An Empirical Study on the Usage of the Swift Programming Language

Marcel Rebouças, Gustavo Pinto, Felipe Ebert, Weslley Torres
Federal University of Pernambuco
Recife, PE, Brazil
{mscr,ghlp,fe,wst}@cin.ufpe.br

Alexander Serebrenik
Eindhoven University of Technology
Eindhoven, The Netherlands
a.serebrenik@tue.nl

Fernando Castor
Federal University of Pernambuco
Recife, PE, Brazil
castor@cin.ufpe.br

***Abstract*—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.**

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.

I. 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[1], 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[2][3], 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 [4]. 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. Despite the growing interest of studies aimed at identifying the impact of specific language designs on people [15], [16], not many works target specifically elements of the Apple ecosystem. Until January 2015, app store developers earned a revenue of US$ 25 billion[5], which is larger than any other app store, to the best of our knowledge.

This work 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. Since this research 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?

Although many researchers have proposed methods for evaluating programming languages [14], [16], no consensus has emerged from a methodological standpoint, *e.g,* methods proposed in the literature are prone to be subjective [15]. 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.

II. Related Work

There are a plethora of studies targeting different programming language usage and constructs. They vary from unsafe classes [8], the `goto` statement [9], concurrent constructs [12], generics [11], inheritance [17], among many others. In the programming language usage arena, Schmager *et al.* [15]

[1] http://www.entrepreneur.com/article/236832/
[2] http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html
[3] http://redmonk.com/sogrady/2015/07/01/language-rankings-6-15/
[4] http://cnet.co/1Q7v1ML
[5] http://apple.co/1IvR211

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 [16] 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.* [1]. However, although we shared the same methodology, Barua *et al.* focused on a broader question of 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 theirs in any sense.

III. Methodology

Here we describe how we acquired and processed data (§ III-A), and how the interviews were conducted (§ III-B). All data is available for replication purposes[6].

A. Study 1: Mining Software Repositories

In this section we describe how we collected our dataset.

Data Extraction. We used the StackExchange[7] 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. 59,156 questions were found. They span the period from 2/06/2014 (when Swift was released) to 11/10/2015 (when we ran the query).

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[8]. 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 [13], to reduce words to their base form (*e.g., "compilation" and "compiler" are reduced to "compil"*). Figure 1 shows an example of a question before and after the pre-processing step.

(a) before processing

 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?

⇓

(b) after processing

swift appl develop account compil run xcode

Fig. 1. An example of a question (a) as it appears on StackOverflow, and (b) after the pre-processing steps.

LDA uses different parameters. The first one is the number of topics created. There is no "right" value for this parameter as it depends on the granularity one wants to achieve. We chose 25 after testing values ranging from 10 to 35. Also, since a document can have multiple topics, we denote the *membership* of 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 *et al.* [1]. This threshold removes noise topics from the output, while keeping the dominant ones.

Output. The output of is a list of 25 topics mapped to each document analyzed. We used this processed data to provide answers to **RQ1**, and we performed additional queries to answer **RQ2** and **RQ3**.

B. Study 2: Interviews

We conducted semi-structured interviews to cross-validate the results obtained in our first study. We interviewed 2 students and 10 professionals. Three interviews were conducted in person, while the remaining ones were conducted via phone. Interviews lasted approximately 20 minutes and audio was recorded. Among the professionals, 3 are Swift instructors in educational programs for iOS development, 1 works for a Brazilian software company, and 6 are open-source developers of popular (in terms of number of stars) Swift projects hosted on Github. 6 of them consider themselves as "strongly familiar" with Swift, and the other 6 consider themselves as "familiar". Also, 11 of them have an Objective-C background. On average, they have 4 years of software development experience. We refer to them as P1—P12.

The interviews were grounded in **RQ1–3**. We started asking about the interviewee's background in software development. Then, we moved to specific questions, including three questions about the learning process and two questions about the challenges that they faced when writing Swift applications, if any. Finally, we asked 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

[6]http://bit.ly/saner-swift-developers

[7]http://data.stackexchange.com/

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

TABLE I
THE LDA TOPICS, SHARE, AND THE % OF QUESTIONS ASSOCIATED.

# Topic	Share	Questions	Category
Error—General	7.5%	36.7%	General Problems
UI—Navigation	5.0%	15.5%	Cocoa Framework
Error—Debugging	4.5%	16.5%	Testing and Errors
Q&A	4.5%	22.0%	General Problems
Data Storage	4.4%	14.6%	Standard Library
OO Programming	4.3%	15.3%	Standard Library
Data Types	4.1%	13.4%	Standard Library
Objective-C Interop.	3.8%	11.8%	Objective-C
UI—TableView	3.7%	12.0%	Cocoa Framework
IDE—Xcode	3.6%	13.5%	IDE
iOS Testing	3.5%	13.2%	Testing and Errors
UI—Positioning	3.5%	10.6%	Cocoa Framework
UI—Actions	3.4%	12.8%	Cocoa Framework
Cloud/Social Media	2.9%	9.4%	Others
Image Handling	2.9%	10.1%	Cocoa Framework
Networking	2.8%	8.8%	Others
Game Development	2.8%	7.2%	Others
Variables Def/Use	2.5%	9.6%	Standard Library
UI—Animations	2.4%	8.1%	Cocoa Framework
Noise—General Words	2.2%	9.6%	Others
Multithread/Sched. Func	2.2%	8.7%	Standard Library
Optionals/Nil	2.2%	8.5%	Testing and Errors
UI—Text	2.0%	8.0%	Cocoa Framework
Media/Time Comp.	2.0%	6.0%	Others
Location/Web Comp.	1.7%	5.8%	Others

the keywords, organized them into categories. We followed the guidelines on the open coding procedure [7].

IV. RESULTS

Here we organize the results in terms of each **RQ**.

A. RQ1: The most common problems

Here we use a similar *share* metric [1] (Equation 1) to rank the topics, shown at Table I. The *share* of a topic t_i is the sum of the memberships of this particular topic for each document d. Also, since we used a membership threshold of 10%, 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%.

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

We then manually grouped the topics for simplifying purposes. Table I shows the seven main categories, namely *Swift Standard Library*, *Cocoa Framework*, *General Code Problems*, *Testing and Errors*, *Integration with Objective-C*, *IDE* and *Others*. Three of them are described below.

Swift Standard Library (5 topics, 17.5% share). This topic comprehends a base layer of functionality for writing Swift programs, including data types, data structures, functions and methods, protocols, among many others. Even though Swift is advertised as "*a language that is easy and fun to use*"[9], and as "*friendly to new programmers*"[10], almost 1/5 of the questions are about the language syntax and constructs. Interestingly, 10 of the interviewees reported that they did not have problems with the language syntax. Still, 9 of them suggested that the syntax is very similar to other languages they knew (*e.g.,* Java).

This paradox may be explained due to the fact that all of the interviewees were experienced with other programming languages, and 11 out of 12 knew Objective-C beforehand. Since Swift and Objective-C share some features (*e.g.,* the same readability of named parameters), part of the learning curve is reduced. This suggests that Swift might be easy to learn if developers have previous experience with other languages, specially Objective-C. Meanwhile, P4 and P6 assured that they do not think that knowing Objective-C helped on learning Swift.

Cocoa Framework (7 topics, 22.9% share). Cocoa Touch is the core framework used to develop iOS applications. It defines the appearance and also provides the basic application infrastructure. It is interesting to observe that the number of topics, and the total share of this category are higher than the first one (Swift Standard Library). This high number of topics, and therefore questions, is because Swift 2.0 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"*. However, it is expected that Swift become Open Source[11], which may allow its usage in other areas of development.

Testing and Errors (3 topics, 10.2% share). 11 of 12 of the interviewees complained about the Swift compiler and the error messages, and 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."*. The Swift changelog[12] shows that those issues were known to the language designers. P4 was more critical saying that *"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."*. Unknown error messages are also common problems reported in StackOverflow (*e.g.,* Q32743382). The lack of backward compatibility as also a nuisance reported in the interviews, as P10 said *"the version changes made some of the outdated code to stop working, which was a problem when using APIs"*.

Integration with Objective-C (1 topic, 3.8% share). Swift and Objective-C share frameworks, which allows an easier migration. Therefore, Swift developers need to have an understanding of Objective-C. This need is recognized by the interviewees: *"To someone to be considered an iOS developer, he needs to know Objetive-C"* (P11). However, StackOverflow users often report the need of workaround to integrate with Objective-C (*e.g.,* Q24841144).

B. RQ2: The problems with Optionals usage

We studied Swift's optionals for at least three reasons. First, although commonplace in functional languages, such

[9]https://developer.apple.com/swift/
[10]http://apple.co/1DgqEVo
[11]https://developer.apple.com/swift/blog/?id=29
[12]https://developer.apple.com/swift/blog/?id=22

as Haskell and ML, optionals types are rarely available in imperative languages. Second, in Swift optionals types are also pervasive. 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.

The LDA technique classified 1,451 questions (8.5% of the total) as Optionals related (Table I). 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.,* Q30147712, 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.,* Q29313022), 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. Due to space constrains, we provide discussions to only three of them.

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.

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.

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*".

C. RQ3: The problems with error handling usage

We study the Error Handling mechanism because Swift only recently introduced it in its 2.0 release. Before that, Swift developers had to use the old associative Objective-C solution (based on `NSError`). Here we analyze if developers are using `NSError`, using ad hoc solution, or if they are migrating to Swift 2.0. 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.,* Q27325139 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.

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 [10]. One StackOverflow user also raised the same point (Q30812959).

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`. Another example are the questions (Q32390611) that asks about the difference between `try`, `try!` and `try?`. This kind of question shows that this aspect of the language may not be very intuitive yet.

V. 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*"[13].

Implications. 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 are related to User Interface (**RQ1**), and develop tools to make the usage, customization and animations of UI elements easier. Still, the 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. Still, we found that several developers were having problems with the transition from Objective-C to Swift (**RQ3**). For instance, 38 questions asked about migrating their code from Objective-C `NSError` mechanism to the Swift 2 error handling one. Tool vendors can provide mechanism to assist developers to ease this migration process.

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 to verify the number of topics that best fits in our study. Second, due to the high number of questions, we manually analyzed only a subset of Optionals and Error Handling-related questions. However, we believe that this sample is representative (See the beginning of § IV-B and § IV-C). Third, we use a single Q&A website, StackOverflow, which is one of the most popular among developers. We also correlated the findings with 12 interviews. Fourth, our interview script might not have covered all possible questions. However, we also designed the interviews to be semistructured. This allowed us to ask questions that were not listed in the script.

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

VI. 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. 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*, 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 Workshop on Adversarial Information Retrieval on the Web*, 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] 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 *OOPSLA*, pages 695–710, 2015.

[9] 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 *ESEC/FSE*, pages 404–414, 2015.

[10] S. Okur, D. L. Hartveld, D. Dig, and A. v. Deursen. A study and toolkit for asynchronous programming in c#. In *ICSE*, pages 1117–1127, 2014.

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

[12] 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.

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

[14] T. W. Pratt and M. V. Zelkowitz. *Programming Languages: Design and Implementation*. Prentice Hall PTR, 4th edition, 2000.

[15] F. Schmager, N. Cameron, and J. Noble. Gohotdraw: Evaluating the go programming language with design patterns. In *PLATEAU*, pages 10:1–10:6, 2010.

[16] A. Stefik and S. Hanenberg. The programming language wars: Questions and responsibilities for the programming language community. In *Onward!*, pages 283–299, 2014.

[17] E. Tempero, J. Noble, and H. Melton. How do java programs use inheritance? an empirical study of inheritance in java software. In *ECOOP*, pages 667–691, 2008.