On the Adoption of Kotlin on Android Development: A Triangulation Study

Victor Oliveira
Informatics Center - UFPE
Tempest Security Intelligence
Recife, Brazil
victor.oliveira@tempest.com.br

Leopoldo Teixeira
Informatics Center
Federal University of Pernambuco
Recife, Brazil
lmt@cin.ufpe.br

Felipe Ebert
Informatics Center
Federal University of Pernambuco
Recife, Brazil
fe@cin.ufpe.br

Abstract*—Background: In 2017, Google announced Kotlin as one of the officially supported languages for Android development. Among the reasons for choosing Kotlin, Google mentioned it is "concise, expressive, and designed to be type and null-safe". Another important reason is that Kotlin is a language fully interoperable with Java and runs on the JVM. Aims: Despite Kotlin's rapid rise in the industry, very little has been done in academia to understand how developers are dealing with the adoption of Kotlin. The goal of this study is to understand how developers are dealing with the recent adoption of Kotlin as an official language for Android development, their perception about the advantages and disadvantages related to its usage, and the most common problems faced by them. Method: This research was conducted using the concurrent triangulation strategy, which is a mixed-method approach. We performed a thorough analysis of 9,405 questions related to Kotlin development for the Android platform on Stack Overflow. Concurrently, we also conducted a basic qualitative research interviewing seven Android developers that use Kotlin to confirm and cross-validate our findings. Results: Our study reveals that developers do seem to find the language easy to understand and to adopt it. This perception begins to change when the functional paradigm becomes more evident. Accordingly to the developers, the readability and legibility are compromised if developers overuse the functional flexibility that the language provides. The developers also consider that Kotlin increases the quality of the produced code mainly due to its null-safety guarantees, but it can also become a challenge when interoperating with Java, despite the interoperability being considered as an advantage. Conclusions: While adopting Kotlin requires some care from developers, its benefits seem to bring many advantages to the platform according to the developers, especially in the aspect of adopting a more modern language while maintaining the consolidated Java-based development environment.

***Index Terms*—Android, Kotlin, Java**

I. Introduction

In May of 2017, Google announced the official support for Kotlin[1] on the Android platform. Kotlin is a statically-typed programming language that runs on the Java Virtual Machine (JVM) and fully interoperates with Java. Soon after the announcement, the language grew in popularity, appearing in rankings such as the 2018 and 2019 Stack Overflow Annual Developer Surveys [7], [8]. It was also rapidly adopted in industry by companies such as Pinterest, Uber, and Pivotal, not only to build Android Apps but also for internal purposes [6].

[1]https://kotlinlang.org

Despite its rapid adoption, few studies have been conducted so far to understand how developers are dealing with the adoption of Kotlin [50], [52]. This motivates the need for research in this field to assist industry and other researchers towards a better comprehension of the technology and to guide the development of new tools and further research. Therefore, this work aims to further investigate this subject.

In this study, we aim to collect evidence to understand how developers are adopting Kotlin as an official language for Android development, their perceptions about advantages and disadvantages of using Kotlin, and the most common problems faced by them. We do that by a triangulation strategy [31] which consists of a mixed-method approach to analyse data through the mixture of qualitative and quantitative methods. Its central premise is that the use of quantitative and qualitative approaches in combination provides a better understanding of research problems that either approach alone [42].

Thus, our first research question aims at identifying the most popular problems faced by developers: **RQ1.** *What are the most common problems faced by Kotlin developers on Android Platform?* Since interoperability with Java is one of the greatest benefits of Kotlin, we also investigate it: **RQ2.** *How are Android developers dealing with the Java-Kotlin interoperability?* Another stated benefit of Kotlin is the functional programming paradigm available in the language. However, functional programming can be challenging for developers, which leads us to our third research question: **RQ3.** *How are Android developers dealing with the functional paradigm in Kotlin?* The creation of a new programming language brings the need for proper tool support. We investigate this in our fourth research question: **RQ4.** *How are Android developers dealing with the development environment tools available for Kotlin?* Finally, we want to understand developers' perceptions about Kotlin adoption. The goal of this research question is to investigate the *user experience* insights: **RQ5.** *What is the perception of Android developers about Kotlin adoption?*

Previous works have proposed methods and criteria for evaluating programming languages [9]–[11], however, no consensus has emerged since most of them are prone to subjective assessment. In this work, we use the concurrent triangulation strategy [31] to mix different methods and enrich our results. One method is based on two previous works [12], [13], and

consists of data analysis from Stack Overflow questions using LDA [27], a statistical topic modeling technique, to automatically discover the main topics discussed. The other method uses basic qualitative strategies [14], collecting data through seven semi-structured interviews with Android developers.

Our main findings indicate that developers seem to find the language easy to understand and to adopt due to Java interoperability, null-safety guarantee, less verbosity, and functional programming support. However, they face problems regarding interoperability, automatic conversion from Java to Kotlin in Android Studio IDE, and the degradation of the readability due to the overuse of functional programming. Therefore, companies and developers that are considering Kotlin, may benefit from our results, as well as researchers interested on improving techniques, tools, processes, and mechanisms to make the best use of the language on the Android platform.

II. Background

In this section, we briefly introduce Kotlin and its role in Android development. We also present related work.

A. The Kotlin Programming Language & Android

Kotlin was created by JetBrains[2] in 2010 to improve the programming experience for the JVM [15]. It is a multi-paradigm language, supporting both object-oriented and functional programming paradigms, *i.e.*, it allows developers to combine them, as with most modern languages nowadays [18]. Its support for non-nullable types might make applications less prone to null pointer exceptions. It also includes smart casting, higher-order functions, and extension functions [17], [19]. The documentation [40] also states that the language follows the principle of pragmatic evolution, according to three main aspects: (i) to keep the language modern over time; (ii) to keep in constant feedback loop with the users, and (iii) to make updating to new versions comfortable for users.

Kotlin adoption rapidly increased after its v1.0 release in 2016. Figure 1 shows the increase on Stack Overflow[3] questions containing the `kotlin` tag, as well as on the number of GitHub[4] repositories using Kotlin. This growth also follows Google's official Kotlin support announcement in 2017. A potential reason for such popularity is the full interoperability with Java, as both languages can be freely mixed and migration can be gradual. Developers can also take advantage of the Java environment using all its existing libraries and frameworks [16], [19], while taking advantage of modern language features [39]. Moreover, its adoption by the Android community [20], [21] was reinforced by the Stack Overflow developer survey, which showed Kotlin among the most loved programming languages in 2018 and 2019 [7], [8]. We also observe the growth of Stack Overflow questions about Android development with Kotlin. Table I indicates that Kotlin was used in Android development even before its official support. Afterwards, with the default support for the language in Android Studio—the official IDE for building Android apps—the growth in questions has dramatically increased.

[2]https://www.jetbrains.com/
[3]https://stackoverflow.com
[4]https://github.com

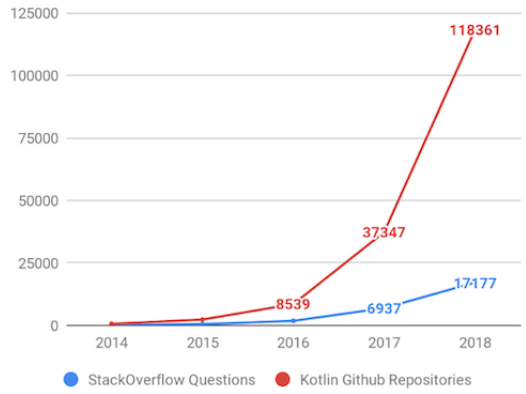

Fig. 1. Stack Overflow Questions vs. Kotlin Repositories.

TABLE I
KOTLIN WITH ANDROID YOY GROWTH IN STACK OVERFLOW.

Year	Number of questions[a]
2018	9,405
2017 (Official support in Android	3,071
2016 (Kotlin v1 release)	506

[a]These numbers are only for Kotlin regarding Android Development

B. Related Work

Mateus and Martinez [50] investigated the degree of Kotlin adoption on the development of open-source Android apps, *i.e.*, they measured the proportion of Kotlin code within those systems. They also measured and compared the quality of applications using Kotlin against those only using Java. Their results show that Kotlin is present in 11.26% of applications, and also that introducing Kotlin improves quality with regard to code smells. They also conducted a second study to investigate how Kotlin features have been used [51]. They analysed different aspects, such as which features are adopted by developers, the degree of such adoption, when they are added into the code, and so on. By inspecting the source code of open-source apps, they found that 15 out of 19 features are used in half of Kotlin applications, and the most used features are type inference, lambda, and safe call.

Coppola *et al.* [52] evaluated Kotlin adoption in open-source Android apps, investigating whether it impacted in their success. They analysed the relationship between the presence of Kotlin code in apps and their popularity. They found that almost 20% of the applications adopted Kotlin, and about 12% have more Kotlin code than Java. Furthermore, apps using Kotlin have higher popularity on average.

Banerjee *et al.* [38] conducted a study of various features of both Java and Kotlin. They concluded that Kotlin is safer and more concise than Java, but it suffers from the lack of community support material.

Flauzino *et al.* [18] analysed 100 GitHub repositories (50 Kotlin/50 Java), with respect to five code smells. The findings support the hypothesis that Kotlin presents fewer code smells than Java according to descriptive statistics, except for long parameter list. They believe that conciseness may have been responsible for the result, as they identified that smells are correlated with lines of code.

Schwermer [23] evaluated Kotlin and Java performance using benchmarks from the Computer Language Benchmarks Game suite,[5] using metrics such as memory consumption, and bytecode n-grams. The results show that Kotlin is slower than Java for all benchmarks, and that there is an underlying garbage collection overhead when reclaiming Kotlin objects. Furthermore, Kotlin often produces larger bytecode than Java.

Despite the fact there are already some studies about the adoption of Kotlin [18], [23], [38], [50]–[52], they are focused on the quantitative aspects of the language, such as proportion of Kotlin code or relation with code smells. In this study, our concern is related to qualitative aspects of Kotlin's adoption, such as the problems faced by developers and how they are dealing with some Kotlin programming aspects. To the best of our knowledge, this is the first study in this regard. Nonetheless, several studies focused on investigating Q&A websites. Barua *et al.* [13] analysed the textual content of Stack Overflow discussions using Latent Dirichlet Allocation (LDA) [27]. This approach was also used by Rebouças *et al.* [12] to assess the adoption of Swift. The latter also combined a qualitative analysis to cross-validate the results. A similar strategy is used in this study to evaluate Kotlin adoption.

III. Methodology

Our methodology is based on a concurrent triangulation strategy [31]. As Easterbrook *et al.* state, triangulation is motivated by the fact that often "what people say" could be different than "what people do", and thus, collecting data from multiple sources helps increasing the study validity [31]. So, (1) we analyse developers' discussions about Kotlin for Android on Stack Overflow (Section III-A), and (2) conduct a qualitative study by interviewing Android developers (Section III-B). The results for **RQ2**, **RQ3**, and **RQ4** were obtained from the combination of both sources (1 and 2). To answer **RQ1** we rely on the Stack Overflow data, while **RQ5** uses the interviews. All of our steps and data are publicly available.[6]

A. Stack Overflow Data Analysis

The first part of this study aims to analyse Stack Overflow with the specific goal of uncovering the main topics discussed about Kotlin and Android. We reproduced the method from Barua *et al.* [13], using LDA to analyse textual content. The goal is to automatically discover the most frequent topics present in developers' discussions [27]. We used LDAvis [25], a web-based interactive visualization of the estimated topics.

1) Acquiring and Pre-processing Data: We used the StackExchange Database[7] to extract Stack Overflow questions[8] related to Kotlin and Android. We retrieved questions containing the combination of 'kotlin' and any Android-related tags. By related tags we mean not only the explicit 'android' tag but also platform-related tags such as: 'fragments', 'recyclerview', 'intent', 'retrofit', among 14 others. We did not consider associated answers or comments, only the questions. One might argue that tagging is a manual process, which could incur in incorrectly tagged questions. However, Stack Overflow has an autocomplete feature for tagging to reduce the probability of using a misleading tag. A total of 9,405 questions were found.

Before applying topic modeling with LDA in our corpus, we performed pre-processing steps. (1) Removal of the content inside the `<code>`, `<pre>` and `<blockquotes>` tags; (2) HTML tag removal; (3) URL removal; (4) Stop words and one-letter-words removal; (5) Stripping suffixes of the remaining words using the Porter stemming algorithm [26]. For instance, after all these steps the following text:

> *" How can I make a phone call or dial a number in Android Kotlin? For example: Call <code>*21*2#</code>
"*

is reduced to:

> *"make phone call dial number android kotlin exampl call"*

2) Topic Modeling and Visualization: Since manual inspection of 9,405 questions would require too much effort and time, we use LDA, which is a flexible generative probabilistic model for collections of discrete data. In our analysis, each Stack Overflow question, *i.e.*, composition of the title and body, is considered as one document for LDA, so that each word in the document can be associated with one or more topics with a certain weight for each topic.

LDA has several parameters. The first is the number of resulting topics. This value depends on the needed granularity, so we chose 15 after testing values ranging from 5 to 20. The second is hyperparameter optimization: it allows the model to better fit the data by allowing some topics to be more prominent than others. We used 20 after testing values between 10 and 100. Lastly, we used 1,000 as the number of sampling iterations, which should be a trade off between execution time and the quality of the topic model. To support the analysis, we used LDAvis, an interactive visualization of LDA-estimated topics [25]. It also proposes a novel measure, relevance, by which to rank terms within topics to help on interpretation.

B. Basic Qualitative Study - Interviews

In a basic qualitative study, data can be collected through different forms. In this study, we conducted semi-structured interviews [14], [28] with Android developers.

[5]https://benchmarksgame-team.pages.debian.net/benchmarksgame

[6]https://github.com/victorlaerte/kotlin-adoption-analysis

[7]https://archive.org/details/stackexchange

[8]The publication date of the Stack Overflow dump is 2019-03-04.

TABLE II
CHARACTERISTICS OF RESEARCH PARTICIPANTS.

Id	Age	Role	Development years		
			Total	Android	Kotlin
D1	29	Mobile Developer	6	5	3
D2	27	Mobile Engineer	6	1.5	0.75
D3	25	Android Test Analyst	5	0.66	0.66
D4	34	Mobile Leader Engineer	12	9	1
D5	30	Mobile Consultant	6	5	2
D6	28	Mobile Engineer	5	1.33	0.33
D7	38	Mobile Developer	7	7	2

1) Sample Selection: The sample selection strategy adopted was *intentional*, based on the assumption that researchers choose the most appropriate sample to learn about the investigated phenomenon [29], and *convenience-based*, a non-probability sampling where the sample is taken from a group of people easy to contact or to reach [49]. We selected a group of individuals with the purpose of meeting specific prescribed criteria: (1) Previous experience with Android and Kotlin development; (2) No restrictions on the experience level of the developer; (3) Fluency in English or Portuguese (the languages in which researchers are fluent).

Table II presents some characteristics of the interviewees. The participants include people with many experience levels and different roles such as developers, engineers, testers, and team leaders, distributed in two different countries. All of them were selected according to the previously established criteria, assessed from a volunteer form sent in typical communication channels for developers: email groups, Slack and Facebook communities, and the official Kotlin forum.Seven professionals were interviewed, and voluntarily participated in the research. During the interviews, the participants reported their experiences with Kotlin development on Android.

2) Data Collection: We collected data through semi-structured interviews [29] with seven professional developers. In total, 215 minutes of interviews were recorded. Three interviews were conducted in person, while the remaining ones were conducted via video-conference. Three professionals are Android Specialists, three others consider themselves as Android developers, but also work with iOS or hybrid frameworks. One works with Android test automation. On average, they have 6.71 years of software development experience (SD 2.24), 4.21 years of Android development experience (SD 2.93), and 1.39 years of experience with Kotlin development (SD 0.89). Six of them have strong experience with Java.

The interviews were grounded in our research questions. We started by asking their background in software development. Then, we moved to questions about Kotlin, Java-Kotlin Interoperability, Functional Programming, and Development Tools. All interviews were recorded and transcribed *ipsis litteris*. The interview guide is publicly available.[9]

3) Data Analysis: The qualitative analysis was based on the open and axial coding. Corbin and Strauss [30] define the coding task as the process of making notations next to bits of data that strike you as potentially relevant for answering your research questions. The research steps were assisted by the qualitative data analysis software, MAXQDA 18.2.0.

The open coding of the transcripts was performed with the selection of text segments relevant to the research. The first author conducted the coding process, with support from the second. The third author helped to perform the review process. Codes were generated using an iterative approach, for each interview, and constantly compared to each other, both within the same interview and between interviews, to identify similarities and differences. They were then grouped into categories (axial coding) [30]. The names of our categories were derived from a mix of sources: (i) the researchers, (ii) the literature, and (iii) Kotlin and Android documentation.

4) Ethics: To follow research ethics regulations, all participants agreed with a consent form following the Research Ethics Board constraints. All of the consents were previously sent to the interviewers by email and the agreement is registered in the recorded interview.

[9]https://github.com/victorlaerte/kotlin-adoption-analysis/blob/master/interview/INTERVIEW-SCRIPT%20(en).md

TABLE III
LDA RESULTS.

Topic Name (Mallet)	Questions (%)	Topic Weight[a]
General Questions (14)	4,907 (52,17%)	0.36189
Java–Kotlin (13)	3,402 (36,17%)	0.23409
Classes, Objects, and Methods/Functions (4)	2,813 (29,90%)	0.19012
Build–Compilation (10)	2,038 (21,66%)	0.13145
UI–Layout (7)	1,712 (18,20%)	0.10986
UI–Navigation (11)	1,528 (16,24%)	0.0956
Data Types and Structures (5)	1,440 (15,31%)	0.09439
Google/Android Components (1)	1,388 (14,75%)	0.08868
Background Tasks (6)	1,373 (14,59%)	0.08586
UI–Data View Layouts (3)	1,199 (12,74%)	0.07408
Connectivity (0)	1,140 (12,12%)	0.06363
Multimedia Handling (2)	908 (9,65%)	0.05532
Data Storage (12)	867 (9,21%)	0.05353
Dependency Injection (8)	797 (8,47%)	0.04669
Testing (9)	510 (5,42%)	0.02913

[a]The table is sorted by topic weight in descending order
[b]LDAvis column is used to identify topics in Figure 2

IV. RESULTS

In this section, we describe the general results of the Stack Overflow data analysis and the qualitative basic study. We then discuss each research question.

A. General Results

Table III summarizes the 15 generated topics, after 1,000 iterations. The first column represents the topics already labeled, and its Mallet identifier within the parentheses. The second column shows the number of questions inside the topic. Questions can be associated with more than one topic. Hence, to compute this metric we did not take into consideration questions with weight lower than 10% for that topic. Topics are sorted by their weight (third column).

TABLE IV
OPEN CODE ANALYSES.

Category	Topic	Codes	%
Language Paradigm and Style	Functional Programming	17	9.14
	Less Verbose/More Concise	15	8.06
	Pragmatic Evolution	7	3.76
	Multi-Paradigm Language	6	3.23
	Modern Language	5	2.69
	Programming Style	5	2.69
Tools	Android Studio	15	8.06
	Code Hints	8	4.30
	Gradle	7	3.76
	Code Convertion	6	3.23
Java Interop	Soft/Optional Migration	8	4.30
	Calling Kotlin from Java	8	4.30
	JVM Annotations	6	3.23
	Constraints to Keep Interop	3	1.61
	Constructor Overloading	3	1.61
	Calling Java from Kotlin	1	0.54
Performance, Productivity, and QA	Readability/Legibility	13	6.99
	Performance/Productivity	9	4.84
	QA	5	2.69
Classes and Objects	Null Safety/Optionals	9	4.84
	Scope Functions	4	2.15
	Companion Object	3	1.61
	Extension Functions	2	1.08
	Collections	1	0.54
	Selead Classes	1	0.54
Documentation	Stack Overflow	5	2.69
	Oficial Documentation	3	1.61
Similarities between Kotlin and Swift		11	5.91
		186	**100%**

Figure 2 shows the LDAvis output for our topic model with relevance adjusted to $\lambda = 0.6$ [25]. The left panel shows topics as circles in which the area is proportional to the relative prevalence of the topic in the corpus. When a circle is selected, the right panel shows the most relevant terms. As there is no selected topic in Figure 2, it only shows the overall term frequency for all topics. Selecting a term reveals the conditional distribution over topics for the selected term. These interactions can be checked online.[10]

Table IV summarizes the interview results. The first column represents the main categories, in which we group topics in the second column. The third column represents the number of codes, *i.e.*, the number of text segments that compose each topic. It is followed by the percentage over all codes. The last category does not have subdivisions.

B. RQ1 - What are the most common problems faced by Kotlin developers on Android Platform?

To answer **RQ1**, we rely on the Stack Overflow data analysis. Table III summarizes the topics asked about Kotlin and Android, named with the support of LDAvis.

Among the topics with highest weights, we have ***(i) General Questions*** — this topic consists on general questions about Kotlin and the Android platform, such as asking for a solution given a stack trace log or general-purpose questions, *e.g.*, *"What are the advantages of Kotlin programming language?"* ($\mathbb{Q}170$). The ***(ii) Java–Kotlin*** topic contains questions on converting Java code to Kotlin and also interoperability, *e.g.*, *"Can we build Kotin and Java Mix application?"* ($\mathbb{Q}5187$).***(iii) Classes, Objects, and Methods/Functions*** addresses issues related to the structure of classes, objects, and methods in Kotlin. It contains questions that refer to using properties, lambdas, constructors, or static attributes, *e.g.*, *"In which situation val/var is necessary in Kotlin constructor parameter?"* ($\mathbb{Q}1624$). ***(iv) Build–Compilation*** consists mainly of questions related to Gradle and Android Studio, but also problems with versioning, libraries, modules, that lead to compiling failures, *e.g.*, *"Android 3.1 build gradle 4.4 error occurred configuring project ':app'"* ($\mathbb{Q}4400$). Finally, in the ***(v) UI–Layout*** topic, we find questions related to layout construction, such as the use of visual Android components and custom layout creation, *e.g.*, *"How add TextView to View in kotlin"* ($\mathbb{Q}9383$).

With medium weight, ***(vi) UI–Navigation*** also consists on UI questions, but those related to the Android navigation mechanisms, such as fragments, activities, tab and drawer layouts, *e.g.*, *"Android, how to replace initial fragment?"* ($\mathbb{Q}6504$). In ***(vii) Data Types and Structures***, we see questions about basic Kotlin data types, such as strings, numbers, and lists, *e.g.*, *"How to append 2 strings in Kotlin?"* ($\mathbb{Q}6570$). The ***(viii) Google/Android Components*** topic is composed of various questions about how Android/Google components are behaving in different devices, such as notification services, Google Play, Maps, and Firebase, *e.g.*, *"Android Notification Not Showing On API 26"* ($\mathbb{Q}Q1159$). ***(ix) Background Tasks*** refers to multi-threaded methods on Android and brings together various questions about coroutines, and related libraries such as RxJava, *e.g.*, *"Android ViewState using RxJava or kotlin coroutines"* ($\mathbb{Q}3583$). Finally, ***(x) UI–Data View Layouts*** deals specifically with components for data visualization like *RecyclerView* and *ListView*, *e.g.*, *"How to show single item selected in recyclerview using kotlin"* ($\mathbb{Q}2808$).

The topics with lowest weight are ***(xi) Connectivity***, *e.g.*, *"Retrofit parse result in Kotlin"* ($\mathbb{Q}3839$), ***(xii) Multimedia Handling***, *e.g.*, *"How to save captured photos as jpg files on android camera2"* ($\mathbb{Q}5096$), ***(xiii) Data Storage***, *e.g.*, *"How to make primary key as autoincrement for Room Persistence lib"* ($\mathbb{Q}773$), ***(xiv) Dependency Injection***, *e.g.*, *"Dagger2 + Kotlin: lateinit property has not been initialized"* ($\mathbb{Q}5412$), and ***(xv) Testing***, *e.g.*, *"How can I run a single Android Test using Kotlin?"* ($\mathbb{Q}324$).

> ***RQ1 Summary:*** Although developers suffer from problems common to all Android developers, there are specific issues regarding code conversion, Java–Kotlin interoperability, how Kotlin class members work, and compilation problems.

C. RQ2 - How are Android developers dealing with the Java-Kotlin interoperability?

We study Java—Kotlin interoperability as this is one of the main reasons given by Kotlin creators for its adoption. Furthermore, another reason is that although the languages

[10]https://www.victorlaerte.com/kotlin-adoption-analysis/#topic=0&lambda=0.6&term=

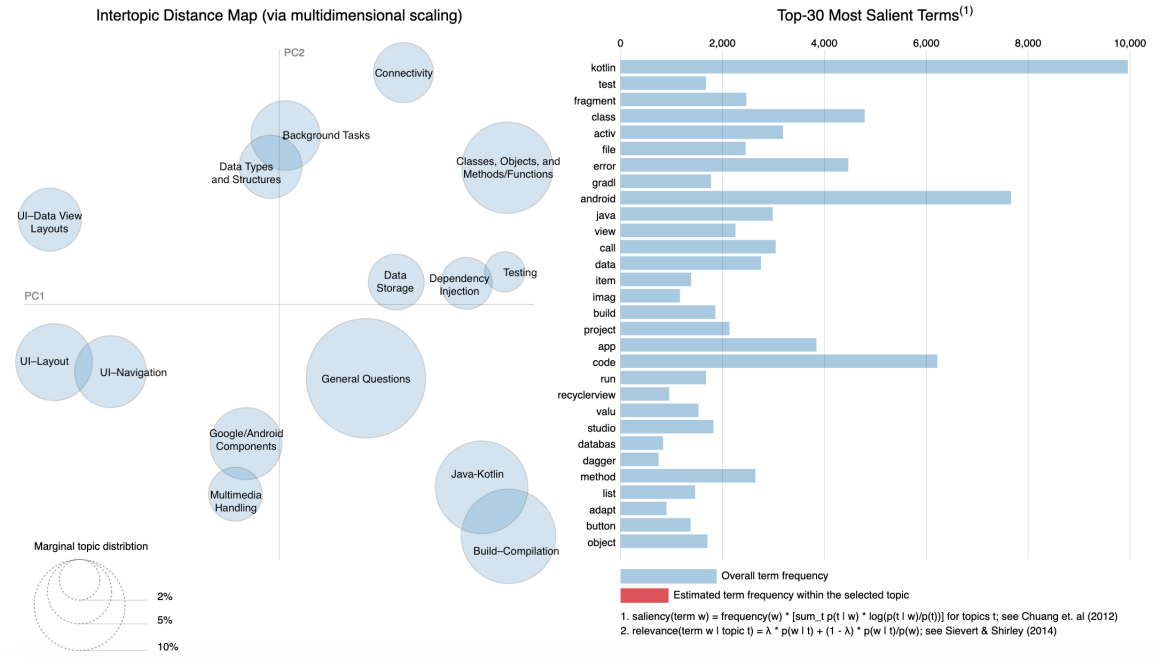

Fig. 2. LDAvis: global topic view on the left, and the term bar charts on the right.

interoperate, they have distinct characteristics such as native null safety support in Kotlin. Therefore, these differences might lead to interesting results. To answer this **RQ**, we first analysed the LDA results and then went deeper into the subject by conducting interviews to collect further evidence.

According to the LDA analysis, the most related topic is ***Java–Kotlin***. Nonetheless, due to LDAvis visualization, it was possible to see that the *'java'* term is also relevant to other three different topics, namely ***Classes, Objects, and Methods/Functions***, ***Build–Compilation***, and ***Data Storage***.

We selected and manually investigated the 100 top questions of each topic to double check the classification. During this analysis, we removed the ***Data Storage*** topic due to the many number of false-positives, resulting in 300 questions in total. After examining the title, body, and associated tags of these questions, we ended up with questions regarding problems related to new topics, such as discussed below. Besides those topics, there were also general problems that we did not address due to space constraints and over-generality.

- **Optionals:** In this category, we group questions regarding problems and complaints with optionals and null pointer exceptions crashing apps, *e.g.*, *"Non-null assert is needed even after checking is not null in kotlin"* (Q3853).
- **Properties:** Here we group questions on how Kotlin represents Java *getters* and *setters* as properties, *e.g.*, *"Why do some Java setter methods automatically become Kotlin properties but some don't?"* (Q2698).
- **Generics:** This category groups questions about using Generics in Kotlin such as the question Q3215, where the user did not understand the differences in declaration-site variance for Generics between Java and Kotlin.
- **Static Objects:** The usage of static objects between Kotlin and Java leads to questions such as: *"How to access static variable of java class in kotlin?"* (Q6396).
- **Android API and Libraries Compatibility:** These are mainly questions about using parts of Android API and libraries that still do not provide the best support for Kotlin, such as the use of support annotations, or the Intent constructor (Q4171).

Regarding the interviews, all interviews pointed to interoperability as an advantage of using Kotlin. Some of them drew attention to the possibility of adopting Kotlin without the need for full codebase migration.

> *"There are complex things already written in Java, so you don't have to go out redoing it all just because you're writing a new app in Kotlin."* (D1)

Some of them also pointed out that to keep full interoperability, there are constraints that the language has to follow.

> *"(...) to be interoperable with Java, there are some deficiencies in the language that I don't like, something that it doesn't support but I would like to have it supported. Like different support for generics, for example."* (D4)

Optionals and the difficulty to deal with nullable objects are among the main problems reported by the interviewees.

> *"There's no null safety in Java, so you can simply call a code that has no nullability control inside Kotlin passing something that is null and it will crash."* (D2)

Other reported problems are related to static objects that in Kotlin need to be declared in a companion object and

annotated as `@JvmStatic` to be called as static objects in Java, and also overriding methods in Kotlin:

> *"Companion Object. I think this is ugly and could have been abstracted for the developer."* (D1)

RQ2 Summary: Despite experiencing problems with some of the language features, developers seem to consider interoperability as a great benefit of adopting Kotlin.

D. RQ3 - How are Android developers dealing with the functional paradigm in Kotlin?

Functional programming is one of the benefits that the language creators argue for its adoption [22]. Adopting a functional programming language might be challenging even for experienced programmers, as shown by Chambers *et al.* [32]. Hence, the need to understand how developers deal with the new possibilities that the paradigm brings to the platform.

To answer this question, we proceeded similarly to **RQ2**. We selected 100 questions from each topic with terms related to the functional paradigm according to LDAvis. The topics were ***Classes, Objects, and Methods/Functions***, ***General Questions***, and ***Background Tasks***. During the analysis of these questions, we removed the ***Background Tasks*** topic due to the high number of false-positive questions, resulting in 200 questions in total. After the manual process, we ended up with questions regarding functional programming problems as detailed below, as well as general problems.

- **Higher-order Functions/Lambdas:** This category contains questions about using lambdas and higher order functions, *e.g.*, *"How to pass a function as parameter in kotlin - Android"* ($\mathbb{Q}$6181).
- **Closures:** Questions regarding the scope of enclosing functions, *e.g.*, *"How to make 'this' a reference of Listener instead of the Activity in Kotlin?"* ($\mathbb{Q}$3432).
- **Scope Functions:** This category considers special functions provided by the standard library, such as `let`, `run`, `with`, `apply`, and `also`, whose sole purpose is to execute a block of code within the context of an object [19]. For instance, *"difference between kotlin also, apply, let, use, takeIf and takeUnless in Kotlin"* ($\mathbb{Q}$2698).
- **First Class Functions:** These are questions about using function as first class citizens, *i.e.*, assignee functions to variables or storing them in data structures, *e.g.*, ($\mathbb{Q}$1585).

During the open coding process, we coded 17 segments from the interviews as related to Functional Programming. Five of the interviewees consider Kotlin as fully supporting the functional paradigm, two of them partially or did not responded to these questions due to lack of experience in the subject. Some respondents mentioned functional programming as a factor that enhances the flexibility and modernity of language.

> *"I think Kotlin supports functional programming. Everything from the collections is very functional programming oriented. Lambdas, functions as first-class citizens, having functions apart from classes, all those things I think that help to program in a functional programming way very easily"* (D4)

When asked about the problems associated with the functional paradigm, two interviewees mentioned difficulties to understand *scope functions*, and one mentioned several problems using *inline functions*.

> *"I remember having to look for several problems I found compiling with inline functions. We were using inline functions a lot, because of the reified generics and I had to search for solutions to several problems with it."* (D4)

Despite the fact that the interviewees consider the functional paradigm as helpful in code readability, they also state that overusing the paradigm might result in the opposite effect.

> *"Lambdas are things that make our day to day easier for coding, but can also make it difficult to read."* (D7)

RQ3 Summary: Although developers consider the functional paradigm as a factor that enhances the flexibility and modernity of the language, they are facing problems regarding the usage of lambdas and closures. They also consider that its overuse can decrease code readability.

E. RQ4 - How are Android developers dealing with the development environment tools available for Kotlin?

We study development tools because, as mentioned by Breslav [15], creating a new programming language is more than creating a compiler. A whole ecosystem of supporting tools needs to be created, to foster adoption. Hence, we believe that by assessing the tools available for a programming language, we are also partially assessing how Kotlin adoption is being perceived by the developers [37].

To answer this **RQ** we selected 100 questions from each of the ***Build–Compilation*** and ***Java–Kotlin*** topics. Relevant terms for such topics include 'gradle' and 'studio'. We also considered the relevance of other terms that refer to tools, such as 'plugin', 'librari', 'proguard' and 'kapt'. After analysing the 200 questions, we categorized the problems as follows.

- **Android Studio code converter:** This category groups questions on problems with the Android Studio code converter feature, that automatically translates Java into Kotlin. Most questions report compilation failures after conversion, *e.g.*, *"Converting Java file to Kotlin now it won't compile - Internal compiler error"* ($\mathbb{Q}$8770).
- **Error After Upgrade:** This category refers to problems that appear after upgrading development tools, *e.g.*, *"Android unable to build project after updating kotlin runtime to 1.2.31"* ($\mathbb{Q}$4689).
- **Version Conflicts:** This category consists of versioning conflicts, *e.g.*, *"Warning "Kotlin plugin version is not the same as library version" (but it is!)"* ($\mathbb{Q}$4784).
- **Kotlin Setup:** Here we group questions about difficulties reported by developers on setting up Kotlin in their projects, *e.g.*, *"I am unable to configure Kotlin in*

my android studio. getting error Error: Unable to find method'BaseVariantData.getOutputs()...;'." ($\mathbb{Q}$1002).

- **kapt & databinding:** The last category contains questions on problems with the Kotlin Annotation Processor (kapt) and Databinding Library. Both terms presented high relevance in LDAvis and manual analysis.

During the open coding process, we coded 36 segments as related to development tools. The majority of the reported problems relates to version upgrades of such tools. The responses point out that even minor version upgrades can cause problems that might lead their projects to stop compiling.

> *"I didn't have such a good, fluid experience [with Android Studio]. I think we've seen the project break sometimes because of updates."* (D1)

Some also report that fixing such issues is difficult because it is hard to identify the root of the problem.

> *"I think it wasn't an easy thing to understand, especially when you have multiple projects with multiple dependencies... sometimes it's not so clear what's wrong. After some time suffering you can come up with some trick or set of steps that facilitate how to solve some problems."* (D1)

Regarding Android Studio, the interviewees think there are many features that help with development in Kotlin, such as the automatic code conversion feature.

> *"I know that someone already solved that problem in Java so I can copy it, paste the code and see it converted to Kotlin. This is very cool."* (D3)

However, they point out that there is still a lot to improve on automatic code conversion and code suggestions.

> *"...the code translation between Java code can be improved to generate code that is more idiomatic. It's working, but it's creating code that it's not following the best practices of Kotlin."* (D4)

RQ4 Summary: Developers face many problems with development tools, specially regarding the Android Studio code converter, Gradle, and new library versions. Interviewees report that even minor upgrades can lead to problems.

F. RQ5 - What is the perception of Android developers about Kotlin adoption?

The goal of this **RQ** is to investigate the developers' perception about Kotlin adoption in Android development through the analysis of the interviews. In this section, we address the most relevant topics identified in the open coding process (Table IV) which were not previously analysed. In this **RQ**, we decided not to address topics within the *Tools*, *Interoperability*, and *Classes and Objects* categories as they were already addressed by previous questions.

In the ***Language Paradigm and Style (55)*** category, we removed the ***Functional Programming (17)*** topic from the analysis, since **RQ3** already discusses it. We analysed the remaining topics: ***Less Verbose/More Concise (15)***, ***Pragmatic Evolution (7)***, ***Multi-Paradigm Language (6)***, ***Modern Language (5)***, and ***Programming Style (5)***. As a result it was possible to identify that interviewees consider Kotlin more concise and less verbose than Java.

> *"I think it's very concise, not as verbose as Java, more elegant. I don't have to specify everything that takes a lot of time. I think the conciseness of the language is the thing I like the most."* (D4)

Developers points out that Kotlin is in constant evolution, and sometimes it is even hard to follow it:

> *"Kotlin is constantly bringing several improvements, sometimes it is even difficult to follow."* (D5)

They also consider that multi-paradigm support makes the language more flexible, which they believe is a characteristic of modern languages, as well as the new style of programming that Kotlin brings to the Android platform.

> *"...You can define functions receiving other functions, which shows that it was designed such that you can program in a much simpler way than in Java [...] Since it is a language that already was born with functional paradigm it makes simpler to, for example, define higher order functions, or extension functions, or... everything is much simpler."* (D7)

The ***Performance, Productivity and QA (27)*** category was subdivided into the following topics: ***Readability/Legibility (13)***, ***Performance/Productivity (9)***, and ***QA (5)***. It was possible to identify that developers consider the language to be more idiomatic, making it easier to read code, and improving individual performance. They also point out that such readability may deteriorate with the overuse of the functional paradigm.

> *"In my opinion, it is much easier to read a Kotlin code than a Java code for example."* (D3)

> *"Kotlin brings many functions that improve readability. For example, aggregation functions that help you a lot... iterating over a list sometimes is not so clear in Java, you have too much code that decreases understanding. Kotlin performs well in this aspect, producing very good code output. Sometimes it is also the opposite when you use too much of Kotlin's resources you can end up with code that is not so clear, for example, using too many 'let' or 'also' you can end up with hard to understand code."* (D2)

All interviewees consider that the Kotlin language leads to an improvement in the quality of the developed code, pointing to null safety as the main factor.

> *"I think that what helps developers most in Android is the null support, dealing with null references in an easy way. Also the support for lambdas, all those mini features that you can use in Android, help in code quality, especially the null support."* (D4)

The ***Documentation (8)*** category, with the ***Stack Overflow (5)*** and ***Official documentation (3)*** topics, show us that some interviewees believe that the official documentation still lacks specific information for Android Platform.

"I think what's missing for Kotlin is some very specific documentation for the platform." (D5)

They also point out the importance of Stack Overflow as a source of knowledge.

"...nowadays every developer uses Stack Overflow. It is a very large community to ask questions and solve specific problems that perhaps the documentation can not address. Kotlin has a very good, robust documentation, but it will not be able to cover all use cases and needs. Stack Overflow ends up being a tool, a very fast parallel platform for you to solve a problem. And typically, answers include examples in a very simple and practical way." (D6)

The last category, ***Similarities between Kotlin and Swift (11)***, could only be identified due to the semi-structured interview method we followed. We did not identify additional topics underlying this category. Four interviewees brought to light similarities between Kotlin and Swift (used for iOS programming). Some of them report that, as they had experience with Swift, the learning curve for adopting Kotlin was reduced, due to the similarity between languages. Another interviewee pointed that since his company needed multidisciplinary teams, they decided to adopt Kotlin because people with experience in Swift could also work well with Kotlin.

"It was more natural to me because it was very similar to Swift." (D1)

RQ5 Summary: Developers consider Kotlin a modern language, which helps to improve productivity and quality. The similarity with Swift also helps. They claim that documentation still lacks Android-specific material.

V. Discussions

This section discusses the results, with an initial overall assessment, followed by comparison to the existing literature, discussion of implications, and threats to validity.

A. Overall Assessment

Developers consider Kotlin a modern language with many features that facilitate its adoption. Being able to use a new language without having to undergo an abrupt migration is an advantage pointed out by all interviewees. In fact, none of them cited a case where their code base was fully migrated to Kotlin. In most cases they were partially migrated, or only new features were written in Kotlin. Some of them also stated that being able to use the entire Java API also helps. We found many Stack Overflow questions on problems using null variables from Java, which lacks nullability control, which was also reported by the interviewed developers.

The multi-paradigm support is reported by respondents as a feature that brings more flexibility to development. However, they also warn that overuse of the functional paradigm can make the code more complex and difficult to understand. This problem also appears in the Stack Overflow questions, where it is possible to find questions regarding scope context within closures and the use of higher-order functions. Additionally, many Stack Overflow questions pertain to problems in the toolchain, specially with upgrading, as well as hard to understand or unhelpful error messages.

According to the interviewees, Kotlin contributes to improving productivity and performance because it is a modern language, less verbose, and improves readability when compared to Java (only when the functional paradigm is not overused). It also brings more quality to the code produced. They also point out that a different mindset, or programming style, is required in order to use Kotlin's features in the best way possible.

Regarding documentation, developers believe that much can still be done to improve the support material of the language and point out Stack Overflow as a platform of utmost importance to the developer community. Finally, developers report that similarities between Kotlin and Swift help on its adoption. One interviewee states that adopting it can greatly benefit multidisciplinary teams, especially in a startup environment, where, according to him, due to scarce resources, many developers have to work on both Android and iOS.

B. Literature Discussion

In this section, we compare our results with works mentioned in Sections I and II to identify similarities and differences. According to JetBrains [16], [17] and Google [20], the reason for Kotlin's popularity is the flexibility, modernity, and the possibility to fully interoperate with Java. Our results provide evidence to this being the main factor for its adoption, followed by flexibility and modernity. Nonetheless, we also observe that although interoperability is a facilitator, it also might lead developers to face problems, being the second topic with more questions and reported by interviewees.

JetBrains [16], [17], and Google [20] also state that smart casting, higher-order functions, and extension functions allow developers to focus on making code more readable and less verbose. This statement is partially supported by this research because despite developers considering Kotlin as a language that improves readability, they believe that overusing functional capacities and operators such as the *elvis operator*, can cause the opposite effect, making code difficult to read and understand. This might be explained by Chambers *et al.* [32] who conducted a long-term observational study to understand how experienced imperative programmers performed in an introductory graduate course on functional programming. They reported that students commonly encounter several conceptual difficulties when learning functional languages, specially implementing recursive functions, and nested operations. Other studies also present similar results where functional programming demonstrate several challenges to learners such as understanding the concept of higher-order functions [33], the type of functional expressions [34], and the evaluation of iterative and recursive functions [35], [36].

Shafirov [22] affirms in the official Kotlin Blog that developers do not need to install any extra plugin or worry about compatibility since JetBrains and Google improved Kotlin support in the Android Studio IDE after its official announcement

as an Android supported language. However, developers are still facing problems regarding version compatibility of their development tools, specially for Android Studio and Gradle.

In a large-scale empirical study with GitHub repositories involving more than 6 million lines of code, Flauzino *et al.* [18] identified that Kotlin code usually had fewer code smells than Java code. The same finding is presented by Banerjee *et al.* [38]. Our study found similar results where developers believe that Kotlin improves code quality, especially because of the null-safety guarantee, scope functions, and lambdas.

Although Rebouças *et al.* [12] studied Swift, the similarity between both languages came to light throughout our work. By comparing their findings with ours, we identify that in both studies the interviewees find languages easy to adopt. Another related finding is that optional variables seem to cause problems in both languages. Developers also report problems with tools, which can be explained by the tools being recent. Unlike their work, we did not find major problems with the compiler, besides compilation time.

C. Implications

As a general implication of this work, we provide an empirical evidence for companies, researchers, and developers who want to make a preliminary analysis before adopting Kotlin. For those who have already adopted the Kotlin language, it might be relevant for improving techniques, tools, processes, and mechanisms to make the best use of the language on the Android platform.

Regarding **RQ1**, companies and developers can know in advance what kind of problems they will face if they decide to adopt Kotlin. With the support of **RQ2** they can also understand the advantages and disadvantages of interoperating Java and Kotlin, so they might be better equipped to choose between a partial or full Java—Kotlin migration. In **RQ3** we bring a better understanding of the functional paradigm, so researchers can use these results for studying if the usage of Kotlin in programming disciplines can help the understanding of the functional paradigm. Toolmakers can take advantage of the findings presented in **RQ4** to create new tools and improve existing ones. Finally, in **RQ5** companies, developers, and researchers can understand how Kotlin adoption has been perceived by the developers, taking into account productivity, readability, and code quality to create processes and mechanisms to maximize gains and mitigate losses.

D. Threats to Validity

The first threat to validity 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 fit our study. Even though, we had to label the topic 2 (***Java–Kotlin***) with a more generic name due to the diversity of questions that the topic presented, especially regarding interoperability and code conversion, manual or automatic. Despite of that, we believe this is not a limitation of our work due to the fact that we approached the topic in detail in other research questions, **RQ2** and **RQ4**. This can also be explained by the fact that the most difficult part to migrate the codebase from Java to Kotlin is how to deal with null variables from Java. Secondly, due to the high number of questions, we manually analysed only 700 questions. However, this number exceeds the minimum of 564 required to achieve a 95% confidence level with 4% of error margin for the population of 9,405 questions analysed in this study [48]. We also used LDAvis analyses[11] to support and corroborate our manual analyses. We also face as a threat the fact that not all of the Stack Overflow questions regard problems, but some of them are just questions asking for best practices, implementation examples or performance benchmarks. However, in our manual analysis, it was observed that the vast majority of questions of this type appear in the topic *General Questions*, while other topics are more heterogeneous. Finally, our interview script might not have covered all possible questions. However, the interviews were designed as semi-structured. Hence, this allowed us to cover other kind of questions during the interview that were not listed in the script.

VI. Conclusion

In this study, we investigated how developers are dealing with the recent adoption of Kotlin as an official language for the Android Platform by performing an in-depth analysis of Stack Overflow questions related to Kotlin and Android, and interviewing seven developers. Our results indicate that developers seem to find the language easy to understand and to adopt. They believe that the use of Kotlin can improve code quality, readability, and productivity. Among the main factors are: Java interoperability, the null safety guarantee, the less verbosity, the lambdas and higher-order functions, and the good support in the IDE. However, even with all the advantages that Kotlin has brought to the Android platform, developers continue to face many problems using it, which includes the use of null variables and optionals within Kotlin, the problems with the toolset, the degradation of readability with the overuse of the functional paradigm, and the interoperation with Java. Finally, the main finding of this study is that developers consider that Kotlin adoption brings many advantages to the Android platform, specially in the aspect of adopting a more modern language while maintaining the consolidated Java-based development environment.

We believe this study serves as a starting point for understanding Kotlin adoption, and more similar studies are needed to offer proper developer decisions support.

VII. Acknowledgments

We acknowledge support from FACEPE (BCT-0229-1.03/19 and APQ-0570-1.03/14), CAPES (88887.333966/2019-00), and CNPq (409335/2016-9). This research was partially funded by INES 2.0, FACEPE grants PRONEX APQ-0388-1.03/14 and APQ-0399-1.03/17, and CNPq grant 465614/2014-0. We also thank Tempest Security Intelligence for the research grant support for attending SANER 2020.

[11]https://www.victorlaerte.com/kotlin-adoption-analysis/#topic=0&lambda=0.6&term=

REFERENCES

[1] "The World's Most Valuable Brands". Forbes, 2018. Available: https://www.forbes.com/powerful-brands/list/.

[2] "Global Mobile Market Report - Free Version". Newzoo, 2018. Available: https://resources.newzoo.com/hubfs/Reports/Newzoo_2018_Global_Mobile_Market_Report_Free.pdf.

[3] "Use Java 8 language features". Android Developers, 2017. Available: https://developer.android.com/studio/write/java8-support.

[4] "Compiling with Jack". Android Open Source Project, 2017. Available: https://source.android.com/setup/build/jack.

[5] J. Titus. "Developer Keynote". Google I/O, 2017. Available: https://events.google.com/io2017/schedule/?section=may-17&sid=__keynote2__.

[6] "Kotlin Language Official Website". Available: https://kotlinlang.org/.

[7] "Developer Survey Results". Stack Overflow, 2018. Available: https://insights.stackoverflow.com/survey/2018/.

[8] "Developer Survey Results". Stack Overflow, 2018. Available: https://insights.stackoverflow.com/survey/2019/.

[9] N. Wirth. "Programming languages: What to demand and how to assess them.". Symposium on Software Engineering, Belfast, 1976.

[10] S. S. Chandra, K. Chandra. "A Comparison of Java and C#". J. Comput. Small Coll., 20(3):238254, 2005.

[11] F. Schmager, N. Cameron and J. Noble. "GoHotDraw: Evaluating the Go programming language with design patterns". In PLATEAU, pages 10:110:6, 2010.

[12] M. Rebouças, G. Pinto, F. Ebert, W. Torres, A. Serebrenik, and F. Castor. "An Empirical Study on the Usage of the Swift Programming Language". 2016. 634-638. 10.1109/SANER.2016.66.

[13] 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):619654, June 2014

[14] S. Merriam. "B.Qualitative Research: A Guide To Design And Implementation". San Francisco, Calif. Jossey-Bass, 2009.

[15] A. Breslav. "History of Kotlin". Kotlin for Java Developers - Coursera. Available: https://www.coursera.org/lecture/kotlin-for-java-developers/history-of-kotlin-K8pZr.

[16] "JVM Languages Report extended interview with Kotlin creator Andrey Breslav". RebelLabs, 2013. Available: https://jrebel.com/rebellabs/jvm-languages-report-extended-interview-with-kotlin-creator-andrey-breslav/.

[17] A. Breslav. "Kotlin 1.0 Released: Pragmatic Language for JVM and Android". Kotlin Blog, 2016. Available: https://blog.jetbrains.com/kotlin/2016/02/kotlin-1-0-released-pragmatic-language-for-jvm-and-android/.

[18] M. Flauzino, J. Verssimo, R. Terra, E. Cirilo, V. H. S. Durelli, and R. S. Durelli. "Are you still smelling it?: A comparative study between Java and Kotlin language". 2018. In Proceedings of the VII Brazilian Symposium on Software Components, Architectures, and Reuse (SBCARS '18). ACM, New York, NY, USA, 23-32. DOI: https://doi.org/10.1145/3267183.3267186.

[19] "Kotlin Docs". Kotlin Official Documentation. Available: https://kotlinlang.org/docs/reference/

[20] M. Cleron. "Android Announces Support for Kotlin". Android Developers Blog, 2017. Available: https://android-developers.googleblog.com/2017/05/android-announces-support-for-kotlin.html

[21] "Develop Android apps with Kotlin". Android Developers. Available: https://developer.android.com/kotlin/index.html

[22] M. Shafirov. "Kotlin on Android. Now official". Kotlin Blog, 2017. Available: https://blog.jetbrains.com/kotlin/2017/05/kotlin-on-android-now-official/

[23] P. Schwermer. "Performance Evaluation of Kotlin and Java on Android Runtime". Dissertation, 2018.

[24] Y. Shah, J. Shah, and K. Kansara. "Code obfuscating a Kotlin-based App with Proguard". 2018. 1-5. 10.1109/ICAECC.2018.8479507.

[25] C. Sievert, K. E. Shirley "LDAvis: A method for visualizing and interpreting topics. Proceedings of the Workshop on Interactive Language Learning, Visualization, and Interfaces". 2014. 63-70. 10.1109/ICAECC.2018.8479507.

[26] M. F. Porter. "An algorithm for suffix stripping". 1980. Program, Vol. 14 Issue: 3, pp.130-137, https://doi.org/10.1108/eb046814

[27] D. M. Blei, A. Y. Ng, and M. I. Jordan. "Latent dirichlet allocation.". 2003. J. Mach. Learn. Res. 3 (March 2003), 993-1022.

[28] D. I. K. Sjøberg, T. Dybå, B. C. D. Anda, and J. E. Hannay. "Building Theories in Software Engineering". 2008. Springer London, London, pp. 312336. https://doi.org/10.1007/978-1-84800-044-5_12

[29] P. Runeson and M. Höst. "Guidelines for conducting and reporting case study research in software engineering". 2008. Empirical Software Engineering 14, 2 (19 Dec 2008), 131. https://doi.org/10.1007/s10664-008-9102-8

[30] J. Corbin, and A. Strauss. "Basics of qualitative research: Techniques and procedures for developing grounded theory". 2007. (3rd ed.). Thousand Oaks, CA: Sage.

[31] S. Easterbrook, J. Singer, M.-A. Storey, and D. Damian. "Selecting Empirical Methods for Software Engineering Research". 2008. Springer London, London, pp. 285-311.

[32] C. Chambers, S. Sheng, D. Le, and C. Scaffidi. "The Function, and Dysfunction, of Information Sources in Learning Functional Programming". 2012. J. Comput. Sci. Coll. 28, 1 (October 2012), 220-226.

[33] M. Chakravarty, and G. Keller. "The risks and benefits of teaching purely functional programming in first year". 2004. Journal of Functional Programming, 14 (1), 113-123.

[34] S. Joosten, van den Berg, K. G van der Hoeven. "Teaching functional programming to first-year students". 1993. Journal of Functional Programming, 3 (1), 49- 65.

[35] A. Ebrahimi "Novice programmer errors: Language constructs and plan composition". 1994 Intl. Journal of Human Computer Studies, 41 (4), 457-480.

[36] J. Segal "Empirical studies of functional programming learners evaluating recursive functions". 1994. Instructional Science, 22 (5), 385-411.

[37] E. Murphy-Hill, S. Markstrum, and C. Anslow. "Evaluation and usability of programming languages and tools (PLATEAU)". 2010. 265-266. 10.1145/1869542.1869605.

[38] M. Banerjee, S. Bose, A. Kundu, and M. Mukherjee. "A Comparative Study: Java Vs Kotlin Programming in Android Application Development." 2018. International Journal of Advanced Research in Computer Science, 9(3), 41-45. doi:https://doi.org/10.26483/ijarcs.v9i3.5978 10.1145/1869542.1869605.

[39] R. K. Panchal, and, A. K. Patel. "A comparative study: Java Vs Kotlin Programming in Android". 2017. International Journal of Innovative Trends in Engineering Research, September 2017, vol 2 Issue 9, pp 4 10.

[40] "Comparison to Java Programming Language". Kotlin Official Documentation. Available: https://kotlinlang.org/docs/reference/comparison-to-java.html

[41] "Android KTX". Android Official Documentation. Available: https://developer.android.com/kotlin/ktx

[42] J. Creswell. "Research Design Qualitative, Quantitative and Mixed-Methods Approaches". 2003. Thousand Oaks, CA: Sage.

[43] J. Creswell, and V. Plano Clark. "Designing and Conducting Mixed-Methods Research". 2007. Thousand Oaks, CA: Sage.

[44] A. Atif, D. Richards, and A. Bilgin. "A Student Retention Model: Empirical, Theoretical and Pragmatic Considerations". 2013. Proceedings of the 24th Australasian Conference on Information Systems.

[45] J. Morse. "Principles of Mixed-Methods and Multi-Method Research". 2003. In Tashakkori, A. and Teddlie, C. (Eds.), Handbook of Mixed-Methods in Social and Behavioural Research, pp 189-208. Thousand, Oaks, CA: Sage.

[46] V. R. Basili. "Software Modeling and Measurement: The Goal/Question/Metric Paradigm". 1992. College Park, MD, USA.

[47] "Statista dossier about Smartphones". Statista. 2019. Available: https://www.statista.com/study/10490/smartphones-statista-dossier/.

[48] "Sample Size Calculator". Creative Research Systems. 2012. Available: https://www.surveysystem.com/sscalc.htm.

[49] Saumure, K. and Given, Lisa M. "The SAGE Encyclopedia of Qualitative Research Methods". 2008. SAGE Publications, Inc. Thousand Oaks. 10.4135/9781412963909. Available: http://sk.sagepub.com/reference/research.

[50] Góis Mateus, B and Martinez, M. "An empirical study on quality of Android applications written in Kotlin language". Empir Software Eng (2019). https://doi.org/10.1007/s10664-019-09727-4.

[51] Góis Mateus, B and Martinez, M. "On the adoption, usage and evolution of Kotlin Features on Android development". The Computing Research Repository (CoRR), 2019. http://arxiv.org/abs/1907.09003.

[52] Riccardo Coppola, Luca Ardito, and Marco Torchiano. 2019. Characterizing the transition to Kotlin of Android apps: a study on F-Droid, Play Store, and GitHub. In Proceedings of the 3rd ACM SIGSOFT International Workshop on App Market Analytics (WAMA 2019). ACM, New York, NY, USA, 8-14. DOI: https://doi.org/10.1145/3340496.3342759.