International Journal of Engineering Applied Sciences and Technology, 2016
Vol. 1, Issue 10, ISSN No. 2455-2143, Pages 185-189
Published Online August - September 2016 in IJEAST (http://www.ijeast.com)

SPEED PERFORMANCE BETWEEN SWIFT AND OBJECTIVE-C

Harwinder Singh
Department of CSE
DIET, Regional Centre PTU,
Mohali, INDIA

***Abstracts*: The appearance of a new programming language gives the necessity to contrast its contribution with the existing programming languages to evaluate the novelties and improvements that the new programming language offers for developers. Intended to eventually replace Objective-C as Apple's language of choice, Swift needs to convince developers to switch over to the new language. Apple has promised that Swift will be faster than Objective-C, as well as offer more modern language features, be very safe, and be easy to learn and use. In this thesis developer test these claims by creating an iOS application entirely in Swift as well as benchmarking two different algorithms. Developer finds that while Swift is faster than Objective-C, it does not see the speedup projected by Apple. Swift was launched to offer an alternative to Objective-C because this has a syntax which barely evolved from it was created and has a great difference with other programming languages that have appeared in the latest years, because these have based on the C++ syntax. For this, Swift is inspired in new programming languages like C++11, C#, F#, Go, Haskell, Java, JavaScript, Python, Ruby. Then his syntax is totally different than its predecessor. The Swift's syntax is more simplified because it does not use pointers and includes improvements in its data structures and in its syntax.**

Keywords: Swift vs Objective-C, Swift in iOS mobile app, Swift,

I. INTRODUCTION

In the summer of 2008 Apple launched the App Store for the iPhone and iPod touch. Originally containing only 522 apps, as of 2014 the App store houses over 1 million apps and has seen over 75 billion app downloads. This platform has attracted thousands of developers to create applications for iOS devices, and has launched thousands of careers and companies. As Objective-C aged it became harder for new developers, unfamiliar with C and SmallTalk, to learn and understand. Languages such as Java, Python, and JavaScript became widely used and began to set the standard for modern programming languages. Developers began to complain that Objective-C was di cult to learn and uncomfortable to use.

Swift is a new programming language for iOS and OS X apps that builds on the best of C and Objective-C, without the constraints of C compatibility. Swift adopts safe programming patterns and adds modern features to make programming easier, more flexible, and more fun. Swift's clean slate, backed by the mature and much loved Cocoa and Cocoa Touch frameworks, is an opportunity to reimaging how software development works.

II. LITERATURE SURVEY

By Christ Lattner(2015)Released in June of 2014 by Apple Swift is a statically typed language and compiled language that uses the LLVM compiler infrastructure and the Objective-C runtime . Since Swift uses the same runtime as Objective-C the two languages can be intermixed in a single program or project, as both will compile down to native machine code. Swift can access Objective-C classes, types, functions, and variables through a "bridging header", as well as by extension C and C++ code. Similarly Objective-C can access code written in Swift, with some exceptions. This allows Swift to work with the Cocoa and Cocoa Touch frameworks and existing Objective-C apps and libraries without rewriting the large body of code that was written for iOS devices. Swift is heavily influenced by many other languages such as Rust, Haskell, Ruby, Python, and C#, and offers many of the object-oriented and functional features found in these languages. Swift also includes a read-eval-print-loop (REPL) that can be accessed in Xcode as well as on the command line.

Prof. Diwakar Gupta: Many previous researchers have proposed methods for evaluating and com-paring new programming languages. Languages are often compared against one another on a number of different criteria. Developers compared C++, Java, Perl, and Lisp together and their approach was extended to even more languages by other developers. Both of these papers conclude that each language has various pros and cons and are suited to different types of tasks, with Java and C receiving the most favourable reviews. Many programming language evaluations examine a language holistically and qualitatively, although attempts have been made to be more rigorous and quantitative. Urban propose a qualitative framework for assessing languages in terms of twelve different attributes including regularity, readability, reliability, portability, and Input/output. This framework provides a standardized way to evaluate a language in isolation and describes the key attributes important in any language design. Although these frameworks and comparisons help unveil the important aspects of a programming language, they are too high level to be appropriately applied to Swift. Other researchers have looked at programming languages as they apply to a specific domain. Since swift is meant to be used primarily for mobile devices, this type of research is more applicable. Gupta discusses the appropriateness of programming languages for teaching beginners or teaching, ultimately recommending basic or C. Howatt recommends evaluating a language based on how well it solves a given project or task on hand although he has doubts about the real world relevance of this approach. Oppermann and Compos discuss several popular languages used for mobile clients and server-side development. They conclude that using single language on both the mobile client and server offers a distinct advantage and that Java and Python are the best choices for this approach. Developers use design patterns to evaluate the Go programming language .They implement a subset of the Hot Draw framework in Go and use their implementation to motivate a discussion of the language . This project was the main source of inspiration for my analysis of Swift, since the authors used a large project to demonstrate their view on a new language. While I do not use design patterns or the Hot Draw frame-work in my Swift application.

By Apple's Developers Team(2014)Development on Swift was begun in July 2010 by Chris Lattner, with the eventual collaboration of many other programmers at Apple. Swift took language ideas "from Objective-C,Rust, Haskell, Ruby, Python, C#, CLU, and far too many others to list". On June 2, 2014, the Apple Worldwide Developers Conference (WWDC) application became the first publicly released app written in Swift. A beta version of the programming language was released to registered Apple developers at the conference, but the company did not promise that the final version of Swift would be source code compatible with the test version. Apple planned to make source code converters available if needed for the full release.

The Swift Programming Language, a free 500-page manual, was also released at WWDC, and is available on the iBooks Store and the official website.

Swift reached the 1.0 milestone on September 9, 2014, with the Gold Master of Xcode 6.0 for iOS.Swift 1.1 was released on October 22, 2014, alongside the launch of Xcode 6.1. Swift 1.2 was released on April 8, 2015, along with Xcode 6.3. Swift 2.0 was announced at WWDC 2015, and was made available for publishing apps in the App Store in September 21, 2015.A Swift 3.0 roadmap was announced on the Swift blog on December 3, 2015. However, before that, an intermediate Swift 2.2 embracing new syntax and features was introduced. This also omits some outdated components including Tuple splat syntax, C-style for loops.

Swift won first place for Most Loved Programming Language in the Stack Overflow Developer Survey 2015 and second place in 2016.

Google is said to be considering using swift as a first class language for its operating system android. During the wwd 2016, apple announced an ipad exclusive app, named swift playgrounds that will easily teach people how to code in swift. The app is presented in an interface which provides feedback when lines of code are placed in a certain order and executed.

III. RESEARCH WORK

Swift needs to convince developers to switch over to the new language. As it assumed that swift will be faster than objective-c, as well as swift is safe, and easy to learn and use. In this thesis developers, developing an ios application entirely in swift as well as benchmarking two different algorithms. As it's mentioned earlier that swift is faster than objective-c and it does not find the performance projected by ***DEVELOPERS***. I ALSO CONCLUDE THAT SWIFT HAS MANY ADVANTAGES OVER OBJECTIVE-C, AND IS EASY for developers to learn and use. However there are some weak areas of Swift involving interactions with Objective-C and the strictness of the compiler that can make the language bit difficult to work with. Apart from all these drawbacks, Swift is overall a successful software generating language for us.

The four stated goals of swift (safety, clarity, modernity, and performance) reflect both an accurate analysis of the deficiencies of objective-c and an fair assessment of the various developments that have taken place over the last three decades in programming language design--mostly object oriented languages like c++, java, python, and ruby, but also more specialized languages such as the functional language haskell.

If you have experience using Objective-C to develop for Apple platforms, you may be wondering: "Why did Apple release a new language?" After all, developers had been producing high-quality apps for Mac OS X and iOS for years. Apple has a few things in mind. First, Objective-C is an older language. And while this is not always a problem, it leads to some difficulty in this case. The syntax of Objective-C was solidified prior to the rise of prominent scripting languages in the 1990s that popularized more streamlined and elegant syntax (e.g., JavaScript, Python, PHP, Ruby, and others). This means Objective-C feels strange to most developers when they get started, so its syntax can be an impediment to developer productivity. Additionally, as an older language, Objective-C is missing many advancements developers in modern languages currently enjoy. Also, Swift aims to be safe. Objective-C did not aim to be unsafe, but things have changed quite a bit since Objective-C was released in the 1980s. For example, the Swift compiler aims to minimize undefined behavior, which is intended to save the developer time debugging code that failed during the runtime of an application. Another goal of Swift is to be a suitable replacement for the C family of languages (C, C++, and Objective-C). That means Swift has be fast. Indeed, Swift's performance is comparable to these languages in most cases. Swift gives you safety and performance all in a clean, modern syntax. The language is quite expressive; developers can write code that feels natural. This feature makes Swift a joy to write and easy to read, which makes it great for collaborating on larger projects.

3.1 Objectives

- To calculate speed of code execution with swift Language over Objective-c and other Programming Language used in iOS mobile application.

To measure performance of swift language on iOS mobile application platform.

Finding out Feature set and safety course for Swift Language.

3.2 Software And Hardware Requirements

3.2.1 Software Used:

- Mac operating system El-Capitan

IDE Xcode(Latest 7.3)

Language Swift (Version 2.2)

3.2.2 Hardware Used:

- Mac mini system form Apple or Other Mac CPU with OSX installed.

3.4 Methodology

Swift and Objective-C compilers are based on the LLVM Compiler Infrastructure, and there is a single iOS SDK for both Swift and Objective-C. That's why there isn’t much difference between the ways the programming languages work with the Cocoa frameworks.

We decided to examine both Swift and Objective-C performance by comparing their data structures. For that we took Objective-C Foundation framework and Swift’s native solutions.

In Swift, all classes are created during compile-time. Methods cannot be added on-the-fly and all types are known before the run time. Since everything is known beforehand, a compiler can optimize code without any problem.

Objective-C, on the other hand, can’t optimize as effectively, because all dynamic languages work slower than static.

Swift drops the two-file requirement. Xcode and the LLVM compiler can figure out dependencies and perform incremental builds automatically in Swift 1.2. As a result, the repetitive task of separating the table of contents (header file) from the body (implementation file) is a thing of the past. Swift combines the Objective-C header (.h) and implementation files (.m) into a single code file (.swift).

Objective-C’s two-file system imposes additional work on programmers -- and its work that distracts programmers from the bigger picture. In Objective-C you have to manually synchronize method names and comments between files, hopefully using a standard convention, but this isn’t guaranteed unless the team has rules and code reviews in place.

IV. RESULT

Test performed with an algorithm in each language to sort a list of 1,000,000 objects in ascending order. The objects were sorted in order based on a randomly generated numerical instance variable that ranged from −1000 to 1000. Since C does not support objects a struct was used instead. I ran each algorithm 25 times and plotted the results. Swift and Objective-C both performed approximately equal, running on average 1.4x faster than Python. However both languages paled in comparison to Java and C, with Java being on average twice as fast as either language.

Next performed task as multiplied two 500x500 matrices together with an O(n3) algorithm in each of the five languages, again running each algorithm 25 times in each C by -erformed Objectivelanguage. In this test Swift far outp a significant margin, running on average almost 10x faster C. While still not -than python and 4x faster than Objective s performance was on par with ’quite as fast as C, Swift Java. While these results are approximate they still show that it is unlikely that Swift is quite as fast as projected by .Apple

Apart from the performance benchmark swift contains the feature list, taken from other programming languages as they are listed below.

SWIFT FEATURE	LANGUAGES WITH SIMILAR FEATURES
CLOSURES	JAVASCRIPT
GENERICS	JAVA
TYPE INFERENCE	HASKELL
TUPLES	PYTHON
FUNCTIONS AS FIRST CLASS OBJECTS	JAVASCRIPT
OPERATOR OVERLOADING	C++
PATTERN MATCHING	SCALA,HASKELL

OPTIONAL TYPES	HASKELL, RUST
AUTOMATIC REFERENCE COUNTING	OBJECTIVE-C
PROTOCOLS	JAVA,C++
READ-EVAL-PRINT-LOOP (REPL)	PYTHON

V. CONCLUSION

Swift's level of raw processing performance has yet to be credibly demonstrated. But there is good reason to believe that this will be achieved, with Swift performing far faster than Python and Ruby, at least modestly faster than Objective-C, and even slightly faster than C in some circumstances.
Swift is not yet a production language for large-scale projects. Small and perhaps medium-scale apps have been successfully built and are in the App Store, built by developers with a certain persistence and willingness to work around problems. The compiler is slow, with delays apparently exponentially related to the amount of code and dependent upon which files and modules the code is packaged in. Error messages are not very informative, often at an extremely low level. Automatic bridging that is intended to make Swift work "seamlessly" with the iOS API doesn't always work. As discussed earlier, the compiler often produces poorly optimised code, including code with extraneous retain-release statements. The Xcode interactive development environment can be slow to respond with autocompletion and with flagging errors and turning error flags off when they have been fixed.

The reaction to Swift from the developer community has generally been very positive. Criticism has been minor. This is perhaps surprising given the programmer culture in which people have strong and often crazy opinions about everything, no matter how ill-informed.

Some of the negative reactions have been predictable: Enthusiasts of scripting languages like JavaScript think that not having implicit type conversion (automatically converting a data value's type when necessary) makes the language too inflexible. This is mostly a philosophical disagreement that has no solution. Some programmers feel that errors are inevitable and want their programs to run no matter what. Others want them to quickly fail in hopes of getting every LAST BUG OUT.

O Some major projects that were started in Swift reverted to Objective-C when compiler and other issues developed. Most of these projects are looking to migrate to Swift as soon as they reasonably can.

Some programmers will not be so quick to switch. Automatic Reference Counting has solved the most significant annoyance with Objective-C (having to allocate and deallocate memory manually) and its source of the most critical errors (memory leaks when memory not deallocated properly and crashes when deallocation is done unnecessarily). Many programmers will have worked with Objective-C so long that they have adapted to its quirks, are blind to its confusing aspects, and can work productively with it (although they surely spend a lot of time debugging.) There are other programmers who like doing tricky (and arguably unsafe) things with low level pointers.

Apple is quick to deprecate APIs that it no longer considers those it wants developers to use, and is aggressive about pushing users and developers to the latest versions of iOS and to relatively recent hardware. But Apple clearly needs Objective-C to maintain the legacy APIs, and parts of the iOS and OS X operating systems, that have been written in it. And there is a large base of app code that has been written in Objective-C. Apple is unlikely to be very quickly aggressive about getting to developers to switch to Swift. But Apple has made it easy to mix Swift and Objective-C code. It is quite possible that Apple will slowly nudge developers in the direction of using Swift, without prohibiting Objective-C. This might include requiring aÍpps being submitted to the App Store to be have their root controller written in Swift but allow calling on Objective-C code. It could also involve developing new APIs for Swift that do not work in Objective-C.

VI. REFERENCES

[1] @ADAMJLEONARD, @THINKCLAY, AND @CESAR_DEVERS, "SWIFT TOOLBOX," HTTP://WWW.SWIFTTOOLBOX.IO/, 2014. [ONLINE]. AVAILABLE: HTTP://WWW.SWIFTTOOLBOX.IO/. [ACCESSED: 17-APR-2015].
[2] E. GONZÁLEZ, H. FERNÁNDEZ, AND V. DÍAZ, "GENERAL PURPOSE MDE TOOLS," INT. J. INTERACT. MULTIMED. ARTIF. INTELL., VOL. 1, PP. 72–75, 2008.

[3] E. R. NÚÑEZ-VALDEZ, O. SANJUAN-MARTINEZ, C. P. G. BUSTELO, J. M. C. LOVELLE, AND G. INFANTE-HERNANDEZ, "GADE4ALL: DEVELOPING MULTIPLATFORM VIDEOGAMES BASED ON DOMAIN SPECIFIC LANGUAGES AND MODEL DRIVEN ENGINEERING," INT. J. INTERACT. MULTIMED. ARTIF. INTELL., VOL. 2, NO. REGULAR ISSUE, PP. 33–42, 2013.

[4] R. GONZALEZ-CRESPO, S. R. AGUILAR, R. F. ESCOBAR, AND N. TORRES, "DYNAMIC, ECOLOGICAL, ACCESSIBLE AND 3D VIRTUAL WORLDS-BASED LIBRARIES USING OPENSIM AND SLOODLE ALONG WITH MOBILE LOCATION AND NFC FOR CHECKING IN," INT. J. INTERACT. MULTIMED. ARTIF. INTELL., VOL. 1, NO. 7, PP. 63–69, 2012.

[5] A. PUDER AND I.YOON, "SMARTPHONE CROSS-COMPILATION FRAMEWORK FOR MULTIPLAYER ONLINE GAMES" IN 2ND INT. CONF. ON MOBILE, HYBRID, AND ON-LINE LEARNING, 2010 © IEEE COMPUTER SOCIETY.

DOI: 10.1109/ELML.2010.13.

[6] A. PUDER AND P. ANTEBI, "CROSS-COMPILING ANDROID APPLICATIONS TO IOS AND WINDOWS PHONE 7" IN MOBILE

NETWAPPL, 2012 © SPRINGER SCIENCE+BUSINESS MEDIA, LLC. DOI: 10.1107/S11036-012-0374-2

[7]INDERJEET SINGH AND MANUEL PALMIERI, "COMPARISON OF CROSS-PLATFORM MOBILE DEVELOPMENT TOOLS", IDT MALARDALEN UNIVERSITY

[8] C. XIN, "CROSS-PLATFORM MOBILE PHONE GAME DEVELOPMENT ENVIRONMENT" IN INT. CONF. ON INDUSTRIAL AND INFORMATION SYSTEMS, 2009 © IEEE COMPUTER SOCIETY. DOI: 10.1109/IIS.2009.96

[9]INC. APPLE. USING SWIFT WITH COCOA AND OBJECTIVE-C. 2. APPLE, INC., 2014.

[10] SULTAN S AL-QAHTANI, PAWELPIETRZYNSKI, LUIS F GUZMAN, RAfiKARIF, AND ADRIEN TEVOEDJRE. COMPARING SELECTED CRITERIA OF PROGRAMMING LANGUAGES JAVA, PHP, C++, PERL, HASKELL, ASPECTJ, RUBY, COBOL, BASH SCRIPTS AND SCHEME REVISION 1.0-A TEAM CPLGROUP COMP6411-S10 TERM REPORT. 2012.

[11] LEX FRIEDMAN. THE APP STORE TURNS fiVE: A LOOK BACKWARD AND FORWARD. HTTP://WWW.MACWORLD.COM/ARTICLE/2043841/ THE-APP-STORE-TURNS-FIVE-A-LOOK-BACK-AND-FORWARD. HTML, JULY 2013.

[12] JAMES HOWATT. A PROJECT-BASED APPROACH TO PROGRAMMING LANGUAGE EVALUATION. ACM SIGPLAN NOTICES, 30(7):37–40, 1995.

[13]PRASHANT KULKARNI, HD KAILASH, VAIBHAV SHANKAR, SHASHINAGARAJAN, AND DL GOUTHAM. PROGRAMMING LANGUAGES: A COMPARATIVE STUDY. 2008.

[14] STEPHEN G. KOCHAN. PROGRAMMING IN OBJECTIVE-C. SAMS PUBLISHING, 1999.