developers.

```
private static Object processParameters(String paramType) {
  // TODO Auto-generated method stub
  Random rand = new Random();
  if (paramType.contains("String")) {
    return "test";
  } else if (paramType.contains("int")) {
    return rand.nextInt(2);
  } else if (paramType.contains("boolean")) {
    return true;
  } else if (paramType.contains("long") || paramType.contains("float") ||
      paramType.contains("double")) {
    return 0.0;
  }
  return "This is a test for native2native.";
}
```

Figure G.1: Data Structures subject case for processing parameters

```
/*
 * Compares this string to the specified object. The result is true if and
 * only if the argument is not null and is a String object that represents
 * the same sequence of characters as this object.
 */
public static boolean equals_Wrapper(String A) {
  System.out.println("equals_Wrapper testing");
  String equalsString = "";
  return equalsString.equals(A);
}
```

Figure G.2: Data Structures subject case for equals wrapper

```
/*
 * This object (which is already a string!) is itself returned.
 */
public static String toString_Wrapper(String A) {
  System.out.println("toString_Wrapper testing");
  String toString = A;
  return toString.toString();
}
```

Figure G.3: Data Structures subject case for toString wrapper

```
/*
 * Compares two strings lexicographically. The comparison is based on the
 * Unicode value of each character in the strings. The character sequence
 * represented by this String object is compared lexicographically to the
 * character sequence represented by the argument string. The result is a
 * negative integer if this String object lexicographically precedes the
 * argument string. The result is a positive integer if this String object
 * lexicographically follows the argument string. The result is zero if the
 * strings are equal; compareTo returns 0 exactly when the equals(Object)
 * method would return true.
 */
public static int compareTo_Wrapper(String A, String B) {
  System.out.println("compareTo_Wrapper testing");
  String compareToString = A;
  return compareToString.compareTo(B);
}
```

Figure G.4: Data Structures subject case for compareTo wrapper

```
/*
 * Returns the index within this string of the first occurrence of the
 * specified substring, starting at the specified index.
 */
public static int indexOf_Wrapper(String A, String B, Integer C) {
  System.out.println("indexOfStartingAt_Wrapper testing");
  String indexOfStartingAtString = A;
  return indexOfStartingAtString.indexOf(B, C);
}
```

Figure G.5: Data Structures subject case for indexOf wrapper

```
/*
 * Returns the index within this string of the first occurrence of the
 * specified substring.
 */
public static int indexOf_Wrapper(String A, String B) {
  System.out.println("indexOfSubstring_Wrapper testing");
  String indexOfSubstringString = A;
  return indexOfSubstringString.indexOf(B);
}
```

Figure G.6: Data Structures subject case for indexOf - multiple parameters wrapper

```
/*
 * Returns the string representation of the String argument.
 */
public static String valueOf_Wrapper(String A) {
  System.out.println("valueOfString_Wrapper testing");
  String valueOfString = A;
  return String.valueOf(valueOfString);
}
```

Figure G.7: Data Structures subject case for valueOf-string wrapper

```
/*
 * Returns the string representation of the Boolean argument.
 */
public static String valueOf_Wrapper(Boolean A) {
  System.out.println("valueOfBoolean_Wrapper testing");
  Boolean valueOfBoolean = A;
  return String.valueOf(valueOfBoolean);
}
```

Figure G.8: Data Structures subject case for valueOf-boolean wrapper

```
/*
 * Returns the string representation of the Integer argument.
 */
public static String valueOf_Wrapper(Integer A) {
  System.out.println("valueOfInteger_Wrapper testing");
  Integer valueOfInteger_Wrapper = A;
  return String.valueOf(valueOfInteger_Wrapper);
}
```

Figure G.9: Data Structures subject case for valueOf-integer wrapper

```
/*
 * Returns the string representation of the Double argument.
 */
public static String valueOf_Wrapper(Double A) {
  System.out.println("valueOfDouble_Wrapper testing");
  Double valueOfDouble_Wrapper = A;
  return String.valueOf(valueOfDouble_Wrapper);
}
```

Figure G.10: Data Structures subject case for valueOf-double wrapper

```
/*
 * Returns the length of this string. The length is equal to the number of
 * Unicode code units in the string.
 */
public static int length_Wrapper(String A) {
  System.out.println("length_Wrapper testing");
  String lengthString = A;
  return lengthString.length();
}
```

Figure G.11: Data Structures subject case for length wrapper

```
/*
 * Returns true if, and only if, length() is 0.
 */
public static boolean isEmpty_Wrapper(String A) {
  System.out.println("isEmpty_Wrapper testing");
  String isEmptyString = A;
  return isEmptyString.isEmpty();
}
```

Figure G.12: Data Structures subject case for isEmpty wrapper

```
/*
 * Returns the char value at the specified index. An index ranges from 0 to
 * length() - 1. The first char value of the sequence is at index 0, the
 * next at index 1, and so on, as for array indexing.
 */
public static char charAt_Wrapper(String A, Integer B) {
  System.out.println("charAt_Wrapper testing");
  String charAtString = A;
  return charAtString.charAt(B);
}
```

Figure G.13: Data Structures subject case for charAt wrapper

```
/*
 * Compares this String to another String, ignoring case considerations. Two
 * strings are considered equal ignoring case if they are of the same length
 * and corresponding characters in the two strings are equal ignoring case.
 */
public static boolean equalsIgnoreCase_Wrapper(String A, String B) {
  System.out.println("equalsIgnoreCase_Wrapper testing");
  String equalsIgnoreCaseString = A;
  return equalsIgnoreCaseString.equalsIgnoreCase(B);
}
```

Figure G.14: Data Structures subject case for equalsIgnoreCase wrapper

```
/*
 * Compares two strings lexicographically, ignoring case differences. This
 * method returns an integer whose sign is that of calling compareTo with
 * normalized versions of the strings where case differences have been
 * eliminated by calling
 * Character.toLowerCase(Character.toUpperCase(character)) on each
 * character.
 */
public static int compareToIgnoreCase_Wrapper(String A, String B) {
  System.out.println("compareToIgnoreCase_Wrapper testing");
  String compareToIgnoreCaseString = A;
  return compareToIgnoreCaseString.compareToIgnoreCase(B) > 0 ? 1 : 0;
}
```

Figure G.15: Data Structures subject case for compareToIgnoreCase wrapper

```
/*
 * Tests if the substring of this string beginning at the specified index
 * starts with the specified prefix.
 */
public static boolean startsWith_Wrapper(String A, String B, Integer C) {
  System.out.println("startsWith_index_Wrapper testing");
  String startsWithIndexString = A;
  return startsWithIndexString.startsWith(B, C);
}
```

Figure G.16: Data Structures subject case for startsWith Wrapper

```
/*
 * Tests if this string starts with the specified prefix.
 */
public static boolean startsWith_Wrapper(String A, String B) {
  System.out.println("startsWith_Wrapper testing");
  String startsWithString = A;
  return startsWithString.startsWith(B);
}
```

Figure G.17: Data Structures subject case for startsWith-multiple parameters Wrapper

```
/*
 * Tests if this string ends with the specified suffix.
 */
public static boolean endsWith_Wrapper(String A, String B) {
  System.out.println("endsWith_Wrapper testing");
  String endsWithString = A;
  return endsWithString.endsWith(B);
}
```

Figure G.18: Data Structures subject case for endsWith Wrapper

```
/*
 * Returns a new string that is a substring of this string. The substring
 * begins at the specified beginIndex and extends to the character at index
 * endIndex - 1. Thus the length of the substring is endIndex-beginIndex.
 */
public static String substring_Wrapper(String A, Integer B, Integer C) {
  System.out.println("substring_begins_ends_Wrapper testing");
  String substringBeginsEndsString = A;
  return substringBeginsEndsString.substring(B, C);
}
```

Figure G.19: Data Structures subject case for substring Wrapper

```
/*
 * Returns a new string that is a substring of this string. The substring
 * begins with the character at the specified index and extends to the end
 * of this string.
 */
public static String substring_Wrapper(String A, Integer B) {
  System.out.println("substring_begins_Wrapper testing");
  String substringBeginsString = A;
  return substringBeginsString.substring(B);
}
```

Figure G.20: Data Structures subject case for substring-multiple parameters Wrapper

```
/*
 * Concatenates the specified string to the end of this string.
 *
 * If the length of the argument string is 0, then this String object is
 * returned. Otherwise, a new String object is created, representing a
 * character sequence that is the concatenation of the character sequence
 * represented by this String object and the character sequence represented
 * by the argument string.
 */
public static String concat_Wrapper(String A, String B) {
  System.out.println("concat_Wrapper testing");
  String concatString = A;
  return concatString.concat(B);
}
```

Figure G.21: Data Structures subject case for concat Wrapper

```
/*
 * Replaces the first substring of this string that matches the given
 * regular expression with the given replacement.
 */
public static String replaceFirst_Wrapper(String A, String B, String C) {
  System.out.println("replaceFirst_Wrapper testing");
  String replaceFirstString = A;
  return replaceFirstString.replaceFirst(B, C);
}
```

Figure G.22: Data Structures subject case for replaceFirst Wrapper

```
/*
 * Replaces each substring of this string that matches the given regular
 * expression with the given replacement.
 */
public static String replaceAll_Wrapper(String A, String B, String C) {
  System.out.println("replaceAll_Wrapper testing");
  String replaceAllString = A;
  return replaceAllString.replaceAll(B, C);
}
```

Figure G.23: Data Structures subject case for replaceAll Wrapper

```
/*
 * Converts all of the characters in this String to lower case using the
 * rules of the default locale. This is equivalent to calling
 * toLowerCase(Locale.getDefault()).
 */
public static String toLowerCase_Wrapper(String A) {
  System.out.println("toLowerCase_Wrapper testing");
  String toLowerCaseString = A;
  return toLowerCaseString.toUpperCase();
}
```

Figure G.24: Data Structures subject case for toLowerCase Wrapper

```
/*
 * Converts all of the characters in this String to upper case using the
 * rules of the default locale. This method is equivalent to
 * toUpperCase(Locale.getDefault()).
 */
public static String toUpperCase_Wrapper(String A) {
  System.out.println("toUpperCase_Wrapper testing");
  String toUpperCaseString = A;
  return toUpperCaseString.toUpperCase();
}
```

Figure G.25: Data Structures subject case for toUpperCase Wrapper

```
/*
 * Returns a copy of the string, with leading and trailing whitespace
 * omitted.
 */
public static String trim_Wrapper(String A) {
  System.out.println("trim_Wrapper testing");
  String trimString = A;
  return trimString.trim();
}
```

Figure G.26: Data Structures subject case for trim Wrapper

# Appendix H

# Subject Cases: Data Structures-ArrayList

This appendix shows the subject test cases for ArrayList from the category of Data Structures APIs.

```
/*
 * Inserts the specified element at the specified position in this list.
 * Shifts the element currently at that position (if any) and any subsequent
 * elements to the right (adds one to their indices).
 */
public static ArrayList<String> add_Wrapper(ArrayList<String> A,
    Integer B, String C) {
  System.out.println("addToIndex_Wrapper testing");
  ArrayList<String> addToIndexArrayList = A;
  addToIndexArrayList.add(B, C);
  return addToIndexArrayList;
}
```

Figure H.1: Data Structures subject case for add Wrapper

```
/*
 * Appends the specified element to the end of this list.
 */
public static ArrayList<String> add_Wrapper(ArrayList<String> A, String B) {
  System.out.println("addToEnd_Wrapper testing");
  ArrayList<String> addToEndArrayList = A;
  addToEndArrayList.add(B);
  return addToEndArrayList;
}
```

Figure H.2: Data Structures subject case for add-list Wrapper

```
/*
 * Removes the element at the specified position in this list. Shifts any
 * subsequent elements to the left (subtracts one from their indices).
 */
public static ArrayList<String> remove_Wrapper(ArrayList<String> A, Integer B) {
  System.out.println("removeAtIndex_Wrapper testing");
  ArrayList<String> removeAtIndexArrayList = A;
  removeAtIndexArrayList.remove(B.intValue());
  return removeAtIndexArrayList;
}
```

Figure H.3: Data Structures subject case for remove Wrapper

```
/*
 * Removes the first occurrence of the specified element from this list, if
 * it is present.
 */
public static ArrayList<String> remove_Wrapper(ArrayList<String> A, String B) {
  System.out.println("remove_Wrapper testing");
  ArrayList<String> removeArrayList = A;
  removeArrayList.remove(B);
  return removeArrayList;
}
```

Figure H.4: Data Structures subject case for remove-list Wrapper

```
/*
 * Returns the element at the specified position in this list.
 */
public static String get_Wrapper(ArrayList<String> A, Integer B) {
  System.out.println("get_Wrapper testing");
  ArrayList<String> getArrayList = A;
  return getArrayList.get(B);
}
```

Figure H.5: Data Structures subject case for get Wrapper

```
/*
 * Returns a shallow copy of this ArrayList instance. (The elements
 * themselves are not copied.)
 */
public static ArrayList<String> clone_Wrapper(ArrayList<String> A) {
  System.out.println("clone_Wrapper testing");
  ArrayList<String> cloneArrayList = A;
  cloneArrayList.clone();
  return cloneArrayList;
}
```

Figure H.6: Data Structures subject case for clone Wrapper

```
/*
 * Returns the index of the first occurrence of the specified element in
 * this list, or -1 if this list does not contain the element.
 */
public static Integer indexOf_Wrapper(ArrayList<String> A, String B) {
  System.out.println("indexOf_Wrapper testing");
  ArrayList<String> indexOfArrayList = A;
  return indexOfArrayList.indexOf(B);
}
```

Figure H.7: Data Structures subject case for indexOf Wrapper

```
/*
 * Removes all of the elements from this list. The list will be empty after
 * this call returns.
 */
public static ArrayList<String> clear_Wrapper(ArrayList<String> A) {
  System.out.println("clear_Wrapper testing");
  ArrayList<String> clearArrayList = A;
  clearArrayList.clear();
  return clearArrayList;
}
```

Figure H.8: Data Structures subject case for clear Wrapper

```
/*
 * Returns true if this list contains no elements.
 */
public static boolean isEmpty_Wrapper(ArrayList<String> A) {
  System.out.println("isEmpty_Wrapper testing");
  ArrayList<String> isEmptyArrayList = A;
  return isEmptyArrayList.isEmpty();
}
```

Figure H.9: Data Structures subject case for isEmpty Wrapper

```
/*
 * Returns the index of the last occurrence of the specified element in this
 * list, or -1 if this list does not contain the element. More formally,
 * returns the highest index i such that (o==null ? get(i)==null :
 * o.equals(get(i))), or -1 if there is no such index.
 */
public static Integer lastIndexOf_Wrapper(ArrayList<String> A, String B) {
  System.out.println("lastIndexOf_Wrapper testing");
  ArrayList<String> lastIndexOfArrayList = A;
  return lastIndexOfArrayList.lastIndexOf(B);
}
```

Figure H.10: Data Structures subject case for lastIndexOf Wrapper

```
/*
 * Returns true if this list contains the specified element
 */
public static Boolean contains_Wrapper(ArrayList<String> A, String B) {
  System.out.println("contains_Wrapper testing");
  ArrayList<String> containsArrayList = A;
  return containsArrayList.contains(B);
}
```

Figure H.11: Data Structures subject case for contains Wrapper

```
/*
 * Returns the number of elements in this list .
 */
public static Integer size_Wrapper(ArrayList<String> A) {
  System.out.println("size_Wrapper testing");
  ArrayList<String> sizeArrayList = A;
  return sizeArrayList.size();
}
```

Figure H.12: Data Structures subject case for size Wrapper

```
/*
 * Returns a view of the portion of this list between the specified
 * fromIndex, inclusive , and toIndex, exclusive . (If fromIndex and toIndex
 * are equal, the returned list is empty.) The returned list is backed by
 * this list , so non-structural changes in the returned list are reflected
 * in this list , and vice-versa. The returned list supports all of the
 * optional list operations .
 */
public static List<String> subList_Wrapper(ArrayList<String> A, Integer B, Integer C) {
  System.out.println("subList_Wrapper testing");
  ArrayList<String> subListArrayList = A;
  return subListArrayList.subList(B, C);
}
```

Figure H.13: Data Structures subject case for subList Wrapper

```
/*
 * Appends all of the elements in the specified collection to the end of
 * this list, in the order that they are returned by the specified
 * collection's Iterator. The behavior of this operation is undefined if the
 * specified collection is modified while the operation is in progress.
 * (This implies that the behavior of this call is undefined if the
 * specified collection is this list, and this list is nonempty.)
 */
public static boolean addAll_Wrapper(ArrayList<String> A, ArrayList<String> B) {
  System.out.println("addAll_Wrapper testing");
  ArrayList<String> addAllArrayList = A;
  return addAllArrayList.addAll(B);
}
```

Figure H.14: Data Structures subject case for addAll Wrapper

```
/*
 * nserts all of the elements in the specified collection into this list,
 * starting at the specified position. Shifts the element currently at that
 * position (if any) and any subsequent elements to the right (increases
 * their indices). The new elements will appear in the list in the order
 * that they are returned by the specified collection's iterator.
 */
public static boolean addAll_Wrapper(ArrayList<String> A, Integer B, ArrayList<String> C) {
  System.out.println("addAllAtIndex_Wrapper testing");
  ArrayList<String> addAllAtIndexArrayList = A;
  return addAllAtIndexArrayList.addAll(B, C);
}
```

Figure H.15: Data Structures subject case for addAll-multiple parameters Wrapper

```
/*
 * Replaces the element at the specified position in this list with the
 * specified element.
 */
public static ArrayList<String> set_Wrapper(ArrayList<String> A, Integer B, String C) {
  System.out.println("set_Wrapper testing");
  ArrayList<String> setArrayList = A;
  setArrayList.set(B, C);
  return setArrayList;
}
```

Figure H.16: Data Structures subject case for set Wrapper

```
/*
 * Increases the capacity of this ArrayList instance, if necessary, to
 * ensure that it can hold at least the number of elements specified by the
 * minimum capacity argument.
 */
public static ArrayList<String> ensureCapacity_Wrapper(ArrayList<String> A, Integer B) {
  System.out.println("ensureCapacity_Wrapper testing");
  ArrayList<String> ensureCapacityArrayList = A;
  ensureCapacityArrayList.ensureCapacity(B);
  return ensureCapacityArrayList;
}
```

Figure H.17: Data Structures subject case for ensureCapacity Wrapper

```
/*
 * Trims the capacity of this ArrayList instance to be the list's current
 * size. An application can use this operation to minimize the storage of an
 * ArrayList instance.
 */
public static ArrayList<String> trimToSize_Wrapper(ArrayList<String> A) {
  System.out.println("trimToSize_Wrapper testing");
  ArrayList<String> trimToSizeArrayList = A;
  trimToSizeArrayList.trimToSize();
  return trimToSizeArrayList;
}
```

Figure H.18: Data Structures subject case for trimToSize Wrapper

```
/*
 * Removes from this list all of its elements that are contained in the
 * specified collection.
 */
public static ArrayList<String> removeAll_Wrapper(ArrayList<String> A, ArrayList<String> B) {
  System.out.println("removeAll_Wrapper testing");
  ArrayList<String> removeAllArrayList = A;
  removeAllArrayList.removeAll(B);
  return removeAllArrayList;
}
```

Figure H.19: Data Structures subject case for removeAll Wrapper

```
/*
 * Retains only the elements in this list that are contained in the
 * specified collection. In other words, removes from this list all of its
 * elements that are not contained in the specified collection.
 */
public static ArrayList<String> retainAll_Wrapper(ArrayList<String> A, ArrayList<String> B) {
  System.out.println("retainAll_Wrapper testing");
  ArrayList<String> retainAllArrayList = A;
  retainAllArrayList.retainAll(B);
  return retainAllArrayList;
}
```

Figure H.20: Data Structures subject case for retainAll Wrapper

```
/*
 * Returns a list iterator over the elements in this list (in proper
 * sequence), starting at the specified position in the list
 */
public static ArrayList<String> listIterator_Wrapper(ArrayList<String> A, Integer B) {
  System.out.println("listIteratorFromIndex_Wrapper testing");
  ArrayList<String> listIteratorFromIndexArrayList = A;
  listIteratorFromIndexArrayList.listIterator(B);
  return listIteratorFromIndexArrayList;
}
```

Figure H.21: Data Structures subject case for listIterator Wrapper

```
/*
 * Returns a list iterator over the elements in this list (in proper
 * sequence).
 */
public static ArrayList<String> listIterator_Wrapper(ArrayList<String> A) {
  System.out.println("listIterator_Wrapper testing");
  ArrayList<String> listIteratorArrayList = A;
  listIteratorArrayList.listIterator();
  return listIteratorArrayList;
}
```

Figure H.22: Data Structures subject case for listIterator-single parameter Wrapper

# Appendix I

# Subject Cases: Data Structures-HashMap

This appendix shows the subject test cases for HashMap from the category of Data Structures APIs.

```
/*
 * Removes the mapping for the specified key from this map if present.
 */
public static HashMap<String, String> remove_Wrapper(HashMap<String, String> A, String B) {
  System.out.println("remove_Wrapper testing");
  HashMap<String, String> removeHashMap = A;
  removeHashMap.remove(B);
  return removeHashMap;
}
```

Figure I.1: Data Structures subject case for remove Wrapper

```
/*
 * Returns the value to which the specified key is mapped, or null if this
 * map contains no mapping for the key.
 */
public static HashMap<String, String> get_Wrapper(HashMap<String, String> A, String B) {
  System.out.println("get_Wrapper testing");
  HashMap<String, String> getHashMap = A;
  getHashMap.get(B);
  return getHashMap;
}
```

Figure I.2: Data Structures subject case for get Wrapper

```
/*
 * Associates the specified value with the specified key in this map. If the
 * map previously contained a mapping for the key, the old value is
 * replaced.
 */
public static HashMap<String, String> put_Wrapper(HashMap<String, String> A, String B, String C) {
  System.out.println("put_Wrapper testing");
  HashMap<String, String> putHashMap = A;
  putHashMap.put(B, C);
  return putHashMap;
}
```

Figure I.3: Data Structures subject case for put Wrapper

```
/*
 * Returns a Collection view of the values contained in this map. The
 * collection is backed by the map, so changes to the map are reflected in
 * the collection, and vice-versa. If the map is modified while an iteration
 * over the collection is in progress (except through the iterator's own
 * remove operation), the results of the iteration are undefined. The
 * collection supports element removal, which removes the corresponding
 * mapping from the map, via the Iterator.remove, Collection.remove,
 * removeAll, retainAll and clear operations. It does not support the add or
 * addAll operations
 */
public static HashMap<String, String> values_Wrapper(HashMap<String, String> A) {
  System.out.println("values_Wrapper testing");
  HashMap<String, String> valuesHashMap = A;
  valuesHashMap.values();
  return valuesHashMap;
}
```

Figure I.4: Data Structures subject case for values Wrapper

```
/*
 * Returns a shallow copy of this HashMap instance: the keys and values
 * themselves are not cloned.
 */
public static HashMap<String, String> clone_Wrapper(HashMap<String, String> A) {
  System.out.println("clone_Wrapper testing");
  HashMap<String, String> cloneHashMap = A;
  cloneHashMap.clone();
  return cloneHashMap;
}
```

Figure I.5: Data Structures subject case for clone Wrapper

```
/*
 * Removes all of the mappings from this map. The map will be empty after
 * this call returns.
 */
public static HashMap<String, String> clear_Wrapper(HashMap<String, String> A) {
  System.out.println("clear_Wrapper testing");
  HashMap<String, String> clearHashMap = A;
  clearHashMap.clear();
  return clearHashMap;
}
```

Figure I.6: Data Structures subject case for clear Wrapper

```
/*
 * Returns true if this map contains no key-value mappings.
 */
public static HashMap<String, String> isEmpty_Wrapper(HashMap<String, String> A) {
  System.out.println("isEmpty_Wrapper testing");
  HashMap<String, String> isEmptyHashMap = A;
  isEmptyHashMap.isEmpty();
  return isEmptyHashMap;
}
```

Figure I.7: Data Structures subject case for isEmpty Wrapper

```
/*
 * Returns the number of key-value mappings in this map.
 */
public static HashMap<String, String> size_Wrapper(HashMap<String, String> A) {
  System.out.println("size_Wrapper testing");
  HashMap<String, String> sizeHashMap = A;
  sizeHashMap.size();
  return sizeHashMap;
}
```

Figure I.8: Data Structures subject case for size Wrapper

```
/*
 * Returns a Set view of the mappings contained in this map.
 */
public static HashMap<String, String> entrySet_Wrapper(HashMap<String, String> A) {
  System.out.println("entrySet_Wrapper testing");
  HashMap<String, String> entrySetHashMap = A;
  entrySetHashMap.entrySet();
  return entrySetHashMap;
}
```

Figure I.9: Data Structures subject case for entrySet Wrapper

```
/*
 * Copies all of the mappings from the specified map to this map. These
 * mappings will replace any mappings that this map had for any of the keys
 * currently in the specified map.
 */
public static HashMap<String, String> putAll_Wrapper(HashMap<String, String> A,
    Map<String, String> B) {
  System.out.println("putAll_Wrapper testing");
  HashMap<String, String> putAllHashMap = A;
  putAllHashMap.putAll(B);
  return putAllHashMap;
}
```

Figure I.10: Data Structures subject case for putAll Wrapper

```
/*
 * Returns a Set view of the keys contained in this map.
 */
public static HashMap<String, String> keySet_Wrapper(HashMap<String, String> A) {
  System.out.println("keySet_Wrapper testing");
  HashMap<String, String> keySetHashMap = A;
  keySetHashMap.keySet();
  return keySetHashMap;
}
```

Figure I.11: Data Structures subject case for keySet Wrapper

```
/*
 * Returns true if this map maps one or more keys to the specified value.
 */
public static HashMap<String, String> containsValue_Wrapper(HashMap<String,
    String> A, String B) {
  System.out.println("containsValue_Wrapper testing");
  HashMap<String, String> containsValueHashMap = A;
  containsValueHashMap.containsValue(B);
  return containsValueHashMap;
}
```

Figure I.12: Data Structures subject case for containsValue Wrapper

```
/*
 * Returns true if this map contains a mapping for the specified key.
 */
public static HashMap<String, String> containsKey_Wrapper(HashMap<String,
    String> A, String B) {
  System.out.println("containsKey_Wrapper testing");
  HashMap<String, String> containsKeyHashMap = A;
  containsKeyHashMap.containsKey(B);
  return containsKeyHashMap;
}
```

Figure I.13: Data Structures subject case for containsKey Wrapper

# Appendix J

# Case-by-Case Results

This appendix displays the individual case-by-case results for the various data structure APIs.

| API | Result-Rank 1 | Result-Rank 2 |
|---|---|---|
| equals_Wrapper("") | yes | 0.5 |
| toString_Wrapper("") | no | yes |
| compareTo_Wrapper("") | yes | yes |
| indexOf_Wrapper("", 0) | yes | yes |
| indexOf_Wrapper("") | yes | no |
| valueOf_Wrapper(true) | yes | no |
| valueOf_Wrapper(0.0) | yes | no |
| valueOf_Wrapper(0) | yes | yes |
| valueOf_Wrapper("") | yes | 0.5 |
| length_Wrapper() | no | yes |
| isEmpty_Wrapper() | yes | 0.5 |
| charAt_Wrapper(0) | yes | yes |
| equalsIgnoreCase_Wrapper("") | no | yes |
| compareToIgnoreCase_Wrapper("") | no | 0.5 |
| startsWith_Wrapper("") | yes | 0.5 |
| startsWith_Wrapper("", 0) | yes | 0.5 |
| endsWith_Wrapper("") | yes | 0.5 |
| substring_Wrapper(0) | yes | yes |
| substring_Wrapper(0, 0) | yes | yes |
| concat_Wrapper("") | yes | yes |
| replaceAll_Wrapper("", "") | yes | 0.5 |
| replaceFirst_Wrapper("", "") | 0.5 | no |
| toLowerCase_Wrapper() | yes | yes |
| toUpperCase_Wrapper() | yes | yes |
| trim_Wrapper() | 0.5 | NaCB |

Table J.1: String API Results

| API | Result-Rank 1 | Result-Rank 2 |
|---|---|---|
| add_wrapper | yes | NaCB |
| add_wrapper(list) | yes | NaCB |
| remove_wrapper() | yes | NaCB |
| remove_wrapper(list) | yes | NaCB |
| get_wrapper | yes | no |
| clone_wrapper | yes | NaCB |
| index_wrapper | yes | yes |
| clear_wrapper | 0.5 | NaCB |
| is_empty_wrapper | NaCB | yes |
| last_index_wrapper | yes | yes |
| contains_wrapper | yes | yes |
| size_wrapper | yes | yes |
| sublist_wrapper | no | no |
| add_all_wrapper | no | no |
| add_all_wrapper(list) | no | no |
| set_wrapper | no | yes |
| ensure_capacity | 0.5 | NaCB |
| trim_to_size | no | 0.5 |
| remove_all_wrapper | yes | NaCB |
| retain_all_wrapper | 0.5 | no |
| list_iterator_wrapper | yes | NaCB |
| list_iterator_wrapper(list) | yes | NaCB |

Table J.2: ArrayList API Results

| API | Result-Rank 1 | Result-Rank 2 |
|---|---|---|
| remove_wrapper | yes | no |
| get_wrapper | yes | NaCB |
| put_wrapper | yes | yes |
| values_wrapper | yes | 0.5 |
| clone_wrapper | yes | no |
| clear_wrapper | yes | NaCB |
| is_empty_wrapper | yes | 0.5 |
| size_wrapper | yes | 0.5 |
| entrySet_Wrapper | yes | NaCB |
| putAll_Wrapper | 0.5 | NaCB |
| keySet_Wrapper | 0.5 | yes |
| containsValue_Wrapper | yes | yes |
| containsKey_Wrapper | 0.5 | NaCB |

Table J.3: HashMap API Results

# Appendix K

# Supplemental Synthesized Code Results

This appendix displays some special supplemental synthesized code examples for additional location queries, facial recognition, and HTTP request handling.

```swift
// Auto-generated Swift file from Native-2-Native

/**
* Gets the location from the device's GPS sensor.
* @param context The context of the application
* @return String GPS location
*/
func getLocation(context: /*Unknown token: Context*/) -> /*Unknown token:
    Location*/ {
    // Insert Native-2-Native Result
    return /*location*/
}

/* NATIVE-2-NATIVE RESULT
override func viewDidLoad() {
    super.viewDidLoad()
    // Do any additional setup after loading the view, typically from a nib.
    locationManager = CLLocationManager()
    locationManager.delegate = self
    locationManager.desiredAccuracy = kCLLocationAccuracyBest
    locationManager.requestAlwaysAuthorization()
    locationManager.startUpdatingLocation()
}
*/
```

Figure K.1: Supplemental Location Code Synthesis

```swift
// Auto-generated Swift file from Native-2-Native

/*
* Detects the number of faces from a given image URL.
* @param url The URL of the image to detect faces in
* @return Int The number of faces detected in the image.
*/
func getFaces(url: NSString) -> Int {
    // Insert Native-2-Native Result

    if (/*faces.size() > 0*/) {
        for (var i = 0; i < faces.size(); i++) {

        }
    }

    return /*numFaces*/
}

/* NATIVE-2-NATIVE RESULT
var myImage:CIImage = CIImage(contentsOfURL: url!)
var context:CIContext = CIContext(options: nil)
var opts:NSDictionary = [CIDetectorAccuracy : CIDetectorAccuracyHigh]
var detector:CIDetector = CIDetector(ofType: CIDetectorTypeFace, context:
    context, options: opts as [NSObject : AnyObject])
var features:NSArray = detector.featuresInImage(myImage, options: opts as
    [NSObject : AnyObject])
var result = features.count
*/
```

Figure K.2: Supplemental Facial Recognition Code Synthesis

```
// Auto-generated Swift file from Native-2-Native

/**
* Performs an HTTP request with the given url.
* @param url The url of the website
* @return String The String representation of the processed URL.
*/
func getHTTP(url:NSString) -> NSString {
    // Insert Native-2-Native Result
    while (/*(line = reader.readLine()) != null*/) {

    }
    return /*resString*/
}


/* NATIVE-2-NATIVE RESULT
let url = NSURL(NString: url)

let task = NSURLSession.sharedSession().dataTaskWithURL(url!) {(data,
    response, error) in
println(NSString(data: data, encoding: NSUTF8StringEncoding))
}

task.resume()
*/
```

Figure K.3: Supplemental HTTP Code Synthesis