RECOMENDATION REPORT

To: Manager
From: Chance Daniel
Date: 29 October 2015
Subject: Recommendation Report on the Effectiveness of Swift

Preface

After your approval of my research into the effectiveness of Swift at CompanyX, I began investigating the important differences between Swift and Objective-C, the benefits of using Swift, as well as the drawbacks that come with switching to the language. The following report is the information I found during my studies, as well as my final recommendation for whether or not Swift is ready for CompanyX to adopt it as our primary programming language.

My Background

As an intern, I have the unique perspective of coming to CompanyX with a fresh outlook on the way CompanyX innovates. Unlike most of the other engineers, I am not heavily invested in large existing code bases that are already written in Objective-C. During my last internship, instead of using Objective-C at the start of my project, I opted to use Swift. This allowed me to see what it would be like to create something at CompanyX from scratch using Swift. Once I got deeper into my project, I found that my code was not yet compatible with the existing code that I needed to integrate with, and ultimately I rewrote my code in Objective-C. This is what led me to want to investigate if Swift is ready to replace Objective-C and what it would take to execute the switch. The following information is based upon secondary source research as well as primary research in the form of coding samples produced from my own experimentation; Figures 1 and Figure 2 at the end of the document represent original code samples of both Swift and Objective C.

Introduction to Swift

Swift is an **Object Oriented Programming**[1] (OOP), a programming model that uses objects, language that first appeared to the public in June of 2014 during our annual World Wide Developer Conference (WWDC). Since then the language has moved through several versions and is currently at the latest stable version of 2.1 which was released in October of this year. Apple pitched Swift to developers as "a modern programming language that is safe, fast ,and interactive," [2] and the adoption rate was even better than expected. Apple has also decided to open source the language, so hopefully it is only a matter of time before the industry also makes use of Swift.

Similarities Between Swift and Objective-C

Swift was designed with Objective-C in mind, in order to keep the language feeling familiar. Much of the **syntax**[3] of Swift is adapted from Objective-C and keeps "the power of Objective-C's dynamic object model,"[4] but modernizes the language making it type safe and easier to debug. If you look at Figure 1 and Figure 2, you can see how similar the languages appear.

Differences Between Swift and Objective-C

There are a great number of differences between Swift and its older brother. Perhaps the most notable is the type inference and type safety that Swift has. When programming in Swift it is not necessary to declare what type of variable you are using, and because the language performs a "static dispatch via

table look-up,"[5] the compiler can evaluate and know if there are errors in your code as soon as you write it. In Figure 1, the variable "groceryList" is declared with the keyword var and nothing else; this compared to Figure 2, where "groceryList" is declared as a @property with a type of NSArray. Swift eliminates the need for extra information in the declaration of variables. This allows you to write working code faster, compared to Objective-C code that may compile fine but crash due to a runtime error later.

Next, Swift allows variables to have optional values; an **optional value** means that the variable can have a value that is either "nil" or a valid value. Using optionals allows you to safeguard your code against unwanted errors and exceptions, since the compiler forces you to check for a valid value before using it. In Objective-C, these checks would be up to the programmer, and when not incorporated can cause runtime errors unexpectedly.

Another great benefit of Swift is its improvements in readability over Objective-C. Most of the syntax is cleaner and simpler than its Objective-C counterpart. Without the need for brackets, semicolons, and other legacy conventions, Swift is easier to read, write, and debug. This is made apparent by the two coding samples; the code in Figure 1 is about half of the number of lines of the code in Figure 2 and much easier to read.

Finally, the last notable difference between the languages is the lack of required header files in Swift. A **header file** is used to expose implementation of the class to other developers without revealing source code; also anything the programmer wants **public** (available to other classes) is listed here. In Figure 2, the code in the section labeled @interface would also be the header file if the programmer wanted to expose the "groceryList" variable to other classes. In Swift however, header files are nonessential since they are generated internally to simplify coding for the programmer and to prevent the unnecessary task of duplicating **function** [6] and variable definitions.

Benefits of Switching

When I examine Swift as a language alone, not looking at the politics that would be involved in switching programming languages, transitioning from Objective-C is almost a no brainer. Apart from the benefit of being able to write less code in a easier and more readable way, Swift is also safer than Objective-C giving the programmer the benefit of having fewer bugs to fix. Using optionals mentioned above, the Swift "can generate a compiler error as you write bad code,"[7] meaning that problems can be identified and addressed as they are typed.

At CompanyX, software engineers write a tremendous amount of code, code that has to perform as expected on many devices of different generations, in different languages, and with different configurations. This means that the more bugs that can be prevented during the initial writing of software, the more efficient CompanyX will be, saving resources and money in the process.

In addition to the benefits for CompanyX and the software engineers, there are also quite a few benefits for the consumers. Swift code executes faster and is less memory intensive, "this is an important factor for any programming language that will be used for responsive graphics and user input, especially on a tactile device like the iPhone."[8] In order for CompanyX to continue to innovate, it is imperative that we stay on the bleeding edge of technology and adopting Swift as our primary language keeps us on that track.

The final benefit I want to mention is a new tool called Playground, which was released as a companion to Swift. **Playground**, " is a 'compiler-in-real- time' which executes the code immediately while the user develops,"[9] and allows a developer to write and test lines of code independent from an app. This means that a programmer can try out a new idea or block of code and get instant feedback as to whether it will work in a larger project. Developers can cut down on programming and testing cycles because instead of writing the code in an app, building the project and running it on a device, they can quickly prototype something in Playground and then drop it into the app when they are satisfied that it will work.

CompanyX's Difficult Position

Even though the benefits of Swift look like they far outweigh the drawbacks, there are other things to consider than just the changes of the language before switching. The majority of CompanyX's source code is written in Objective-C, that is millions, if not billions, of lines of code that span across all of CompanyX's software products. While Swift was built with Objective-C compatibility in mind, there is still a tremendous amount of code that would have to be re-written or adapted to work with newer projects written in Swift. On top of that, CompanyX software engineers have used Objective-C for decades, so they see it as a more powerful language because they know all the ins and outs. The engineers who actually know Swift, find that they are slower, and less efficient when using Swift over Objective-C because they have yet to learn the nuances of the language; therefore most likely would be a dip in productivity during a transition period from Objective-C to Swift.

Third-Party Developer Position

For most third-party developers, their apps are independent of each other; meaning that they can switch to Swift without having to deal with legacy code and backwards compatibility. With Swift being easier to learn than Objective-C, I suspect that we will see a shift toward the new language as time goes on. CompanyX needs to make use of these developers to lead the industry in the transition to Swift, since they have less ties to Objective-C and can be at the forefront of technological advancements with fewer drawbacks.

Final Recommendation

Overall, my recommendation is that CompanyX is not yet ready to adopt Swift as it's primary language. But we also should begin preparing for the change. Swift is going to be a powerful language in the coming years, with hopefully a rapid adoption rate; CompanyX needs to ensure that they are taking the steps necessary to be able to switch to Swift when the time comes. Swift is an effective language, it just is not compatible with a large organization with existing infrastructure at the moment. CompanyX is an ever changing company and Swift is the modern language that is perfectly suited for it, just not yet.

If you have any questions about my research or final recommendation, please let me know either in person or via email, as I would be happy to discuss any of my findings in further detail.

CC: none

[1]A programming model that uses objects and data instead of logic that takes an input and returns an output. Object constructs are used as blueprints to build objects from a data source. Generally used in development of software that will be displayed to a screen or interface.

[2] "Swift - Apple Developer." Swift. Apple, n.d. Web. 05 Nov. 2015.

[3] The styling and way that a programming language is written and rules that the programming language is able to interpret.

[4] "The Swift Programming Language (Swift 2.1): About Swift." The Swift Programming Language (Swift 2.1): About Swift. Apple, n.d. Web. 05 Nov. 2015.

[5] Malik, Waqar. "Hello Swift." Learn Swift on the Mac: For OS X and IOS. New York, NY: Apress, 2015. N. pag. Springer Link. Web.

[6]A set of instructions or procedures that can be called by other instructions in a program. Generally computes some data or performs some action on the objects represented by the software.

[7] Solt, Paul. "Swift vs. Objective-C: 10 Reasons the Future Favors Swift." InfoWorld. N.p., n.d. Web. 05 Nov. 2015.

[8] Solt.

[9] García, Cristian González, Jordán Pascual-Espada, Cristina Pelayo G-Bustelo, and Juan Manuel Cueva-Lovelle. "Swift vs. Objective-C: A New Programming Language." IJIMAI International Journal of Interactive Multimedia and Artificial Intelligence 3.3 (2015): 74. Web.

Figure 1: Swift Code Example

Variables are declared with implicit types.

```
import Foundation

class SwiftClass {
    var groceryList = ["Bananas", "Oranges", "Cereal", "Peanut Butter", "Milk", "Eggs", "Soda", "Pasta",
        "Yogurt", "Frozen Pizza", "Oatmeal", "Apples", "Swiss Cheese", "Bread"]

    init () {
        groceryList.sortInPlace()
        print(groceryList)
        print(numberOfWords())
        print((numberOfItems()))
    }

    func numberOfWords() -> Int{
        var count = 0
        for item in groceryList {
            let words = item.componentsSeparatedByCharactersInSet(NSCharacterSet.whitespaceCharacterSet())
            count += words.count
        }
        return count
    }

    func numberOfItems() -> Int {
        return groceryList.count
    }
}
```

Functions are using the keyword func.

Note that the Swift syntax allows for shorter, simpler code that is easer to both read and write than the Objective-C counterpart below.

Figure 2: Objective-C Code Sample


```objc
#import "ObjectiveCClass.h"

@interface ObjectiveCClass ()
@property(nonatomic, strong) NSArray *groceryList;
@end

@implementation ObjectiveCClass

- (instancetype)init
{
    self = [super init];
    if (self) {
        self.groceryList = @[@"Bananas", @"Oranges", @"Cereal", @"Peanut Butter", @"Milk", @"Eggs",
                             @"Soda", @"Pasta", @"Yogurt", @"Frozen Pizza", @"Oatmeal", @"Apples",
                             @"Swiss Cheese", @"Bread"];

        NSArray *sortedArray;
        sortedArray = [self.groceryList sortedArrayUsingComparator:^NSComparisonResult(id a, id b) {
            NSString *first = a;
            NSString *second = b;
            return [first compare:second];
        }];

        NSLog(@"%@",sortedArray);
        NSLog(@"%lu", (unsigned long)[self numberOfWords]);
        NSLog(@"%lu", (unsigned long)[self numberOfItems]);
    }
    return self;
}

- (NSUInteger)numberOfWords {
    __block NSUInteger count = 0;
    for (NSString *string in self.groceryList) {
        [string enumerateSubstringsInRange:NSMakeRange(0, string.length)
                                   options:NSStringEnumerationByWords
                                usingBlock:^(NSString * _Nullable substring, NSRange substringRange,
                                             NSRange enclosingRange, BOOL * _Nonnull stop) {
            count++;
        }];
    }
    return count;
}

- (NSUInteger)numberOfItems {
    return self.groceryList.count;
}

@end
```