FEDERAL UNIVERSITY OF PERNAMBUCO

BSc COMPUTER SCIENCE
GRADUATION PROJECT

AN EMPIRICAL STUDY ON THE USAGE OF THE SWIFT PROGRAMMING LANGUAGE

Marcel de Siqueira Campos Rebouças

Recife, January 6, 2016

FEDERAL UNIVERSITY OF PERNAMBUCO

BSc COMPUTER SCIENCE
GRADUATION PROJECT

AN EMPIRICAL STUDY ON THE USAGE OF THE SWIFT PROGRAMMING LANGUAGE

A monograph submitted to the Center of Informatics of the Federal University of Pernambuco in partial fulfillment of the requirements for the degree of Bachelor of Science in Computer Science.

Author: Marcel de Siqueira Campos Rebouças
(mscr@cin.ufpe.br)

Supervisor: Fernando José Castor de Lima Filho
(fjclf@cin.ufpe.br)

Recife, January 6, 2016

Acknowledgements

I would like to express my gratitude to my advisor, Fernando Castor, for his support and encouragement throughout the research, and also for the great opportunity in a field I had no previous background. I am also grateful for the help provided by Gustavo Pinto, Weslley Torres and Felipe Ebert, whose technical advice was essential to the completion of this research and has taught me countless insights and lessons on the fundamentals of academic research in general.

My thanks also go to the friends and teachers that worked with me this year, whose help allowed me to learn about mobile development and Swift.

Last, but not least, I would like to thank my family. To them, I give my wholehearted recognition and love for their commitment and the many years of support during my undergraduate studies that provided the foundation for this work.

Abstract

The mobile market is facing an unprecedented growth, with iOS and Android platforms playing a central role in this arena. Recently, Apple released Swift, a modern programming language built to be the successor of Objective-C. In less than a year and a half after its first release, Swift became one of the most popular programming languages, considering different popularity measures. A significant part of this success is due to Apple's strict control over its ecosystem, and the clear message that it will replace Objective-C in a near future. This distinctive scenario presents a unique opportunity to understand the adoption of a programming language from its very early stages.

According to Apple, *"Swift is a powerful and intuitive programming language[...]. Writing Swift code is interactive and fun, the syntax is concise yet expressive."* However, little is known about how Swift developers perceive these benefits.

In this paper, we conducted two studies aimed at uncovering the questions and strains that arise from this early adoption. First, we perform a thorough analysis on 59,156 questions asked about Swift on StackOverflow. Second, we interviewed 12 Swift developers to cross-validate the initial results. Our study reveals that developers do seem to find the language easy to understand and adopt, although 17.5% of the questions are about basic elements of the language. Still, there are many questions about problems in the toolset (compiler, Xcode, libraries). Some of our interviewees reinforced these problems.

Resumo

O mercado de dispositivos móveis está enfrentando um crescimento sem precedentes, com as plataformas iOS e Android desempenhando um papel central neste processo. Recentemente, a Apple lançou Swift, uma linguagem de programação moderna construída para ser o sucessor de Objective-C. Em menos de um ano e meio após seu primeiro lançamento, Swift tornou-se uma das linguagens de programação mais populares, de acordo com diferentes medidas de popularidade. Uma parte significativa desse sucesso parte do controle rigoroso da Apple sobre seu ecossistema, e a mensagem clara de que ele irá substituir Objective-C em um futuro próximo. Este cenário distinto apresenta uma oportunidade única para entender a adoção de uma linguagem de programação, desde seus estágios iniciais.

Segundo a Apple, *"Swift é uma linguagem de programação poderosa e intuitiva [...]. O código em Swift é interativo e divertido, e a sintaxe é concisa e expressiva."* No entanto, pouco se sabe sobre como os desenvolvedores Swift percebem esses benefícios.

Neste trabalho, realizamos dois estudos destinados a descobrir as questões que surgem a partir desta adoção antecipada. Primeiro, realizamos uma análise aprofundada em 59,156 perguntas sobre Swift no StackOverflow. Em seguida, foram entrevistados 12 desenvolvedores para validar os resultados iniciais. Nosso estudo revela que os desenvolvedores parecem achar a linguagem fácil de entender e utilizar, embora 17,5% das perguntas sejam sobre os elementos básicos da linguagem. Além disso, há muitas perguntas sobre problemas no conjunto de ferramentas (compilador, Xcode, bibliotecas). Alguns de nossos entrevistados reforçaram estes problemas.

Contents

List of Figures

List of Tables

1 Introduction

In the last years, the mobile app market is facing a fascinating growth, with iOS and Android devices playing a central role in this arena. As a recent article shows[2], over a billion of mobile devices are going to be sold in 2015 — which is about twice the number of personal computers. This fact creates a high demand not only for new mobile developers, but also for new techniques, tools, and frameworks to ease mobile programming practice. As an attempt to mitigate this problem, in June 2014 Apple released Swift, a modern, multi-paradigm language that combines imperative, object-oriented, and functional programming.

More interestingly, however, is that Swift is experiencing a fast popularity growth. According to specialized websites[3][4], Swift is already one of the top-20 most popular programming languages in the world. Most of this success is due to the inherited Objective-C ecosystem, with more than 700 million devices sold [18]. Moreover, Swift is intended to be a "safer" alternative to pure Objective-C. For instance, Swift is statically-checked, discourages the use of `nil`, and supports Maybe-like[5] optional types and error handling. These key elements makes Swift adoption almost guaranteed. Also, considering that Apple has been using Objective-C for almost 20 years (it was acquired in 1996), we can expect Swift's lifespan to be similarly long. This distinctive scenario presents a unique opportunity to understand the adoption of a programming language from its very early stages.

This paper is a first step in the quest to understand the benefits, drawbacks, and hurdles of being an early adopter of a programming language that is bound to be widely adopted. earch is still in its early stages, in this paper we focus on a high-level research question and two that emphasize differences between Swift and Objective-C. Specifically, the questions we are trying to answer are:

RQ1. What are the most common problems faced by Swift developers?

RQ2. Are developers having problems with the usage of Optionals?

RQ3. Are developers having problems with error handling in Swift?

[2]`http://www.entrepreneur.com/article/236832/`

[3]http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html

[4]http://redmonk.com/sogrady/2015/07/01/language-rankings-6-15/

[5]https://hackage.haskell.org/package/base-4.8.1.0/docs/Data-Maybe.html

Our interest in Optionals (**RQ2**) and error handling mechanism (**RQ3**) is related to the fact that those features constitute the major differences between Swift and its predecessor Objective-C, and therefore can hinder adoption of Swift by the experienced Objective-C developers.

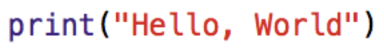
Although many researchers have proposed methods for evaluating programming languages [17], no consensus has emerged from a methodological standpoint, *e.g,* methods proposed in the literature are prone to be subjective [19]. In this paper we present two studies. The first one is based on a quantitative and qualitative analysis of data from StackOverflow, a collaborative Q&A website. We complemented the results from StackOverflow with our second study: 12 interviews with Swift developers with different backgrounds. These two studies provide important insights on the current state of practice Swift programming.

2 Background

2.1 Swift

After using Objective-C - a language first released in 1983 - for 18 years, Apple announced they were releasing a new programming language on June 2014, called Swift. It was designed to be a language that accomodates the best of C and Objective-C, without the constraints of C compatibility, and to be used on every Apple plaftorm (iOS, OS X, watchOS, and tvOS).

Swift is a multi-paradigm (imperative, object-oriented, and functional), statically-checked and compiled programming language. It was designed to work with Apple's main frameworks (*i.e.,* Cocoa and Cocoa Touch) and to be fully compatible with Objective-C, while being less prone to erroneous code (due to features like Optionals) and more concise.



```
print("Hello, World")
```

Figure 1: Hello World Implementation in Swift

Swift shares some of the core concepts of Objective-C. For instance, it uses dynamic dispatch, late binding and extensible programming. It also shares the same memory management (*i.e.,* Automatic Reference Counting). On the other hand, some of Swift's features allowed it to be more similar to the "modern" languages we have today, like Java or Python (*e.g.,* no need for header files, type inference, no need for semicolons, operator overloading, unicode strings, generic programming, functions as first-class objects - which allows it to be passed as an argument - and error handling). Apple also refers to Swift as a "protocol-oriented language"[6], since it added the concept of protocol extensibility, in which types, structs and classes can be extended and modified.

2.1.1 Optionals

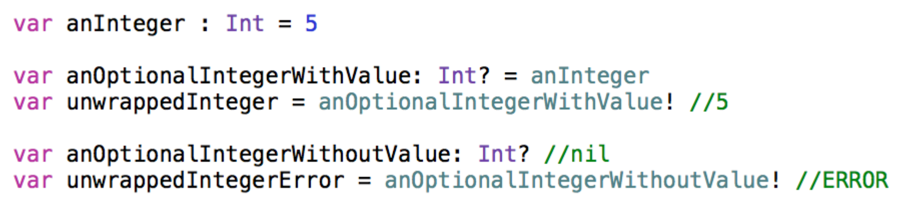
Swift brings a concept that was not part of the Objective-C universe: Optionals.

Added as a safety mechanism, Optional types are types that either contains a value, or contains *nil.* This helps to prevent common programming errors, like null pointers, since the compiler ensures that non-optional types can never be *nil.* This mechanism is similar to *maybe* in Haskell, or *Nullable Types* in C#.

In order to create an Optional type, the developer just needs to add a question mark (?) after the type when declaring a variable or constant (*e.g.,* var name : String?). Then, the marked variable will either have a value of the underlying type or will be empty. The Optional mechanism wraps the base type, and creates a different instance, which means that the types String and String? are different. Furthermore, in order to retrieve the Optional value, the developer can use the *!* operator to unwrap the content of the Optional type, assuming it is not empty. Unwrapping a *nil* value will result in a crash.

This feature also adds a technique that can be used to make the code less complicated, called Optional Chaining. In this technique, the developer is able to call methods from Optional Types, and they will only be executed if the Optional contains a value. If not, the method call will just be ignored.

[6]https://developer.apple.com/videos/play/wwdc2015-408/



```swift
var anInteger : Int = 5

var anOptionalIntegerWithValue: Int? = anInteger
var unwrappedInteger = anOptionalIntegerWithValue! //5

var anOptionalIntegerWithoutValue: Int? //nil
var unwrappedIntegerError = anOptionalIntegerWithoutValue! //ERROR
```

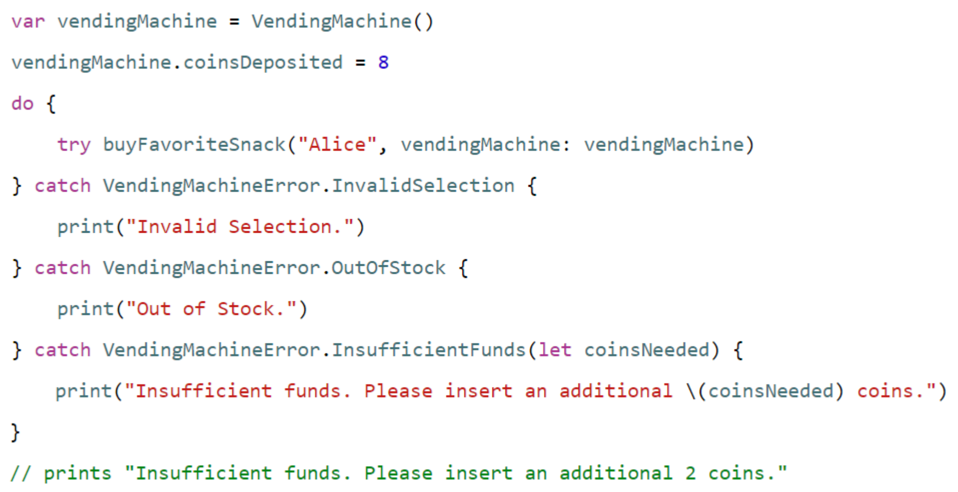
Figure 2: An example of the Optionals syntax. It is shown how to declare an Integer(Int), an Optional Integer(Int?) with a starting value and an Optional Integer starting with a null value.

2.1.2 Error handling

Error handling is the act of recovering from erroneous conditions on the execution of a program.

In Objective-C, dealing with error conditions could be done with the use of the *NSException*(which encapsulates information like the name and reason) and *NSError*(which holds an error code and other data) classes. The former involved the usage of the try-catch-finally pattern, should only be used in critical conditions (able to cause a crash) and is considered to have a poor performance in Objective-C. The latter works differently. Instead of using dedicated constructs, NSErrors are added as an additional argument in a failable method or function and, if an error occurs, the NSError is then filled with the error data. This is, most of the time, done in callback functions, that execute asynchronously (*e.g., loading a webpage, or retrieving data from an API call.*

The first release of Swift only had the NSError construct, as used in Objective-C, and worked the same way. However, the version 2.0 added a new form of error handling to Swift. This version added first-class support for throwing, catching, propagating, and manipulating recoverable errors at runtime, through the usage of do-try-catch syntax. In this new form, errors are represented by values of types that conform to the ErrorType protocol (a protocol that indicates that a type can be used for error handling). Image 3 shows an example of Swift's error handling syntax.



```swift
var vendingMachine = VendingMachine()
vendingMachine.coinsDeposited = 8
do {
    try buyFavoriteSnack("Alice", vendingMachine: vendingMachine)
} catch VendingMachineError.InvalidSelection {
    print("Invalid Selection.")
} catch VendingMachineError.OutOfStock {
    print("Out of Stock.")
} catch VendingMachineError.InsufficientFunds(let coinsNeeded) {
    print("Insufficient funds. Please insert an additional \(coinsNeeded) coins.")
}
// prints "Insufficient funds. Please insert an additional 2 coins."
```

Figure 3: An example of the error handling syntax added in Swift 2.0. Example extracted from Apple's Swift documentation.[8]

2.2 StackOverflow

StackOverflow[9] is an online platform for users to ask and answer questions related to programming and software engineering. Aside from the Q&A part, users can also comment and upvote/downvote posts, based on the relevance or the help it provides. By giving good answers, users can receive reputation points.

StackOverflow makes its data publicly available through the StackExchange website[10], under the Creative Commons license. Using this resource, it is possible to access the data from posts (questions and answers), comments, votes, users, tags and other metadata. Until October 2015, StackOverflow had more that 10M questions and over 18M answers.

StackOverflow uses a tag system to categorize the questions. When a question is created, the user is asked to select the tags the represent that questions (*e.g.,* Swift, iOS). The tags are also autocompleted, which prevents the user from creating new tags or adding typographical errors. In this

[8]http://apple.co/1IXnmQQ
[9]http://stackoverflow.com/
[10]https://data.stackexchange.com/

research, we are interested in studying the posts that contains tags related with Swift.

StackOverflow questions and its metadata served as our main source of data.

2.3 Topic Modeling

Since we will work with thousands of text documents (the questions), manual analysis is not feasible. For this reason, we use a topic modeling approach. A topic model is a type of statistical model that aims to discover abstract "topics" that happens to exist in a set of documents. This form of text mining looks for patterns in the set, and groups clusters of words, for which we call a "topic" (*e.g., "church, priest, religion" or "star, planet, space"*). These words are clustered following a metric of similarity, which depends on the algorithm used.

Given that a document talks about a certain topic (*e.g., space*), it is intuitive that some words are more likely to be used than others (*e.g., "star, planet" are more likely to occurr than "shoes, sandals" in a space-related topic.*). Also, the topic "religion" might have the words "church, priest" in its vocabulary. Furthermore, some words might appear equally on both topics, also known as "stopwords" (*e.g., the, for, about*). A document can have multiple topics, with different proportions (*e.g.,* a topic can be 90% about space, and 10% about travel). The topic model analyzes each word in the document and, through probabilistic approaches, discovers what the topics might be, and which topics are present in each document.

Apart from the necessary large set of documents, it is recommended that each document is "clean" from stopwords and other tokens that might add noise to the results. The method used to prepare the documents are further explained in Section 4.

3 Related Work

There are a plethora of studies targeting the usage of different programming language usage and constructs. They vary from how developers use concurrent techniques [12, 15], the `sun.misc.Unsafe` class [10], the `goto` statement [11], generics [14], inheritance [22], among many others. As regarding mobile application, more recently, some researchers are studying how refactoring can be applied in asynchronous mobile applications [13, 9]. In the programming language usage arena, Schmager *et al.* [19] evaluated the Go programming language, Chandra *et a.* [5] compared Java and C#, in terms of their strengths and weaknesses. Hadjerrouit [6] evaluated Java as a first programming language, and Stefik and Hanenberg [20] discussed the responsibilities the community has in regard to the programming language wars (the discussion about their differences and impacts), among others. We also analyzed the usage of a programming language based on two studies: by looking what developers were asking about it and also by a set of interviews. The closest work to us is from Barua *et al.*. However, although we shared the same methodology, Barua *et al.* were focused on a broad aspect, analyzing the main topics of interest on StackOverflow. In our case, we are more restrict to the Swift usage. Also, when Barua *et al.* conducted their study, Swift was not even launched, so our findings do not overlap with their ones in any sense.

4 Methodology

The goal of this study is to improve the understanding of the problems faced by Swift early-adopter developers, in particular, and possibly of early programming language early adopter in general. In this section we describe the procedure used to download and process the analyzed data (Section 4.1), and how we chose the interviewees, how the interviews were conducted (Section 4.2).

4.1 Study 1: Mining Software Repositories

In this section we describe the steps needed to gather our dataset of questions.

Data Extraction. We used the StackExchange[11] website to extract StackOverflow questions, answers, comments, and their metadata (*e.g.,* Score, View Count, Answer Count, Favorite Count). We retrieve questions that contain any of the following tags: 'swift', 'swift2', 'swift-playground', 'swift-json', 'swift-extensions', 'sqlite.swift', 'swift-protocols', 'swift-array', 'cocos2d-swift', 'swift-dictionary', 'objective-c-swift-bridge', 'rx-swift', 'dollar.swift', 'swift2.0' and 'swift-custom-framework'. We used these tags because StackOverflow associates them when searching for 'swift'. One might argue that tagging is a manual process, which would incur in mistagged questions. However, StackOverflow autocompletes tags, thus preventing one from using a misleading tag.

In total 59,156 questions posted by 33,681 users have been retrieved together with 74,297 answers and 216,403 comments to these questions. These questions span the period from June 2, 2014 (when Swift was released - ℚ24001778) till October 11, 2015 (when we ran the query).

Will Swift-based applications work on OS X 10.9 (Mavericks)/iOS 7 and lower?

ℚ24001778: 50% iOS Testing, 27% IDE—xcode

The first Swift question on StackOverflow

To answer our research questions, we analyzed qualitative properties of questions, answers and tags. Qualitative coding was done with support of

[11] `http://data.stackexchange.com/`

a text mining algorithm, and was further confirmed in collaborative coding sessions.

Data Processing. Since manual inspection of 59 thousand of questions is not feasible, we used the approach of Barua *et al.* [1], based on Latent Dirichlet Allocation (LDA) [4], a topic modeling algorithm[12]. This algorithm summarizes large amounts of text documents. LDA assumes that each document, that is, the StackOverflow questions, in a given set is a mix of different topics, so that each word in the document can be associated with one or more topics with a certain proportion. LDA has been extensively applied in many domains [2, 3].

Before feeding the LDA with our questions, we removed (1) content inside the tags `<code>`, `<pre>` and `<blockquotes>`, (2) HTML tags, (3) URLs, (4) punctuation, (5) one-letter words, and (6) stop word (*e.g., an, by, the*). We also stemmed the remaining words using the Porter stemming algorithm [16], to reduce words to their base form (*e.g., "compilation" and "compiler" are reduced to "compil"*). Figure 4 shows an example of a question before and after the pre-processing step. The resulting files were then used as the LDA input.

(a) before processing

<p> I would like to try out Swift, but currently don't have an Apple developer's account. Would it be possible to compile and run it without having Xcode 6?</p>

$\Downarrow$

(b) after processing

swift appl develop account compil run xcode

Figure 4: An example of a question (a) as it appears in the StackOverflow, and (b) after the pre-processing steps.

LDA deals with documents, which in our case are the StackOverflow questions. LDA assumes that each document in a given set is a mixture of different topics, and that each word in the document can be associated with

[12]We used the implementation available in the Mallet-2.0.8RC2 tool.

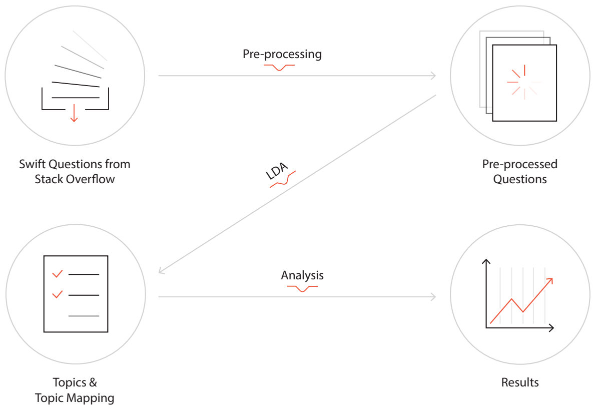


Figure 5: Our research methodology.

one or more topics with a certain proportion. Given a set of documents, LDA tries to identify the topics and creates a mapping between topics and documents. In the current work we used the LDA implementation available in the Mallet tool, version Mallet-2.0.8RC2[13].

LDA can be configured in different ways. Hence, for the sake of completeness, we describe the parameters that we used. The first parameter is the number of topics that LDA will generate, in which we used 25. There is no "right" value for this parameter as it depends on the granularity one wants to achieve. We tested the number of topics ranging from 10 to 35. Using a small number of topics (*i.e.,* 15) resulted in topics that were too general, most of them being perceived as having a mix of subjects. We believe that the mixing of subjects occurs because, since most questions are related to Swift and the iOS environment, they have general terms that would make sense to appear in more than one topic. For example, the word *time* could appear in a *performance-related* topic, or in a *date-related* topic, with the same relevance.

Furthermore, we configured Mallet to look for uni-grams (single words) and bi-grams (sequences of two words). This results in topics that look like, *e.g.,* view, navigation, view_control, objective_c. Using uni-grams and bi-grams improves the quality of the resulting topics [21],.

Since a document can have multiple topics, we denote the *membership* of

[13]http://mallet.cs.umass.edu/

a topic t_i in a document d as $\theta(d_k, t_i)$. According to Blei *et al.* [4], a document normally contains between 1 and 5 topics, with a membership value above 0.10. For this reason, we set the membership threshold δ to 10%, value commonly used in LDA studies[1]. Since Mallet uses a probabilistic approach, some topics can be assigned to documents with a very low membership (e.g. 1%) which can lead to false positives. To mitigate this case, the threshold helps to remove noise topics from the output, while keeping the dominant ones. This same *membership* is used in the *share* and *most_relevant* metrics, described in the RQ1.

Output. The output of executing Mallet is a list of topics and a mapping of the document-topic relations. The list of topics was composed of 25 groups, each containing 10 related words, that we manually labeled for simplifying purposes.Some of the topics required the reading of a sample of questions. We used this processed data to provide answers to **RQ1**. Besides the LDA processing, we performed additional queries to filter representative StackOverflow questions to answer **RQ2** and **RQ3**. Details about this questions are provided at the beginning of each **RQ**'s discussion.

4.2 Study 2: Interviews

We conducted semi-structured interviews in order to cross-validate the results obtained in the StackOverflow study. Therefore, interviews provided additional discussions to **RQ1**, **RQ2**, and **RQ3**. This method was chosen to take advantage of the knowledge of each participant, allowing us to evolve the questions based on gained insights. Also, since participants had different levels of development experience, interviews allowed us to understand different points of view.

Participants with varying development experiences were selected, through different methods. We selected 2 undergrad students, and 10 professionals. Among the professionals, 3 are Swift professors – professors that teach educational programs for iOS development — and 1 works for a local software company. These professionals were reached through convenience sampling. To compensate for the inherent limitations of the convenience sampling, we analyzed the most popular (in terms of stars) Swift software repositories hosted on Github. For each repository, we identified the top-3 contributors of each project and invited them to participate in the interviews. We ended up contacting 43 open-source Swift developers, and 6 of them (14%) participated in our interviews.

Table 1: The LDA Topics, and their set of words.

# Topic	LDA Words
Error—General	code work swift problem fine function issu wrong call work_fine
UI—Navigation	view control view_control segu bar viewcontrol navig storyboard tab back
Error—Debugging	error swift code type line compil crash messag argument issu
Q&A	swift question find answer solut make document appl read understand
Data Storage	arrai data object dictionari save core core_data swift store creat
OO Programming	class method call swift protocol type subclass implement creat deleg
Data Types	string swift function type number convert return int paramet charact
Objective-C Interop.	swift c objective objective_c framework project import header object_c object
UI—TableView	cell tabl tableview view row tabl_view uitableview custom select section
IDE—Xcode	xcode file project swift beta build xcode_beta error creat test
iOS Testing	app io devic iphon simul run applic crash io_app work
UI—Positioning	view size height constraint scroll set screen width layout uiview
UI—Actions	button click press keyboard tap action user swift uibutton custom
Cloud/Social Media	user pars notif app login facebook queri log messag post
Image Handling	imag color background chang photo set camera uiimag pictur uiimageview
Networking	json data request server swift api respons url alamofir send
Game Development	game scene node sprite spritekit screen player move score make
Variables Def/Use	variabl properti swift access set declar struct refer assign object
UI—Animations	anim move draw swipe gestur touch left screen uiview rotat
Noise—General Words	search swift list io creat select user app tutori appreci
Multithread/Sched. Func	call function time updat load complet data block run thread
Optionals/Nil	option nil print return statement unwrap error check found unwrap_option
UI—Text	text label field crash line uilabel font text_field chang set
Media/Time Comp.	plai date video time sound timer audio record swift dai
Location/Web Comp.	locat map user annot page googl url webview html uiwebview

In total 12 interviews were performed. Each participant was interviewed by a member of our research team. 3 of the interviews were conducted in person, the remaining 9 were conducted via phone. Interviews lasted approximately 20 minutes and the audio was recorded. We refer to the anonymized interview participants as P1—P12. Table 2 gives additional demographic information on each of the 12 participants. Participants ranged in professional programming experience from less than a year (graduating students) to 10 years, averaging 4.16 years.

Both graduating students had less than 1 year of mobile development practice, and 3 months of experience with Swift, but mostly on academic projects. The Swift educators we interviewed had more than 5 years of professional software development experience (5, 5 and 10 years). Two of them (P11 and P12), are using Swift since its very first release. Amongst the software developers, their professional experience ranges from 2 to 6 years. Furthermore, 6 of them were top contributors in popular Swift software repositories.

Table 2: Demographic information about our interviewees. The Swift Familiarity is graded as: Somewhat Familiar, Moderately Familiar and Strongly Familiar. Column "Objective-C" shows if the developer has Objective-C background. Column "Experience" describes their experience in software development (in years).

#	Job Title	Swift Familiarity	Objective-C?	Experience
P1	Software Dev.	●●●	✓	5
P2	Software Dev.	●●●	✓	5
P3	Software Dev.	●●●	✓	5
P4	Software Dev.	●●○	✓	6
P5	Software Dev.	●●○	✓	2
P6	Software Dev.	●●●	✓	5
P7	Student	●●○	✓	<1
P8	Student	●●○	✓	<1
P9	Software Dev.	●●○	×	2
P10	Educator	●●○	✓	5
P11	Educator	●●●	✓	5
P12	Educator	●●●	✓	10

The interviews were grounded in research questions RQ1–3. For each interview, we started by asking about the interviewee's background in software development. Some of the questions included (1) *"Do you have professional experience with Objective-C?"* and (2) *"When did you start using Swift?"*. After understanding the background of the interviewee, we then moved to the specific questions about the language. We asked three questions about the learning process, and two questions about the challenges that they faced when writing Swift applications, if any. Then, we ended the interview by asking about problems and solutions found when using Optionals and Error Handling mechanisms.

To analyze the data, we first transcribed all the audio files. Each transcript, along with the associated recording, was analyzed by two of the authors. We then coded the answers, analyzed the keywords, organized them into categories. We followed the guidelines on the open coding procedure [7].

5 Results

In this section we describe our results organized in terms of the research questions.

5.1 RQ1: What are the most common problems faced by Swift developers?

For this RQ, we ranked each topic found in the StackOverflow questions using the *share* metric (Equation 1), which is similar to the one used by Barua *et al.* [1]. This ranking is shown at Table 3. The *share* of a topic t_i is the sum of the memberships of this particular topic for every document d in our dataset. Furthermore, since we used a membership threshold of 10% when running Mallet, the topics with a membership lower than this value for any given document d are not taken into consideration. For this reason, the *share* does not add up to 100%.

$$share(t_i) = \frac{1}{|D|} \sum_{d \in D} \theta(d_k, t_i) \tag{1}$$

Table 3 also presents the *Most Relevant* metric, which is calculated with the following formula 2. Here, D(t_i) represents the subset of documents that contains the topic t_i with a membership higher than the threshold (Equation 3). Also, Equation 4 checks if t_i is the dominant topic - with a higher membership value than all of other topics - for a document d_k. *Most Relevant* represents the proportion of documents in D(t_i) in which t_i has the highest membership value.

$$\text{most\_relevant}(t_i) = \frac{1}{|D(t_i)|} \sum_{d \in D(t_i)} dom(d_k, t_i) \tag{2}$$

$$\text{D}(t_i) = \{d \in D | \theta(d_k, t_i) \geqslant \delta\} \tag{3}$$

$$\text{dom}(d_k, t_i) = \begin{cases} 1, & \text{if } \forall t \in T, \theta(d_k, t_i) \geq \theta(d_k, t) \\ 0, & \text{otherwise} \end{cases} \tag{4}$$

We then manually grouped the topics in Table 3 into seven main themes, namely *Swift Standard Library*, *Cocoa Framework*, *General Code Problems*, *Testing and Errors*, *Integration with Objective-C*, *IDE* and *Others* . These themes are described below. For each of them, we present a representative example of questions, and relate the theme with findings from our interviews.

Table 3: The LDA Topics, with the number of questions associated, and the values of share and relevance.

# Topic	Share (%)	Questions	Most Relevant
Error—General	7.5	36.7%	13.5%
UI—Navigation	5.0	15.5%	44.7%
Error—Debugging	4.5	16.5%	32.9%
Q&A	4.5	22.0%	14.8%
Data Storage	4.4	14.6%	41.3%
OO Programming	4.3	15.3%	34.7%
Data Types	4.1	13.4%	38.2%
Objective-C Interop.	3.8	11.8%	41.0%
UI—TableView	3.7	12.0%	43.4%
IDE—Xcode	3.6	13.5%	30.8%
iOS Testing	3.5	13.2%	30.5%
UI—Positioning	3.5	10.6%	43.9%
UI—Actions	3.4	12.8%	31.1%
Cloud/Social Media	2.9	9.4%	42.0%
Image Handling	2.9	10.1%	36.6%
Networking	2.8	8.8%	44.3%
Game Development	2.8	7.2%	55.9%
Variables Def/Use	2.5	9.6%	29.9%
UI—Animations	2.4	8.1%	37.0%
Noise—General Words	2.2	9.6%	23.0%
Multithread/Sched. Func	2.2	8.7%	28.8%
Optionals/Nil	2.2	8.5%	29.1%
UI—Text	2.0	8.0%	28.0%
Media/Time Comp.	2.0	6.0%	48.7%
Location/Web Comp.	1.7	5.8%	42.0%

5.1.1 Swift Standard Library

Out of the 25 topics, 5 were categorized as *Swift Standard Library*: *Data Storage*, *OO Programming*, *Data Types*, *Variables—Definition and Usage* and *Multithreaded/Scheduled Functions.* The Standard Library comprehends a base layer of functionality for writing Swift programs, which includes the basic data types, data structures, functions and methods, protocols, etc. Their total *share* is 17.5%.

Swift is advertised as "a language that is easy and fun to use"[14], and "friendly to new programmers"[15]. Yet, almost 1/5 of the questions are about the syntax and constructs of the language. However, the fact that Swift was easy to learn was supported by the interviewees. 10 of them reported that they did not have problems with the language syntax. Also, 9 of them suggested that the syntax is very similar to other languages they knew (*e.g.,* Java or Python).

This contrast may be explained by the fact that all of the interviewees were experienced with other programming languages, and 11 out of 12 knew Objective-C beforehand, while developers posting on StackOverflow can have a much broader experience range. Since Swift and Objective-C share some features (*e.g.,* the same readability of named parameters and similar method names), part of the complexity is reduced. The contrast suggests that Swift may be an easy language if the developer has previous experience with other languages, specially Objective-C, but the contrary may not be true . Meanwhile, P4 and P6 do not think that knowing Objective-c helped on learning Swift, as P6 said *"No, I don't think it did. I mean, like knowing Objective-c helped Swift but only a little bit but only because Swift was designed to be familiar to Objective-c developers. If I was a brand new student that had never done Objective-c and never done Swift, Objective-c makes it harder to learn Swift because there are problems in Swift that are only there because of Objective-c".* Examples of questions in this group, with their corresponding topics are:

✎ *"How to remove elements from array that match elements in another array?"*—ℚ`25250809`: 95% *Data Storage.*

✎ *"Is it possible that subclasses from MySuperClass access the class function dummyDict() at runtime?"*—ℚ`24711515`: 91% *OO Programming.*

✎ *"How to create a function accepting any type of Int or Uint in Swift,*

[14]https://developer.apple.com/swift/

[15]http://apple.co/1DgqEVo

and calculate the number of bits regardless of param type?"—ℚ`24986319`: 92% *Data Types.*

5.1.2 Cocoa Framework

Cocoa Touch is the core framework used to develop iOS applications. As its documentation[16] states *"The Cocoa Touch layer contains key frameworks for building iOS apps. These frameworks define the appearance of your app. They also provide the basic app infrastructure and support for key technologies such as multitasking, touch-based input, push notifications, and many high-level system services"*.

We find 7 topics that are related to the Cocoa Framework. They are: *UI—Navigation*, *UI—TableView*, *UI—Layout*, *UI—Actions*, *UI—Animations*, *UI—Text* and *Image Handling*. Their total *share* is equals to 22.9%. It is interesting to observe that the number of topics, and the total share of this theme are higher than the first one (Swift Standard Library).

Even though Swift 2.0 is expected to become Open Source[17], which may allow its usage in other areas of development, currently it is still strongly tied to mobile development, which implies the usage of Cocoa.. As P10 stated *"There isn't much sense in learning Swift without learning and using the frameworks"*. P7 also said *"Since I already knew the frameworks from Objective-C, and it is almost the same in Swift, I could write code almost straight away"*. Examples of questions in this group with their corresponding topics are:

✎ *"I created a UIViewController in my Main.storyboard with a few buttons and labels. I'm trying to switch to that view controller using self.present ViewController but it will not load the view from storyboard. It will only load a blank black screen by default. Any idea on how to load the view from what i've created in storyboard?"*—ℚ`25474115`: 88% *UI—Navigation.*

✎ *"Hi I have a TableViewController with two static cells, however when displayed, it shows the two static cells, and then all the other empty cells for the tableview. However I don't want to display those cells as they are useless. How do I hide the rest of the cells apart from the two static cell?"*—ℚ`28708574`: 88% *UI—TableView.*

✎ *"How do I programmatically create graphical elements (like a UIButton) in Swift? I tried to create and add button into a view, but wasn't able*

[16]http://apple.co/1PnR96G

[17]https://developer.apple.com/swift/blog/?id=29

to."—ℚ24030348: 89% *UI—Actions.*

5.1.3 General Code Problems

The topics *Error—General* and *Q&A* are related to general code issues. These two topics achieved some of the highest *share* values: 7% and 4.5% respectively. But, even though their share is high, they are not related to any technical Swift category. This is something likely to happen, since questions are written in natural language and general terms like *problem* or *answer* are used in order to explain and give meaning to them. Both topics also had the lowest *Most Relevant* values, which means that they were not dominant topics—in 86.5% and 85.2% of their appearances, they were just secondary topics. We do not show examples of questions in this category, since this is not a category we are interested in.

5.1.4 Testing and Errors

Three topics are contained in this category: *Error—Debugging*, *iOS Testing* and *Optionals/Nil*, with a total share of 10.2%.

We found it interesting that 11 of 12 of the interviewees complained about the Swift compiler and the error messages, they also indicated that those were a nuisance in the usage of the language. As P9 stated, *"The compiler was quite unstable sometimes, which led to errors that we didn't expect. Sometimes, I didn't even knew the cause of it, just how to fix it."*. He also said that he *"had problems with the error messages, that were not clear and sometimes led to more doubt"*. The Swift 1.2 change log[18] shows that those issues were known to the language developers: *"Better compiler diagnostics—Clearer error and warning messages, along with new Fix-its, make it easier to write proper Swift 1.2 code."* and *"Stability improvements—The most common compiler crashes have been fixed. You should also see fewer SourceKit warnings within the Xcode editor."*. And P4 was more critical saying that *"the biggest problem right now, like the biggest problem by far is the instability of the tools, because Swift compiler is like the worst compiler I could ever imagine and that multiplied by hundred, I think."*.

Interestingly, we found that the Optionals construct had its own topic (5034 questions were categorized as so). The interviews showed that the

[18]https://developer.apple.com/swift/blog/?id=22

opinions on the matter are quite different. Even though 10 of the interviewees said that Optional was an easy concept to grasp, and recognized its importance, 5 of them stated that they were not 100% sure about its usage in some occasions. P9 also declared that the constant use of the Optionals symbols `?` and `!` made the code a little bit confusing. Examples of questions in this group, with their corresponding topics are:

✎ *"I used this code and I'm getting error "Could not find an overload for "init" that accepts the supplied arguments"."*—ℚ`26511891`: 94% *Error—Debugging.*

✎ *"Ok so I am trying to run this code but I keep on getting this error: fatal error: unexpectedly found nil while unwrapping an Optional value. I don't understand what it means or why im getting it. Any hint?"*—ℚ`24948302`: 91% *Optionals/Nil.*

5.1.5 Integration with Objective-C

This category is represented by the topic *Objective-C Interoperability.* This was the 8th topic on the *share* ranking (Table 3). Swift and Objective-C share the same frameworks and they are interoperable, which means that they can be used together in the same project. This fact also allows an easier migration, since it is possible to translate and replace parts of the Objective-C apps.

Objective-C is, still, more used than Swift[23]. It has a much more content (*e.g.,* 4 times more questions on StackOverflow) and many APIs that were not migrated. Therefore, Swift developers need to have an understanding of Objective-C. This need is recognized by the interviewees: *"I believe that, in order to someone to be considered an iOS developer, he needs to know Objetive-C"* (P11); *"I learned Swift without knowing Objective-C. But soon I had to use an API that reads barcodes, and it only had an Objective-C version. I had to do simple modifications, but it was a little complicated since the syntax is was unfamiliar. After some research, I was able to make it work."* (P9). An example of a question in this theme is:

✎ *"I've made a framework that requires the sqlite3 framework. How do I add a Objective C Bridging Header for my framework that imports sqlite3 into my Swift file? I already have a bridging header file for my project, but not for my framework."*—ℚ`24841144`: 96% *Objective-C Interoperability.*

5.1.6 IDE

The IDE category is represented by the topic *IDE—Xcode*, 10th on the *share* ranking. Xcode is the IDE made by Apple that contains a suite of development tools that enables the development to the iOS ecosystem (and other Apple products, like OS X).

Some of the interviewees related issues that occurred while using the IDE, mainly because of the version changes. Since its release, Swift received two major upgrades (v1.2 and v2.0), Xcode went from v6 (Swift release) to v7 (Swift 2.0) and iOS was updated from v7 to v9. These updates incur changes in the way the code should be used and written, causing the old code no longer to compile. As an example, interviewees indicate that the Swift 2.0 converter, a tool in Xcode that helps to convert older Swift code to the latest Swift syntax, was not 100% reliable: *"When the new version was released, I had to convert my code, since it wasn't working anymore. It wasn't a difficult task. But it was kind of a problem when APIs I was using also stopped working"* (P10). Furthermore, P9 talked about the difficulties to merge changes made to the Storyboard, an Interface Builder present in Xcode. Even though it is not exactly Swift code, it is still something that the developers have to deal with. An example of a question in this theme is:

✎ *"After upgrading to xCode6.1, I can't compile my project which is ok last time. I have never changed anything in this file. Please help me !!!"*—ℚ26502746: 94% *IDE—Xcode.*

5.1.7 Others

Some of the topics were too specific to fit in the categories above, however they are still important to be discussed. The *Game Development* topic shows that there's an interest in developing iOS native games, and the majority of the questions are related to the 2D framework SpriteKit. Another topic, *Cloud/Social Media*, demonstrates that the developers are interested in using third-party APIs, like Parse[19] and Facebook[20], to connect their apps, and that there are doubts about the integration and initial usage (*e.g.,* performing a login on Facebook, or a query on Parse). Moreover, we had problems with topic that we labeled *Noise—General Words.* After looking through a sample of questions, some of it were related to the data structure *List.* Still, a lot of

[19]https://www.parse.com/

[20]https://developers.facebook.com/

other unrelated questions appears because it had general words (*e.g.*, Swift, iOS, App) that could be part of most topics.

5.2 RQ2: Are developers having problems with the usage of Optionals?

Swift makes use of *optional types*: "`var x: Int?`" means that the variable `x` either has a value and this value is an integer, or it does not have a value at all. We studied Swift's optionals for at least three reasons. First, although commonplace in functional languages, such as Haskell and ML, optionals types are rarely available in imperative languages. Second, in Swift optionals types are also pervasive. For example, some functions operating on dictionaries, one of the basic data structures of the language, require the use of optionals. It is not natural to build non-trivial Swift programs without using optionals. Third, optionals are a major feature of Swift that is not available in Objective-C and can thus be an obstacle to its adoption by experienced Objective-C developers.

As we show in our first research question (§ 5.1), the LDA technique successfully identified questions related to Optional usage. In this research question we are motivated to further investigate how the Optional construct is being used in practice. The LDA technique classified 1,451 questions (8.5% of the total) as Optionals related (Table 3). Interestingly, the answer rate of these questions is four times higher than the overall questions about Swift (90.14% of them have answers, and 60.02% have an accepted answer). Since this high number of questions prevents manual analysis from being successful, we decide to study the questions that had a high LDA score. We use this approach because, after a manual investigation, we observed that these questions are more likely to be related to Optionals concerns. On the other hand, questions with a low LDA score are not directly associated with Optionals usage, for instance, the user wants to improve one aspect of her application, which is using an Optional variable (*e.g.,* ℚ`30147712`, score: 0.1053). We then selected and investigated the 3rd quartile of questions (363 ones) ranked using their score value. When analyzing these questions, we found and removed 10 false-positive questions (*e.g.,* ℚ`29313022`), resulting in 353 categorized questions. After examining the title, the question body, and the associated tags of all the 353 selected questions, we ended up with 4 main groups of Optional related questions.

5.2.1 Errors

203 occurrences. Most of questions are related to errors that happen during runtime or compile time. In particular, the "fatal error: unexpectedly found nil while unwrapping an Optional value" error is the most common one, with 185 occurrences. This error occurs when a user is trying to unwrap an optional variable that holds a `nil` value. This error happens in different contexts, mainly because several Swift APIs made use of Optional values, for instance: ℚ24948302 deals with graphic components, ℚ29730819 deals with audio components, and ℚ28882954 deals with URL components. As a solution to this problem, several StackOverflow users have pointed the use of optional chaining. Three interviewees also agree with this suggestion.

5.2.2 Basic Usage

88 occurrences. This category groups questions that deals with Optional basic usage. For instance, (1) checking the value of an optional variable (*e.g.,* ℚ25523305), (2) unwrapping an optional variable (*e.g.,* ℚ33049246), and (3) printing an optional variable. Some interviewees reported difficulties when using the operators `!` and `?`, which "*are not straight-forward to understand*" (P9). Still, although simple, the printing example is rather common (11 occurrences found). One StackOverflow user summarized this problem as: "*For one of my static labels on my main story board, it prints out Optional("United States"). However, I would like it to print out "United States". So my question is, how do I get rid of the "Optional" part?*" (ℚ32101920). This happens because the user is trying to print the value of an optional variable which was not unwrapped. The solution is straightforward: unwrap the variable before printing.

5.2.3 Optional Idiosyncrasies

38 occurrences. Here we group questions that focus on peculiar Optional features. Most of the questions deal with Optional chaining (*e.g.,* ℚ28046614), with 13 occurrences, and Optional binding (*e.g.,* ℚ26576366), with 7 occurrences. Optional chaining is an important strategy to deal with Optionals. It not only favors readability, but it also makes the code safer because it avoids forced unwraps, which could lead to the "unexpectedly found nil while unwrapping" error. Multiple calls can be chained together, and the whole chain fails elegantly if any part of it is nil. P2 also said that optional chaining is "*an alternative instead of using if and elses and it makes your code cleaner*".

5.2.4 Concepts

24 occurrences. Finally, this group of questions is related to the very basic concepts about Optionals. Usually they are high level and aimed at understanding how the technique is used behind the scenes. Some of them are not even related to source code (ℚ`32154698`). Some example of questions include: (1) what happens while wrapping and unwrapping an Optional? (ℚ`27370700`), (2) what is the difference between optional and forced unwrapping? (ℚ`28665375`), and (3) the difference between "optional chaining" and "optional call chaining" (ℚ`31143806`). Interestingly, we found that all these questions have answers, and only 3 of them do not have an accepted answer. When analyzing the questions that do not have an accepted answer, we found that these questions had at lease one answer that one can judge as accepted, but the author of the question did not mark it as so (*e.g.,* ℚ`32154698`).

5.3 RQ3: Are developers having problems with error handling in Swift?

We study the Error Handling mechanism because Swift only recently introduced it in its 2.0 release. In the 2.0 approach, errors are thrown using the `throw` statement, and are handled by using the `do-try-catch` syntax. Before that, Swift developers had to use the old associative Objective-C solution, which envolves using an `NSError` object (an object that encapsulates information about an error condition). Here we analyze if developers are using `NSError`, using ad hoc solutions, or if they are migrating to the Swift 2.0 solutions. Since the LDA technique did not identify topics related to error handling, we performed additional queries with specific Error Handling terms, including "NSError", "except handl", "try", "catch", "error", "finally", "defer", and "throw". This query returned 563 questions. While manually analyzing these question, we found that 411 of them were false positive (*e.g.,* ℚ`27325139` deals with errors in general). After removing these questions, we ended up with 152 Error Handling related questions. These questions are categorized into 2 categories, discussed next.

When analyzing these questions, we found that the majority of them (87.5%) fits into the topic "Error - Debugging" (Table 3). Interestingly, none of these questions are related to "Game Development", "UI - Animations", "UI - Positioning" or "UI - TableView". In order to better understand these questions, we categorized these questions into 2 groups. These groups are discussed next.

5.3.1 How to handle error in Swift?

74 occurrences. We found that developers suggest that error handling can be done using the old associative `NSError` implementation. Furthermore, some developers are also using the newly introduced Error Handling mechanism. More interestingly, however, is the fact that 14 developers are proposing the usage of a particular approach: result enumerations (ℚ27611433, ℚ28552710). Indeed, one interviewee (P3) mentioned that "A lot of people in the community are using result enums". Still, P3 raised that the current mechanism that Swift provides does not support asynchronous computation, which is a unfortunate since mobile applications are becoming much more asynchronous to improve responsiveness [13]. One StackOverflow user also raised the same point (ℚ30812959).

5.3.2 How to migrate to Swift 2.0?

78 occurrences. In this group there are questions about how to translate error handling from another language like Java or Objetive-C into Swift 2.0 (*e.g.,* ℚ31667074). Questions like that, might indicate that developers are migrating to the new error handling mechanism. There are also questions (*e.g.,* ℚ32809294) about compiling errors due to the migration process. These errors happened for various reasons, like not knowing how to use `try` and `catch` statements. Multiple questions (ℚ32694669, ℚ32651449) also asked about the *"Call can throw, but is not marked with try and the error is not handled"* error, which was solved by correctly using the do-try-catch pattern.

6 Discussions

Overall Assessment. Developers seem to find the language easy to understand and adopt. This is the opinion of most of our interviewees. Also, the majority of the questions are about libraries and frameworks, instead of the language itself. Nonetheless, a considerable number of questions are targeting Optional Types, which does not exist in Objective-C. Since Swift has other features that are not in Objective-C which are not mentioned as frequently, such as overflow operators, this large number of questions about optionals suggests that this subject is both relevant and non-trivial. This is reinforced by answers from some of our interviewees.

In addition, it may still be too early to make a switch to Swift for production development. There are many questions on StackOverflow about bugs in the toolset and also about error messages that are either hard-to-understand or unhelpful. One of our interviewees went so far as claiming that "*Swift compiler is like the worst compiler I could ever imagine and that multiplied by hundred*". On the one hand, this result is not entirely surprising, considering that the language has just a year old. On the other hand, the Swift Web site states that "*Swift is ready for your next project*"[21].

Implications. This research has implications for different kinds of stakeholders. Software developers can learn from the mistakes made by their peers. For instance, learning how to use Optional variables is important and since Swift frameworks use this construct intensively, they cannot ignore it (**RQ2**). Since some developers argued that the error handling mechanism that Swift provides is not effective (*e.g,* it does not handle asynchronous code), researchers can conduct empirical studies further investigate this claim. Also, researchers can study strategies for improving this error handling mechanism. Tool makers can take advantage of the high number of questions related to User Interface (**RQ1**), and develop tools to make the usage, customization and animations of UI elements easier. Still, we found that majority of problems related to Optional usage were related to inappropriately unwrapping optional variables (**RQ2**). Tool makers can use this finding, and improve their tools to provide hints of when it is safe to unwrap a variable.

Yet, since Swift Error Handling mechanism is different from the ones found in well-known programming languages (*e.g.,* Java and C#), and Swift developers are facing a hard time using them (**RQ3**), professors can provide a better foundation on this regard. In particular, professors should incentivize students to discuss the strengths and weakness of each approach. Finally, we found that compiler errors raised in the initial versions of the language are, by far, the most unpleasant part of the language. Language designers

[21]https://developer.apple.com/swift/

can benefit from this finding and improve the error messages raised by their compilers.

Threats to Validity. The first threat is related to the number of LDA topics chosen. Although there is no "right" solution, we ran several tests in order to verity the number of topics that best fits in our scope. Also, due to the high number of associated questions, we manually analyzed only a subset of Optionals and Error Handling-related questions. However, we believe that this sample of questions is representative (See the beginning of Section 5.2 and 5.3). Still, even though our analysis focus only on StackOverflow. However, StackOverflow is one of the most popular Q&A websites for developers. We also correlated the findings with interviews with 12 Swift developers. As regarding the interviews, our interview script might not have covered all questions that could have been asked. To mitigate this threat we followed established empirical methods for interviews [8]. Finally, we also designed the interviews to be semistructured. This allowed us to ask questions that were not listed in the script.

7 Conclusion

With less than two years, Swift became one of the most popular programming languages. After conducting two studies (12 interviews plus quantitative and qualitative analysis on StackOverflow), we found that there is no rose garden. Although experienced developers seem to find the language easy to understand and adopt, a significant proportion of questions are about the basic language constructs. Even though optional types are pervasive in Swift development, most of the problems report trivial unwrapping errors. There are many questions about bugs in the toolset (compiler, Xcode, libraries) and also about error messages that are either hard-to-understand or unhelpful. Also, interviewees were unanimous suggesting that the Swift compiler needs urgent improvement.

References

[1] A. Barua, S. W. Thomas, and A. E. Hassan. What are developers talking about? an analysis of topics and trends in stack overflow. *Empirical Softw. Engg.*, 19(3):619–654, June 2014.

[2] I. Bhattacharya and L. Getoor. A latent Dirichlet model for unsupervised entity resolution. In *Proceedings of the Sixth SIAM International Conference on Data Mining, April 20-22, 2006, Bethesda, MD, USA*, pages 47–58, 2006.

[3] I. Bíró, D. Siklósi, J. Szabó, and A. A. Benczúr. Linked latent dirichlet allocation in web spam filtering. In *Proceedings of the 5th International Workshop on Adversarial Information Retrieval on the Web*, AIRWeb '09, pages 37–40, 2009.

[4] D. M. Blei, A. Y. Ng, and M. I. Jordan. Latent dirichlet allocation. *J. Mach. Learn. Res.*, 3:993–1022, Mar. 2003.

[5] S. S. Chandra and K. Chandra. A comparison of java and c#. *J. Comput. Sci. Coll.*, 20(3):238–254, Feb. 2005.

[6] S. Hadjerrouit. Java as first programming language: A critical evaluation. *SIGCSE Bull.*, 30(2):43–47, June 1998.

[7] R. Hoda, J. Noble, and S. Marshall. Developing a grounded theory to explain the practices of self-organizing agile teams. *Empirical Softw. Engg.*, 17(6):609–639, Dec. 2012.

[8] S. Kvale. *Interviews : an introduction to qualitative research interviewing / Steinar Kvale.* Sage Publications Thousand Oaks, Calif, 1996.

[9] Y. Lin, C. Radoi, and D. Dig. Retrofitting concurrency for android applications through refactoring. In *Proceedings of the 22Nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*, FSE 2014, pages 341–352, 2014.

[10] L. Mastrangelo, L. Ponzanelli, A. Mocci, M. Lanza, M. Hauswirth, and N. Nystrom. Use at your own risk: The java unsafe api in the wild. In *Proceedings of the 2015 ACM SIGPLAN International Conference on Object-Oriented Programming, Systems, Languages, and Applications*, OOPSLA 2015, pages 695–710, 2015.

[11] M. Nagappan, R. Robbes, Y. Kamei, E. Tanter, S. McIntosh, A. Mockus, and A. E. Hassan. An empirical study of goto in C code from GitHub repositories. In *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*, ESEC/FSE 2015, pages 404–414, 2015.

[12] S. Okur and D. Dig. How do developers use parallel libraries. In *FSE*, 2012.

[13] S. Okur, D. Hartveld, D. Dig, and A. Deursen. A study and toolkit for asynchronous programming in C#. In *ICSE*, 2014.

[14] C. Parnin, C. Bird, and E. R. Murphy-Hill. Adoption and use of java generics. *Empirical Software Engineering*, 18(6):1047–1089, 2013.

[15] G. Pinto, W. Torres, B. Fernandes, F. Castor, and R. S. Barros. A large-scale study on the usage of javas concurrent programming constructs. *Journal of Systems and Software*, 106(0):59 – 81, 2015.

[16] M. F. Porter. Readings in information retrieval. chapter An Algorithm for Suffix Stripping, pages 313–316. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1997.

[17] T. W. Pratt and M. V. Zelkowitz. *Programming Languages: Design and Implementation*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 4th edition, 2000.

[18] D. Reisinger and S. Tibken. Apple by the numbers: 700m iphones, 25m apple tvs sold, Mar. 2015. Available at `http://www.cnet.com/news/apple-by-the-numbers-453-retail-stores-worldwide-120m-customers/`. Accessed: 2015-09-04.

[19] F. Schmager, N. Cameron, and J. Noble. Gohotdraw: Evaluating the go programming language with design patterns. In *Evaluation and Usability of Programming Languages and Tools*, PLATEAU '10, pages 10:1–10:6, 2010.

[20] A. Stefik and S. Hanenberg. The programming language wars: Questions and responsibilities for the programming language community. In *Proceedings of the 2014 ACM International Symposium on New Ideas, New Paradigms, and Reflections on Programming & Software*, Onward! 2014, pages 283–299, 2014.

[21] C.-M. Tan, Y.-F. Wang, and C.-D. Lee. The use of bigrams to enhance text categorization. *Inf. Process. Manage.*, 38(4):529–546, July 2002.

[22] E. Tempero, J. Noble, and H. Melton. How do java programs use inheritance? an empirical study of inheritance in java software. In *Proceedings of the 22Nd European Conference on Object-Oriented Programming*, ECOOP '08, pages 667–691, 2008.

[23] Tiobe. Tiobe index for september 2015, Sept. 2015. Available at `http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html`. Accessed: 2015-09-09.

Appendices

A Interview Script

- (1) Interviewee Background
  - How long have you been working as software developer?
  - How many/which programming languages have you used in professional software projects?
  - Do you have professional experience with Objective-C? If so, when did you start using it?
  - When did you start using Swift?
  - How familiar are with Swift? (Not that familiar, familiar, strong familiar with)
- (2) Language Adoption
  - What motivated you to learn Swift?
  - What made swift easy/difficult to learn?
  - What are the data sources you use to get updated with Swift? (understand/study/learn)
  - If the interviewee knows objective-c: Did objective-c help to ease the learning? How/Why?
- (3) General Problems
  - What are the most common problems that face when using Swift?

  - How could those problems be solved?

- (4) Specific problems (Optional Usage and Exception Handling)
  - Are you familiar with the idea of Optionals?
  - Did you find any difficulties/problems about the usage of Optionals?
  - Do you see any advantages about its usage?
  - Are you familiar with the concept of Exception Handling?
  - If the interviewee used Swift 1.0: Swift 1.0 doesn't support exception handling, nor does it support catching them. Did you miss EH on Swift 1.0?
    * If yes, what workarounds did you use to handle exceptional behavior?
  - Do you see any advantages about its usage?
  - If the interviewee used Swift 2.0: Have you used it in Swift 2.0? How/Why?
  - If the interviewee used both: Did you refactor your old code to use EH? Do you plan to do this refactor?
    * If yes, Are there challenges in doing so?