Pós-Graduação em Ciência da Computação

Victor Laerte de Oliveira

An Empirical Study on the Usage of the Kotlin Programming Language for Android Development

Universidade Federal de Pernambuco
posgraduacao@cin.ufpe.br
http://cin.ufpe.br/~posgraduacao

Recife
2019

Victor Laerte de Oliveira

An Empirical Study on the Usage of the Kotlin Programming Language for Android Development

Dissertation presented to the Post-Graduate Program in Computer Science of the Informatics Center of the Federal University of Pernambuco as a partial requirement to obtain the Master of Computer Science degree.

Field: Software Engineering
Advisor: Leopoldo Motta Teixeira

Recife
2019

Catalogação na fonte
Bibliotecária Mariana de Souza Alves CRB4-2106

O48e Oliveira, Victor Laerte de.
An Empirical Study on the Usage of the Kotlin Programming Language for Android Development/ Victor Laerte de Oliveira. – 2019.
87 f.: il., fig., tab.

Orientador: Leopoldo Motta Teixeira.
Dissertação (Mestrado) – Universidade Federal de Pernambuco. CIn, Ciência da computação. Recife, 2019.
Inclui referências e apêndices.

1. Engenharia de Software. 2. Linguagens de Programação. 3. Android. 4. Kotlin. I. Teixeira, Leopoldo Motta (orientador). II. Título.

005.1 CDD (22. ed.) UFPE-MEI 2019-161

Victor Laerte de Oliveira

"An Empirical Study on the Usage of the Kotlin Programming Language for Android Development"

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de Pernambuco, como requisito parcial para a obtenção do título de Mestre em Ciência da Computação.

Aprovado em: 08 de agosto de 2019.

To my family, and friends. I love you all.

ACKNOWLEDGEMENTS

A Deus por tudo que tenho em minha vida e por Sua benevolência para comigo, que tornou possível chegar até aqui com o suporte de todos que amo ao meu lado.

À minha esposa, Marianna, a qual eu pude agradecer em minha graduação ainda como namorada e hoje posso agradece-la como minha esposa e companheira por todo o incentivo na busca dos meus objetivos.

À minha família, especialmente aos meus pais Sebastião e Grace, por todo amor e sacrifícios feitos para ajudar na realização dos meus sonhos.

À minha irmã Priscila, pelo coração extremamente generoso e pelo simples fato de existir e fazer parte de minha vida.

Aos meus avós maternos Domingos e Maria, pelas inúmeras vezes que me aconselharam a estudar e procurar sempre o meu melhor. Esse trabalho é para vocês.

Agradeço ao meu Professor e orientador, Leopoldo Teixeira, pelos ensinamentos, paciência e oportunidade de me orientar, mesmo sabendo que eu teria que dividir o tempo entre o estudo e o trabalho.

Aos meus colegas Karine, Alex e Gabriel que entraram junto comigo sob a orientação do Professor Leopoldo e que mesmo não tão próximos no dia a dia conseguimos criar laços de amizade e ajuda mútua.

Agradeço também à empresa Liferay, em especial a Bruno Farache, que por muitas vezes demonstrou sensibilidade em me liberar para atividades acadêmicas e até mesmo me patrocinar em eventos.

A todos vocês que contribuíram direta ou indiretamente com a minha formação, meus sinceros agradecimentos.

"Any sufficiently advanced technology is indistinguishable from magic" (CLARKE, 1973)

ABSTRACT

In 2017, Google announced Kotlin as one of the officially supported languages for Android development. Until then, only Java and C++ were part of this list. 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. Despite Kotlin’s rapid rise in the industry, little has been done in academia to understand how developers are dealing with its adoption. This research aims to gather evidence 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. 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 StackOverflow. Concurrently, we also conducted a basic qualitative research interviewing seven Android developers that use Kotlin, to confirm and cross-validate our results. Our study reveals that developers do seem to find the language easy to understand and to be adopted. This perception begins to change when the functional paradigm becomes more evident. According to the developers, 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. Nonetheless, they also report that it can also become a challenge when interoperating with Java, despite of the interoperability being considered as an advantage. While adopting Kotlin requires some care from developers, the benefits of its adoption on Android 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.

Keywords: Programming Languages. Android. Kotlin. Java.

RESUMO

Em 2017, o Google anunciou o Kotlin como uma das linguagens de programação oficialmente suportadas para o desenvolvimento Android. Até então, apenas Java e C++ faziam parte dessa lista. Entre as razões para a escolha do Kotlin, o Google mencionou que ele é "conciso, expressivo e projetado para ser seguro em termos de tipo e variáveis nulas". Outra razão importante foi que o Kotlin é uma linguagem totalmente interoperável com Java e roda na JVM. Apesar da rápida ascensão de Kotlin na indústria, pouco foi feito na academia para entender como os desenvolvedores estão lidando com sua adoção. Esta pesquisa tem como objetivo coletar evidências para entender como os desenvolvedores estão lidando com a recente adoção do Kotlin como uma linguagem oficial para o desenvolvimento Android, sua percepção sobre as vantagens e desvantagens do Kotlin e os problemas mais comuns enfrentados por eles. Esta pesquisa foi conduzida usando a estratégia de triangulação concorrente, uma abordagem de método misto. Realizamos uma análise completa de 9.405 questões relacionadas ao desenvolvimento do Kotlin para a plataforma Android no StackOverflow. Simultaneamente, também realizamos uma pesquisa qualitativa básica entrevistando sete desenvolvedores Android que usam o Kotlin, para confirmar e validar de forma cruzada nossa análise. Nosso estudo revela que os desenvolvedores parecem achar a linguagem fácil de entender e ser adotado. Essa percepção começa a mudar quando o paradigma funcional se torna mais evidente. De acordo com os desenvolvedores, a legibilidade é comprometida quando os desenvolvedores abusam do paradigma funcional que a linguagem oferece. Os desenvolvedores também consideram que o Kotlin aumenta a qualidade do código produzido, principalmente devido às suas garantias de segurança contra tipos nulo. No entanto, eles também relatam que também pode se tornar um desafio ao interoperar com o Java, apesar da interoperabilidade ser vista como uma vantagem. Embora a adoção do Kotlin exija algum cuidado dos desenvolvedores, de acordo com os desenvolvedores os benefícios de sua adoção no Android parecem trazer muitas vantagens para a plataforma, especialmente pela possibilidade de adotar uma linguagem mais moderna fazendo uso de todo o ambiente de desenvolvimento consolidado baseado em Java.

Palavras-chaves: Linguagens de Programação. Android. Kotlin. Java.

LIST OF FIGURES

LIST OF TABLES

LIST OF CODES

CONTENTS

1 INTRODUCTION

Over the last few years, the mobile smartphone market has become one of the most powerful industries in the world (STATISTA, 2019). Apple and Google, the producers of the most popular smartphone operating systems, iOS and Android, are now the two most valuable companies in the world (FORBES, 2018). Smartphone users around the world surpassed the 3 billion mark and it could reach 3.7 by 2021 (NEWZOO, 2018). Global app revenues also show numbers that exceed the $100 billions in revenue in 2019 with projections of $139.6 billions in 2021.

These facts create a high demand not only for new mobile developers, but also for new techniques, tools, and frameworks to ease mobile programming practice. Until the middle of 2017, Java was the main development language for Android. Although well-established, Java did not have major constant releases, which decelerated its evolution and consequently the modernity of the language and the platform. In the case of Android, the landscape was a bit worse because, for a long time, the platform only supported limited features of Java 8, such as lambda expressions, through third-party libraries[1] or the Jack (Java Android Compiler Kit) (ANDROID, 2017a; ANDROID, 2017b). Kotlin[2] comes as an alternative to this context.

Kotlin is a statically-typed programming language that runs on the Java Virtual Machine and can also be compiled to JavaScript source code. It was released to the public in February 2016. Its primary development is from a team of JetBrains programmers based in Saint Petersburg, Russia. Its name comes from Kotlin Island in Saint Petersburg (PANCHAL, 2016), and it is distributed under the Apache 2 Open Source license.

Kotlin brings the possibility of fully interoperating with Java code, so it is possible to mix both languages freely. Migrating to Kotlin can be gradual and does not have to alter the entire codebase. Developers can also take advantage of the Java environment using all its existing libraries and frameworks running with the same performance level as Java, while taking advantage of modern features such as functional programming, smart casting, destructuring declarations, null safety support, extension functions, lazy loading, and others (JANGID, 2017; BANERJEE et al., 2018; PANCHAL, 2016).

All the advantages brought by Kotlin ended up creating a developer community even before the official announcement of the support for Kotlin in the Android Platform. In fact, Google stated in its blog that many developers have told them they love the Kotlin language. “Many of our own developers on the Android team have also been saying similar things... The Android community has spoken, and we listened” (CLERON, 2017).

In this regard, Google announced in its event, Google I/O in May of 2017, the offi-

[1] <https://github.com/evant/gradle-retrolambda>
[2] <https://kotlinlang.org>

cial support for the Kotlin language on the Android platform. Among other reasons for choosing Kotlin, Google mentioned it is "concise, expressive, and designed to be type and null-safe" and also "many Android developers have already found that Kotlin makes development faster and more fun" (GOOGLE, 2017). Another important reason is that Kotlin is a language fully interoperable with Java and runs on the JVM. From Kotlin source code, it is possible to generate Java bytecode for the JVM or JavaScript source code (KOTLIN, 2019b).

About a year after its 1.0 version release and two years after Google's official announcement of support for Kotlin, the language appeared in 2018 in the ranking of the most popular languages (22nd) and also in the raking of languages most loved by developers (2nd)(STACKOVERFLOW, 2018). Kotlin was rapidly adopted in the industry by companies like Pinterest, Gradle, Evernote, Uber, Corda, Coursera, and Pivotal that are already using Kotlin not only to build Android Apps but also for many other purposes, such as create internal desktop tools, and web applications (KOTLIN, 2019b).

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. This enforces the urgent need for research in this field to assist the industry and other researchers toward a better comprehension of the technology and to guide the development of new tools and new researches. Due to the scarce number of research studies in the area, this work aims to investigate the subject in depth.

In this research we aim to gather evidence to understand how developers are dealing with the recent adoption of Kotlin as an official language for Android development, their perceptions about the advantages and disadvantages using Kotlin, and the most common problems faced by them. To this end we defined the following research questions to guide this study:

- What are the most common problems faced by Kotlin developers on Android Platform?
- How are Android developers dealing with the Java-Kotlin interoperability?
- How are Android developers dealing with the functional paradigm in Kotlin?
- How are Android developers dealing with the development environment tools available for Kotlin?
- What is the perception of Android developers about Kotlin adoption?

Based on the above questions we believe that the main implications of this research are: (i) to be used as a guide for companies and developers who want to make a preliminary analysis before adopting Kotlin; (ii) assist decision making of how is the best way for adopting Kotlin in companies scenario; (iii) to facilitate improving techniques, tools, processes, and mechanisms to make the best use of the language on the Android platform.

Many researchers have proposed methods and criteria for evaluating programming languages (WIRTH, 1976; CHANDRA; CHANDRA, 2005; SCHMAGER; CAMERON; NOBLE, 2010), however, no consensus has emerged since most of them are prone to subjective assessment. In this research, we decided to mix different methods to enrich the results of our analyses through a concurrent triangulation strategy (EASTERBROOK et al., 2008). One method is based on two previous works (REBOUÇAS et al., 2016; BARUA; THOMAS; HASSAN, 2014), and consists on data analysis from StackOverflow questions, a popular collaborative Q&A website for developers. The other method uses basic qualitative strategies (MERRIAM; TISDELL, 2016) to confirm and cross-validate the results.

Our main findings reveal that developers are facing many problems, including code conversion from Java to Kotlin, as well as interoperability between both languages, especially regarding variables coming from Java, which does not have null-safety. Developers do seem to consider Kotlin easy to understand and to adopt it. This perception begins to change when the functional paradigm becomes more evident. According 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.

The remainder of this work is organized as follows:

- Chapter 2 shows an overview of Kotlin and its role in the Android development, aiming to support the understanding about our study.
- Chapter 3 describes the methodology used to conduct this study.
- Chapter 4 presents our finds and organize them in terms of each **RQ**.
- Chapter 5 we perform an overall assessment, a comparison with the literature, and an analysis of the validity and limitations of this work.
- Chapter 6 shows the conclusion of this work and future works proposals.

2 BACKGROUND

In this chapter, we introduce the Kotlin programming language and its role in Android development. We also present related works in Section 2.2

2.1 THE KOTLIN PROGRAMMING LANGUAGE

Kotlin was created by JetBrains in 2010 as an experiment looking at how some Java developers could benefit from a better programming experience for JVM (BRESLAV, 2018). Kotlin is a modern statically typed language that compiles to JVM bytecodes, JavaScript, and native bytecode. The language is backed by JetBrains, creator of IntelliJ IDEA, and distributed under the Apache 2 Open Source license. Its name comes from Kotlin Island in Saint Petersburg (PANCHAL, 2016).

Andrey Breslav, the creator of Kotlin, describes Kotlin as "a modern language for the industry because it is focused on flexible abstractions for code reuse and readability, static type safety for early error detection and explicit capturing of intent for maintainability and clarity" (REBELLABS, 2013).

Kotlin adoption rapidly increased after its v1.0 release in February of 2016. Figure 1 shows how the usage of Kotlin has increased over the years:

The same growth trend is shown in Figure 2, which shows the number of questions with the `kotlin` tag in the StackOverflow forum:

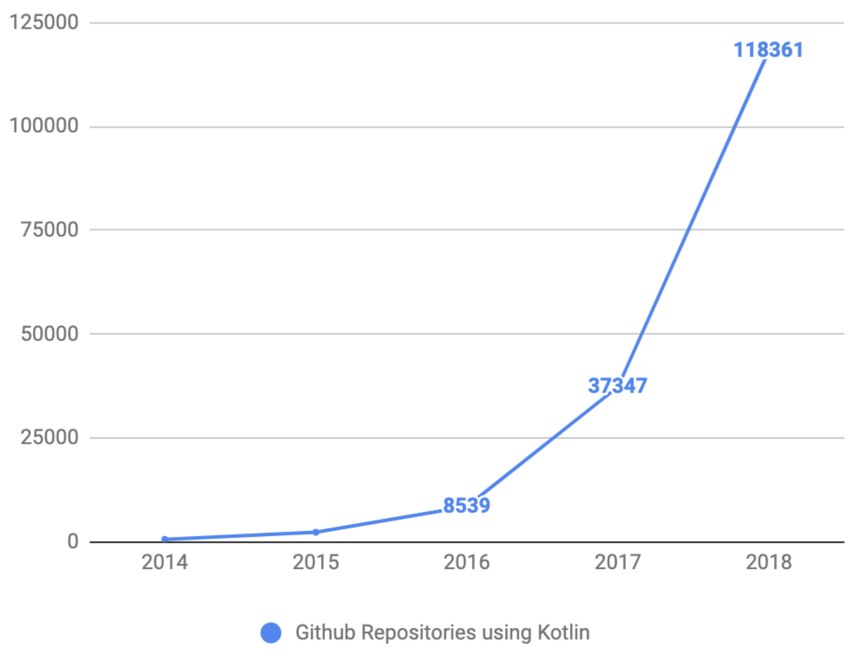
Both Figures (1, and 2) presents a significant increase after Kotlin v1.0 release in 2016, and also after Google's official announce for Kotlin support in 2017. The reason for such popularity success is interpreted by many specialists, including Breslav, because of the possibility to fully interoperate with Java. Hence, both languages can be freely mixed and the migration can be gradual and does not have to alter entire codebase. Developers can also take advantage from the Java environment using all its existing libraries and frameworks running with the same performance level as Java (KOTLIN, 2019c).

2.1.1 Kotlin Language Characteristics

Kotlin is a multi-paradigm language which supports Object-Oriented (OO) and Functional Programming Paradigm (FP), *i.e.*, it allows the developer to use OO and FP, or simply to combine them as it happens in most of the modern languages in industry (FLAUZINO et al., 2018). Furthermore, its support for non-nullable types makes applications less prone to null pointer exceptions. Kotlin also includes smart casting, higher-order functions, and extension functions which allows developers to focus on making code more readable and less verbose (BRESLAV, 2016; KOTLIN, 2019c).

JetBrains describe the main characteristics of Kotlin as the following (KOTLIN, 2019c):

Figure 1 – GitHub Repositories using Kotlin.

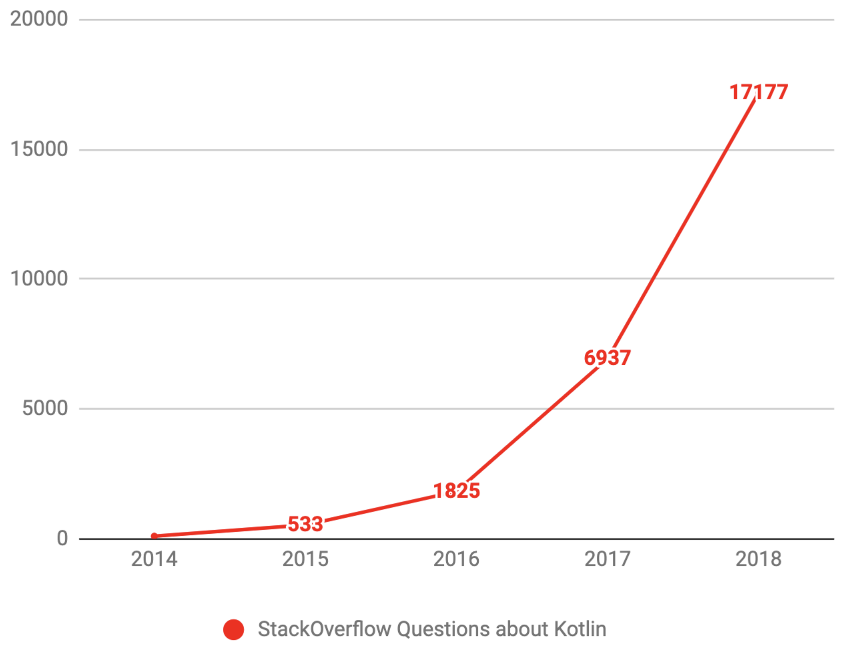


Fonte: The author (2019)

- Expressiveness: Kotlin has innovative language features, such as the support for type-safe builders, and delegated properties.
- Scalability: Kotlin has support for coroutines[1] which help building applications that scale to massive numbers of clients with modest hardware requirements.
- Interoperability: Kotlin is fully compatible with all Java-based frameworks, which lets developers stay on their familiar technology stack while reaping the benefits of a more modern language.
- Migration: Kotlin supports gradual, step by step migration of large codebases from Java to Kotlin. Developers can start writing new code in Kotlin while keeping older parts of your system in Java.
- Tooling: Kotlin is widely supported with plugins for different IDEs also with many framework-specific toolings.

[1] Computer program components that generalize subroutines for non-preemptive multitasking, by allowing execution to be suspended and resumed. <https://kotlinlang.org/docs/reference/coroutines-overview.html>

Figure 2 – StackOverflow Questions about Kotlin.



Fonte: The author (2019)

- Learning Curve: For a Java developer, getting started with Kotlin is very easy. The automated Java to Kotlin converter included in the Kotlin plugin helps with the first steps.

Kotlin documentation (KOTLIN, 2019a) also states that the language follows the principle of pragmatic evolution, *i.e.*, the language is built taking into consideration three main aspects:

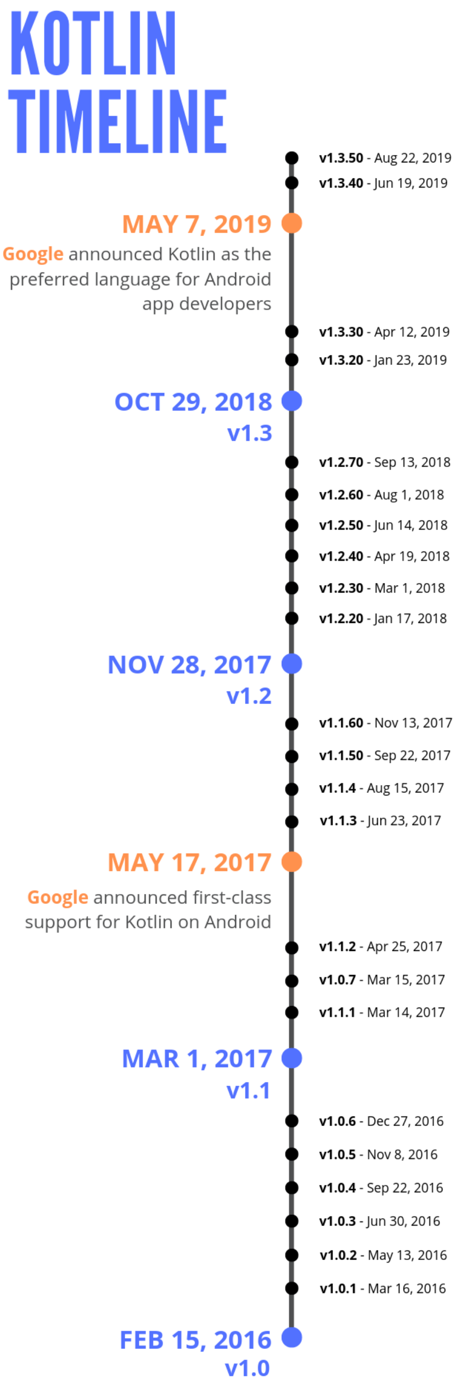
- **Keep the language modern over the years**: Evolve the language to keep it relevant to the needs of the users and up-to-date with their expectations, including not only adding new features but also removing old ones that are no longer recommended for production use and have altogether become legacy.

- **Stay in the constant feedback loop with the users**: Always announce incompatible changes well in advance, mark things as deprecated and provide automated migration tools before the change happens.

- **Make updating to new versions comfortable for the users**: Before take design decisions use every opportunity to make early versions of it with experimental status to gather feedback from early adopters.

Figure 3 presents the Kotlin releases timeline:

Figure 3 – Kotlin Release Timeline



Fonte: The author (2019)

It is possible to identify that the language is evolving with at least one major and many minor releases per year.

2.1.2 Key Features

In this section we present the key features of Kotlin, specially when compared with Java (KOTLIN, 2019a). All the following subsections are part of or refer to the official Kotlin documentation (KOTLIN, 2019c).

2.1.2.1 Data Classes

Data Classes are classes whose main purpose is to hold data. It automatically implements standard functionalities and utility functions that are often mechanically derivable from the data. To use this special type of classes, the class should be marked with `data`:

Code 1 – Data Class example.

```kotlin
data class User(val name: String, val age: Int)
```

The compiler automatically derives the following members from all properties declared in the primary constructor:

- `equals()/hashCode()` pair.
- `toString()` of the form `"User(name=John, age=42)"`.
- `componentN()` functions corresponding to the properties in their order of declaration.
- `copy()` function with a matching signature to copy the object. Providing explicit implementations is not allowed.

To ensure consistency and meaningful behavior of the generated code, data classes **(i)** needs to have at least one parameter; **(ii)** all primary constructor parameters need to be marked as `val` or `var`; **(iii)** cannot be abstract, open, sealed or inner;

2.1.2.2 Destructuring Declarations

Kotlin allows destructuring declarations, which means that you can destructure an object and creates multiple variables from it at once. For example:

Code 2 – Destructuring Declarations example.

```kotlin
val person = User("Laerte", 30)
val (name, age) = person
```

This code is compiled down using `componentN()`:

Code 3 – Destructuring compiled code.

```kotlin
val person = User("Laerte", 30)
val name = person.component1()
val age = person.component2()
```

A very good example of usage for Destructuring Declarations is when you need to iterate through a map:

Code 4 – Destructuring to iterate through a map.

```
for ((key, value) in map) {
    // do something with the key and the value
}
```

2.1.2.3 Null Safety

Kotlin was designed to eliminating the danger of null references from code, also known as the The Billion Dollar Mistake.[2]

In languages like Java, for example, if you access a member with a null reference it will result in a `NullPointerException`. Kotlin aims to eliminate, or at least reducing the opportunities for `NullPointerException` making programming more secure.

In Kotlin, regular variable types, such as `String`, `Int`, and `Boolean`, can not hold `null` references. To allow nulls, the variable should be declared as optional with the `?` operator, *e.g.* `String?`, `Int?`, and `Boolean?`. For example:

Code 5 – Safe and Unsafe declaration.

```
var myString: String = "abc"
var optionalString: String? = null
```

If you try to access an unsafe property directly the compiler will report the error `variable 'optionalString' can be null`, but there are some ways to access null properties safely:

- Checking for `null` in conditions, where you explicitly check if the variable is `null`, and handle the two options separately:

  Code 6 – Checking for null in conditions.

  ```
  val optionalString: String? = "Kotlin"
  if (optionalString != null) {
      print(optionalString.length)
  }
  ```

- Using the safe call operator (`?`):

  Code 7 – Safe call operator (?).

  ```
  val optionalString: String? = "Kotlin"
  println(optionalString?.length)
  ```

  Safe calls are also useful in chains or together with the scope functions:

  Code 8 – Chaining safe call operator.

  ```
  bob?.department?.head?.name
  ```

- Elvis operator (`?:`), which is an alias for the expression "if X is not null, use it, otherwise use some non-null value Z":

[2] <https://www.infoq.com/presentations/Null-References-The-Billion-Dollar-Mistake-Tony-Hoare/>

Code 9 – Elvis operator (?:).

```
    val optionalString: String? = "Kotlin"
    val length = optionalString?.length ?: -1
```

There is also another option with the not-null assertion operator (`!!`) which converts any value to a non-null type, but it throws an `NullPointerException` if the value is `null`. This is not the recommended way to unwrap an optional variable:

Code 10 – Not-null assertion operator (!!).

```
val optionalString: String? = "Kotlin"
val length = optionalString!!.length
```

2.1.2.4 Companion Objects

Kotlin does not have static members in classes or interfaces. Instead, it has companion objects which are more powerful than Java static members because they can extend classes and interfaces, and can be referenced and passed around like other objects. Members of the companion object can be called by using the class name as the qualifier:

Code 11 – Defining and calling a companion object.

```
class MyClass {
    companion object {
        val emptyString = ""
    }
}

val instance = MyClass.emptyString
```

Even though the members of companion objects look like static members in other languages, at runtime those are still instance members of real objects. Therefore, as such, they can, for example, implement interfaces:

Code 12 – Implementing interfaces within a companion object.

```
interface Factory<T> {
    fun create(): T
}

class MyClass {
    companion object : Factory<MyClass> {
        override fun create(): MyClass = MyClass()
    }
}

val f: Factory<MyClass> = MyClass
```

However, on the JVM you can have members of companion objects generated as real static methods and fields, if you use the `@JvmStatic` annotation.

2.1.2.5 Collections

Collections are a common concept for most programming languages. A collection usually contains a number of objects (this number may also be zero) of the same type. The Kotlin Standard Library provides implementations for basic collection types: sets, lists, and maps. The two top interfaces of collections are:

- A **read-only** interface that provides operations for accessing collection elements.
- A **mutable** interface that extends the corresponding read-only interface with write operations: adding, removing, and updating its elements.

The read-only collection types are covariant, *i.e.*, if a `Rectangle` class inherits from `Shape`, you can use a `List<Rectangle>` anywhere the `List<Shape>` is required, because the immutability of the list guarantees that they will always be rectangles in this case, so they are threaded like subtypes. For example:

Code 13 – Covariance with immutable list.

```kotlin
fun main() {
    val rectangles: List<Rectangle> = listOf()
    acceptShape(rectangles) // It compiles!
}

fun acceptShape(shapeList: List<Shape>) {
    // function body
}
```

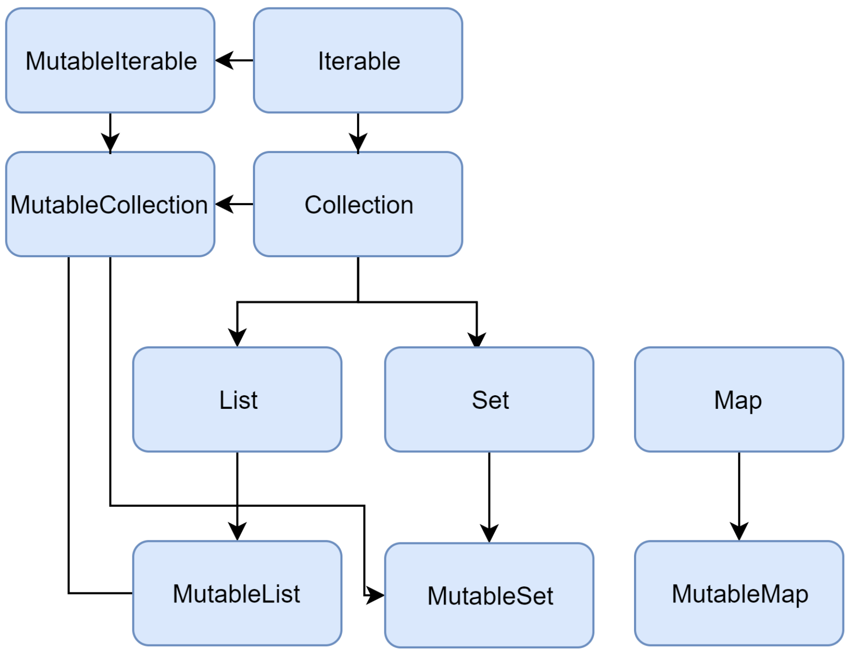
In turn, mutable collections are not covariant, otherwise, this would lead to runtime failures. If `MutableList<Rectangle>` was a subtype of `MutableList<Shape>`, you could insert other `Shape` inheritors (for example, `Circle`) into it, thus violating its `Rectangle` type argument. Figure 4 shows the diagram of the Kotlin collection interfaces.

As shown in Figure 4, `Collection<T>` is the root of the collection hierarchy. This interface represents the common behavior of a read-only collection: retrieving size, checking item membership, and inherits from the `Iterable<T>` interface that defines the operations for iterating elements. A `MutableCollection` add write operations, such as `add` and `remove`. Different from Java, Kotlin has separated interfaces for read-only and mutable collections, but they do not differ in its implementation, *i.e.* the implementation for the `kotlin.collections.MutableList` interface is usually an `java.util.ArrayList`.

The `List<T>`interface stores elements in a specified order and provides indexed access to them. `List` elements (including nulls) can duplicate: a list can contain any number of equal objects or occurrences of a single object. `MutableList` is a `List` with list-specific write operations to add or remove an element at a specific position.

In Kotlin, the default implementation of `List` is `ArrayList` which you can think of as a resizable array:

Figure 4 – Diagram of the Kotlin collection interfaces.



Fonte: Kotlin Official Documentation (KOTLIN, 2019c)

Code 14 – List example usage.

```
val numbers = mutableListOf(1, 2, 3, 4)
numbers.add(5)
numbers.removeAt(1)
numbers[0] = 0
numbers.shuffle()
println(numbers) // [0, 5, 3, 4]
```

`Set<T>` stores unique elements; their order is generally undefined. `null` elements are unique as well: a `Set` can contain only one `null`. `MutableSet` is a `Set` with write operations from `MutableCollection`.

The default implementation of a `Set` is `LinkedHashSet` which preserves the order of inserted elements. Hence, the functions that rely on the order:

Code 15 – Set example usage.

```
val numbers = setOf(1, 2, 3, 4)  // LinkedHashSet is the default implementation
val numbersBackwards = setOf(4, 3, 2, 1)

println(numbers.first() == numbersBackwards.first()) // false
println(numbers.first() == numbersBackwards.last()) // true
```

The alternative implementation is `HashSet` which does not preserve order and requires less memory to store the same number of elements.

`Map<K, V>` is not an inheritor of the `Collection` interface, however, it is a Kotlin collection type as well. A `Map` stores key-value pairs; keys are unique, but different keys can be paired with equal values. The `Map` interface provides specific functions, such as access to value by key, searching keys and values. `MutableMap` is a `Map` with map write operations.

The default implementation of `Map` is `LinkedHashMap` which preserves the order of elements insertion when iterating the map:

Code 16 – Map example usage.

```kotlin
val numbersMap = mutableMapOf("one" to 1, "two" to 2)
numbersMap.put("three", 3)
numbersMap["one"] = 11

println(numbersMap) // {one=11, two=2, three=3}
```

In turn, an alternative implementation is `HashMap` which does not preserves the elements order.

2.1.2.6 Higher-Order Functions and Lambdas

A higher-order function is a function that takes functions as parameters, or returns a function. Kotlin functions are first-class citizen, which means that they can be stored in variables and data structures, passed as arguments and returned from other higher-order functions.

The signature of the functions have a parenthesized parameter types list and a return type, *e.g.*, `(A, B) -> C` which denotes a type that represents functions taking two arguments of types `A` and `B` and returning a value of type `C`. For example:

Code 17 – Function declaration.

```kotlin
class Math {
    fun avg(a: Double, b: Double): Double {
        return  (a + b) / 2
    }
}

Math().avg(10.5, 2.5) // Invoking f(x)
```

Kotlin supports lambda expressions which are function literals, *i.e.* functions that are not declared. Kotlin also support first-class functions, which means that they can be stored in variables and data structures, passed as arguments to and returned from other higher-order functions:

Code 18 – Declaring first class function with lambda expression.

```kotlin
val sum = { x: Int, y: Int -> x + y }
sum.invoke(2, 5) // Invoking f.invoke(x)
```

Code 19 – Passing anonymous function as parameter.

```kotlin
val ints = listOf(-1,0,1,2)
ints.filter({ pred -> pred > 0 }) // this literal is of type '(pred: Int) -> Boolean
    '
```

A lambda expression is always surrounded by curly braces, parameter declarations in the full syntactic form go inside curly braces and have optional type annotations, and the body goes after an `->` sign. If the inferred return type of the lambda is not `Unit`, the last (or possibly single) expression inside the lambda body is treated as the return value.

If the compiler can figure out the signature, it is allowed not to declare the only parameter and omit `->`. The parameter will be implicitly declared under the name `it`:

Code 20 – Passing anonymous function with implicit parameter.

```kotlin
ints.filter({ it > 0 }) // this literal is of type '(it: Int) -> Boolean'
```

In Kotlin, there is a convention that if the last parameter of a function accepts a function, a lambda expression that is passed as the corresponding argument can be placed outside the parentheses:

Code 21 – Passing a lambda to the last parameter.

```kotlin
val product = items.fold(1) { acc, e -> acc * e } //Last parameter omitted

run { println("...") } //If the lambda is the only argument to that call, the
    parentheses can be omitted entirely
```

A lambda expression can also access its closure, *i.e.* the variables declared in the outer scope. Unlike Java, the variables captured in the closure can be modified:

Code 22 – Closures example.

```kotlin
fun bar() {
    val ints = listOf(1,2,3,4,5)
    var sum = 0 // local variable
    ints.filter { it > 0 }.forEach {
        sum += it // sum is outer scope
    }
    print(sum)
}
```

Using higher-order functions imposes certain runtime penalties. Each function is an object, and it captures a closure, *i.e.* those variables that are accessed in the body of the function. Memory allocations (both for function objects and classes) and virtual calls introduce runtime overhead.

Kotlin, provides a method to eliminate this overhead by inlining the lambda expressions by adding the `inline` keyword to the function declaration. When using inline functions, the compiler inlines the function body, *i.e.* it substitutes the body directly into places where the function gets called. This increases the resulting bytecode size.

The `forEach` function from Kotlin collections is marked as `inline`, which means that the following code:

Code 23 – Inline function before compiler translation.

```kotlin
val numbers = listOf(1, 2, 3, 4, 5)
numbers.forEach { println(it) }
```

Is translated to:

Code 24 – Inline function after compiler translation.

```
val numbers = listOf(1, 2, 3, 4, 5)
for (number in numbers)
    println(number)
```

2.1.2.7 Extension Functions

Kotlin, provides the ability to extend a class with new functionality without having to inherit from the class. This is done via special declarations called extensions.

To declare an extension function, we need to prefix the function name with a receiver type, *i.e.* the type being extended:

Code 25 – Defining and calling extension functions.

```
fun MutableList<Int>.swap(index1: Int, index2: Int) {
    val tmp = this[index1] // 'this' corresponds to the list
    this[index1] = this[index2]
    this[index2] = tmp
}

val list = mutableListOf(1, 2, 3)
list.swap(0, 2) // ‘this’ inside ‘swap()’ will hold the value of ‘list’
```

Extensions do not actually modify classes they extend. By defining an extension we do not insert new members into a class, but merely make new functions callable with the dot-notation on variables of this type:

2.1.2.8 Scope Functions

Scopes functions are functions which executes a block of code within the context of an object. Such functions are called on an object with a lambda expression provided, and it forms a temporary scope. In this scope, it is possible to access the object without its name. This type of functions does not introduce any new technical capabilities, because its only purpose is to make the code more concise and readable.

In short, these functions do the same: execute a block of code on an object. What is different is:

- How this object becomes available inside the block:

  Inside the lambda of a scope function, the context object is available by a short reference instead of its actual name. Each scope function uses one of two ways to access the context object: as a lambda receiver (`this`), which is recommended to operate on the object members, *e.g.*, call its functions or assign properties; or as a lambda argument (`it`), which is better when the object is mostly used as an argument in function calls.

- What is the result of the whole expression:

  Scope functions can **return the context of the object**, which allows chaining function calls on the same object after them; or **return the result of the lambda**, making possible to assign the result to a variable, chaining operations on the result, and so on.

There are five scope functions available on Kotlin:

- `let`: It can be used to invoke one or more functions on results of call chains. It's also used to perform non-null checking. **The context object** is available as an argument (`it`). **The return value** is the lambda result:

Code 26 – 'let' function example.

```kotlin
val str: String? = "Hello"
val length = str?.let {
    println("let() called on $it")
    processNonNullString(it) // OK: 'it' is not null inside '?.let { }'
    it.length
}
```

- `with`: It is used to call functions on the context object. It can be read as "with this object, do the following". It is a non-extension function. **The context object** is passed as an argument, but inside the lambda, it's available as a receiver (`this`). **The return value** is the lambda result:

Code 27 – 'with' function example.

```kotlin
val numbers = mutableListOf("one", "two", "three")
val firstAndLast = with(numbers) {
    "The first element is ${first()}," +
            " the last element is ${last()}"
}
println(firstAndLast) // The first element is one, the last element is
    three
```

- `run`: It does the same as `with` but invokes as `let` - as an extension function of the context object, and can also be used to execute a block of several statements where an expression is required. **The context object** is available as a receiver (`this`). **The return value** is the lambda result:

Code 28 – 'run' function example.

```kotlin
val numbers = mutableListOf("one", "two", "three")
val countEndsWithE = numbers.run {
    add("four")
    add("five")
    count { it.endsWith("e") }
}
println("There are $countEndsWithE elements that end with e.") // There
    are 3 elements that end with e.
```

- `apply`: It is used for code blocks that do not return a value and mainly operate on the members of the receiver object. The common case for apply is the object configuration. Such calls can be read as "apply the following assignments to the object". **The context object** is available as a receiver (`this`). **The return value** is the object itself:

Code 29 – 'apply' function example.

```
data class ApplicationConfig(
    var url: String = "UNKNOWN",
    var autoRefresh: Boolean = true,
    var autoLoad: Boolean = false)

val appConfig = ApplicationConfig()
appConfig.apply {
    this.url = "https://www.victorlaerte.com" // 'this' can be omitted
    this.autoRefresh = false
    this.autoLoad = true
}
```

- `also`: It is used for additional actions that do not alter the object, such as logging or printing debug information. Usually, it is possible to remove the calls of `also` from the call chain without breaking the program logic. **The context object** is available as an argument (`it`). **The return value** is the object itself:

Code 30 – 'also' function example.

```
val numbers = mutableListOf("one", "two", "three")
numbers
    .also { println("The list elements before adding new one: $it") }
    .add("four")
```

2.1.2.9 Type Checks and Cast

Kotlin provides the keyword `is` to perform type checking operations and whether an object conforms to a given type at runtime, Kotlin performs an implicit cast if it is an immutable value, for example:

Code 31 – Type checking example.

```
fun demo(x: Any) {
    if (x is String && x.length > 0) {
        print(x) // x is automatically cast to String
    }
}
```

It is also possible to use the operator `as` to cast objects. If the cast is not possible it throws an exception, *i.e.* its an *unsafe* cast. To avoid an exception being thrown, one can use a *safe* cast operator `as?` that returns `null` on failure:

Code 32 – Cast example.

```
val x: String? = y as? String
```

Kotlin ensures type safety of operations involving generics at compile time, while, at runtime, instances of generic types hold no information about their actual type arguments, *e.g.* `List<Foo>` is erased to just `List<*>`. In general, there is no way to check whether an instance belongs to a generic type with certain type arguments at runtime.

2.1.3 Kotlin in Android Development

In 2017, Google announced Kotlin as one of the officially supported languages for Android development. Until then, only Java and C++ were part of this list. Among the reasons for choosing Kotlin, Google mentioned that they believe Kotlin is a "great language that will make writing Android apps easier and more enjoyable." (CLERON, 2017)

Google also believes that Kotlin is a great match for the existing Android ecosystem due to its 100% compatibility with the Java programming language also making both languages fully interoperable, *i.e.*, both languages can be mixed into the existing codebase without any developer effort via some automatically applied translation conventions (*e.g.* property getters and setters are automatic created). It is also possible to customize how the translation is performed through Kotlin annotations. Another reason pointed was the demand by the Android community that has been said to "love the Kotlin language" (CLERON, 2017; ANDROID, 2019b), a statement reinforced by the StackOverflow 2018 developer survey that shown Kotlin as the second most loved programming language (STACKOVERFLOW, 2018). This phenomenon is also observed in the growth of questions related to Kotlin regarding Android development in the most popular question and answer site for software developers, the StackOverflow. Table 1 shows year over year growth.

Table 1 – Kotlin on Android YoY growth in StackOverflow.

Year	**Number of questions**[a]
2018	9,405
2017 (Google announces the official support for Kotlin in Android Development	3,071
2016 (Kotlin v1 release)	506

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

Fonte: The author (2019)

It is possible to identify in Table 1 that Kotlin was being used in Android applications even before Google announces its official support for Kotlin. It was possible due to the capability of full interoperation between Kotlin and Java. However, The Kotlin announcement for Android has started a new chapter in the partnership between JetBrains and Google, which in turn guaranteed to keep supporting Kotlin in Android development while improving the support in its official IDE, Android Studio. Hence, the IDE embedded

Kotlin since version 3.0, *i.e.* developers do not need to install any extra plugin or worry about compatibility issues (SHAFIROV, 2017).

JetBrains describe the main advantages that Kotlin brings for Android platform as the following (KOTLIN, 2019c):

- Compatibility: Kotlin is fully compatible with JDK 6, ensuring that Kotlin applications can run on older Android devices with no issues. The Kotlin tooling is fully supported in Android Studio, and compatible with the Android build system.
- Performance: A Kotlin application runs as fast as an equivalent Java one, due to the very similar bytecode structure. With Kotlin's support for inline functions, code using lambdas often runs even faster than the same code written in Java.
- Interoperability: Kotlin is 100% interoperable with Java, allowing to use all existing Android libraries in a Kotlin application.
- Footprint: Kotlin has a very compact runtime library, which can be further reduced through the use of ProGuard.[3] In a real application, the Kotlin runtime adds only a few hundred methods and less than 100K to the size of the .apk file.
- Compilation Time: Kotlin supports efficient incremental compilation, so while there is some additional overhead for clean builds, incremental builds are usually as fast as or faster than with Java.

Among all the benefits brought by Kotlin to Android, Java's limitations on the platform also contributed greatly to the rapid rise of Kotlin. Even though Android Nougat (API 24) has introduced new features in Java 8 using the Jack compiler (ANDROID, 2017b), this can only be put to use if developers work with a minimal version of the SDK 24 or higher, which makes it a problem for anyone who wants to support older versions of Android, and following trends today, updating Android versions by their users occurs at a very slow pace (JANGID, 2017).

Another important point mentioned by Google and JetBrain's when they announced the official support for Kotlin is regarding Android Studio, the official IDE for Android Development. It is a collaboration between JetBrains and Google, and it is built atop JetBrain's IntelliJ. In JetBrains's official post they pointed out that, "there will be close collaboration between the product teams to make sure that Kotlin is always working correctly in Android Studio" (SHAFIROV, 2017). Since then, Kotlin plugin was bundled with Android Studio and several features for Kotlin were introduced, such as Java–Kotlin code conversion, Kotlin project setup, and great support for code completion. Android Studio also provides a full-featured integration with Git and Gradle (CLERON, 2017).

[3] A free Java class file shrinker, optimizer, and obfuscator. <https://www.guardsquare.com/en/products/proguard>

2.1.4 Android KTX

In February of 2018, Google announced Android KTX, a set of Kotlin extensions that are included with Android Jetpack.[4] (ANDROID, 2019a)

KTX extensions provide concise, idiomatic Kotlin to Jetpack and Android platform APIs. To do so, these extensions leverage several Kotlin language features, such as extension functions, extension properties, lambdas, named parameters, and parameter default (ANDROID, 2019a). With the exception of the core module, `core-ktx`, all KTX module artifacts replace the underlying Java dependency in the `build.gradle` file, *e.g.* it is possible to replace a `androidx.fragment:fragment` dependency with `androidx.fragment:fragment-ktx` (ANDROID, 2019a).

The `core-ktx` include some default packages, such as `androidx.core.animation`, `androidx.core.content`, `androidx.core.graphics`, `androidx.core.graphics.drawable`, `androidx.core.net`, `androidx.core.os`, `androidx.core.text`, `androidx.core.transition`, `androidx.core.util`, `androidx.core.view`, `androidx.core.widget`.

The other availables modules are:

- **Fragment KTX**, provides a number of extensions to simplify the fragment API. For example, one can simplify fragment transactions with lambdas.
- **Palette KTX**, offers idiomatic Kotlin support for working with color palettes. For example, when working with a `Palette` instance, one can retrieve the selected swatch for a given `target` by using the get operator (`[ ]`).
- **SQLite KTX**, wraps SQL-related code in transactions, eliminating a lot of boilerplate code.
- **Collection KTX**, contains utility functions for working with Android's memory-efficient collection libraries, including `ArrayMap`, `LongParseArray`, `LruCache`, and others. Collections extensions take advantage of Kotlin's operator overloading to simplify things such as collection concatenation, *e.g.*, `arrayOf(1, 2) + arrayOf(3, 4)`.
- **ViewModel KTX**, provides a `viewModelScope()` function that makes it easier to launch coroutines[5] from a `ViewModel`.
- **Reactive Streams KTX**, provides functionalities to work with Reactive Programming through `LiveData` observable.
- **Navigation KTX**, provides extension functions to adapt the API of navigation components, such as Fragment, to become more succinct and Kotlin-idiomatic.
- **WorkManager KTX**, adds support for Kotlin coroutines by adding extension functions to `Operations` and `ListenableFutures` to suspend the current coroutine.

[4] <https://developer.android.com/jetpack>
[5] <https://developer.android.com/kotlin/coroutines>

2.2 RELATED WORK

The primary objective of this study is to understand how developers are dealing with the Kotlin adoption on Android Platform. Although, to the best of our knowledge, very few studies have been conducted with this purpose.

Jangid (JANGID, 2017) addressed significant parts where Java lacked as an Android programming language and presented Kotlin advantages and disadvantages over Java. As the main advantages, the author listed the Java interoperability, null safety guarantee, less verbosity, smart casts, destructuring declarations, and good support in the IDE environment. The drawbacks are the increase in compilation time, the small developer community, the increase in package size, and also the Android Studio code converter that still lacks on Java to Kotlin conversion.

Similarly, Banerjee et. al (BANERJEE et al., 2018) 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.* (FLAUZINO et al., 2018) conducted a large-scale empirical study involving more than 6 million lines of code from programs available in 100 repositories (50 Kotlin/50 Java), selected by order of popularity (stargazers count) on GitHub, and including Android and non-Android projects. They analyzed five well-known code smells: data class, large class, long method, long parameter list, and too many methods. 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, since Kotlin is more concise with a significant decrease in the number of lines of code, this may have been responsible for the result, where they identified that the smells are directly bound to this metric.

Schwermer (SCHWERMER, 2018) evaluated the performance of Kotlin and Java on Android Runtime using four benchmarks from the Computer Language Benchmarks Game suite.[6] The metrics used to evaluate the performance includes runtime, memory consumption, garbage collection, boxing of primitives as well as bytecode n-grams. The results show that Kotlin is slower than Java for all studied benchmarks. Another interesting result indicates the existence of an underlying garbage collection overhead when reclaiming Kotlin objects compared to Java. Furthermore, Kotlin produces larger and more varied bytecode than Java for a majority of the benchmarks.

Shah *et al.* (SHAH; SHAH; KANSARA, 2018) implemented prevention against App Repackaging using Proguard against a Kotlin Android app, aiming to assist Android app developers in securing their apps against reserve engineering attacks.

Moreover, there are several studies focused on investigating Q&A websites. For instance, Barua *et al.* (BARUA; THOMAS; HASSAN, 2014), analyzed the textual content of

[6] <https://benchmarksgame-team.pages.debian.net/benchmarksgame>

Stack Overflow discussions using the Latent Dirichlet Allocation (LDA) (BLEI; NG; JORDAN, 2003), a statistical topic modeling technique, to automatically discover the main topics present in developer discussions. The same approach was later used by Rebouças *et al.* (REBOUÇAS et al., 2016) to assess the adoption of the Swift Programming Language. The last study also combined a qualitative analysis to cross-validate the results. A similar strategy is used in this study to evaluate the Kotlin programming language.

3 METHODOLOGY

This chapter presents the research questions which guide our study (Section 3.1) followed by the mixed-method concurrent triangulation strategy used in this research (Section 3.2), the methods used to acquire, process and analyse data from StackOverflow (Section 3.3), and then conduct a basic qualitative study with Android developers (Section 3.4).

3.1 RESEARCH QUESTIONS

The goal of this study is to understand how developers are dealing with the recent adoption of Kotlin as the official language for Android development. By the term 'dealing with' we mean, the perception about the advantages and disadvantages of using Kotlin, and the most common problems faced by them. To guide our study, we established the following research questions:

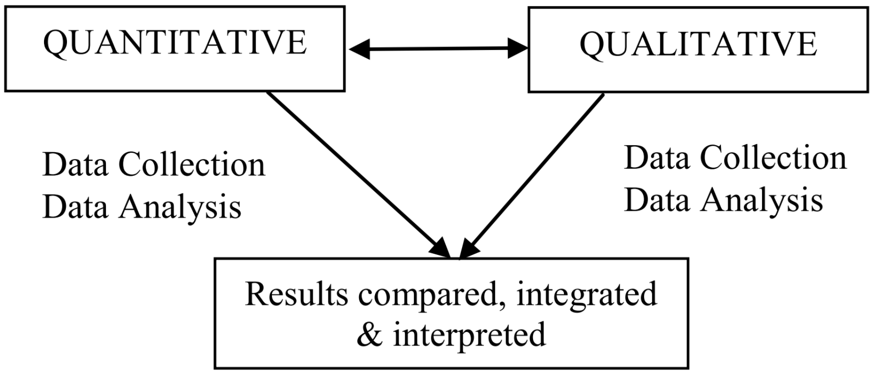
- **RQ1.** What are the most common problems faced by Kotlin developers on Android Platform?
- **RQ2.** How are Android developers dealing with the Java-Kotlin interoperability?
- **RQ3.** How are Android developers dealing with the functional paradigm in Kotlin?
- **RQ4.** How are Android developers dealing with the development environment tools available for Kotlin?
- **RQ5.** What is the perception of Android developers about Kotlin adoption?

3.2 TRIANGULATION STRATEGY

According to Creswell and Clark (CRESWELL; CLARK, 2011), mixed methods research is a research design to analyze data through the mixture of qualitative and quantitative methods in a single study or series of studies. 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.

Our methodology is based on the mixed-method concurrent triangulation strategy (EASTERBROOK et al., 2008). It is a mixed-method approach which uses different methods concurrently, and it aims to confirm, cross-validate or corroborate the results. As Easterbrook et al. (EASTERBROOK et al., 2008) states, the 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 validity. In addition, by collecting both types of data simultaneously, rather than sequentially, each analysis can be adapted to explore emerging results from the other. Thus, we decided to analyse developers' discussions about Kotlin for Android on StackOverflow, and concurrently we conducted a qualitative study by interviewing Android developers. Figure 5 shows a visual diagram of the concurrent triangulation strategy used in this study (ATIF; RICHARDS; BILGIN, 2013).

Figure 5 – Triangulation strategy diagram.



Fonte: Principles of Mixed Methods and Multimethod Research Design (MORSE, 2003)

Figure 5 shows quantitative and qualitative in capital letters. According to Morse (MORSE, 2003) notation system for mixed methods strategies, the capitalisation means that the priority is equal between the two approaches.

3.3 STACKOVERFLOW DATA ANALYSES

The first method of this study aims to analyze the repository of the StackOverflow, with the specific goal of uncovering the main topics discussed about Kotlin and Android. To this end, we reproduced the method from Barua et al. (BARUA; THOMAS; HASSAN, 2014) by analysing textual content using Latent Dirichlet Allocation (LDA), a statistical topic modeling technique, to automatically discover the main topics present in developer discussions (BLEI; NG; JORDAN, 2003). Next, we used LDAvis (SIEVERT; SHIRLEY, 2014), a web-based interactive visualization of the estimated topics.

3.3.1 Acquiring and Pre-processing Data

We used the StackExchange Database, publicly available in Internet Archive[1] to extract StackOverflow questions 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 'android' but also tags that refer to the Android platform like: 'fragments', 'recyclerview', 'intent', 'retrofit', 'room', 'sqlite', 'parcelable', 'glide', 'picasso', 'realm', 'gson', 'dagger', 'butterknife', 'eventbus', 'rxjava', 'volley', 'okhttp', and 'fresco'. One might argue that tagging is a manual process, which would incur in mistagged questions. However,

[1] <https://archive.org/details/stackexchange>

StackOverflow has an autocomplete feature for tagging to help diminishing the probability of using a misleading tag. A total of 9,405 questions were found. The publication date of the StackOverflow dump is 2019-03-04.

Before applying the topic modeling in our corpus, we performed five pre-processing steps (Appendix A):

1. Removal of the content inside the tags `<code>`, `<pre>` and `<blockquotes>`;
2. Removal of the HTML tags;
3. Removal of the URLs;
4. Removal of the stop words and one-letter-words;
5. Stemming of the remaining words using the Porter stemming algorithm (PORTER, 1997)

The Porter stemming is an algorithm for suffix stripping (PORTER, 1997). According to Porter, a document is represented by a vector of words or terms, and these terms with a common stem will usually have similar meanings, *e.g.* connect, connected, connecting, connection, connections. The suffix stripping process is then improved combining groups such as this into a single term. As a consequence, it will also reduce the total number of terms, and hence, diminish the size and complexity of the data in the system, which is advantageous.

For instance, after all these steps, the question *"<p>How can I make a phone call or dial a number in Android Kotlin? For example: Call <code>*21*2#</code></p>&#xA;"* is reduced to *"make phone call dial number android kotlin exampl call"*.

3.3.2 Topic Modeling and Visualization

Since manual inspection of 9,405 questions would require too much effort and too much time, we decided to follow the same approach of Barua *et al.* (BARUA; THOMAS; HASSAN, 2014), based on LDA, which is a flexible generative probabilistic model for collections of discrete data. It is based on a simple exchangeability assumption for the words and topics in a document (BLEI; NG; JORDAN, 2003).

In our analysis, each StackOverflow question (the composition of the title and body) is considered as a 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.

We used the Mallet 2.0.8 LDA implementation publicly available on Mallet's Website,[2] which supports different input parameters for training the model:

[2] <http://mallet.cs.umass.edu/>

- –num-topics 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 choose 15 after testing values ranging from 5 to 20 and have noticed which small values lead to topics too generic while big value leads to topics with small weight and with too much overlapping.

- –optimize-interval turns on hyperparameter optimization, *i.e.* 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 ranging from 10 to 100;

- –num-iterations is the number of sampling iterations and should be a trade off between the time taken to complete the sampling and the quality of the topic model. We chose 1,000;

- –num-top-words defines the top k words for each topic. We used 10 after analyzing that this number is enough to allow a qualitative analysis of the generated topics and also because greater numbers lead to terms with very low weight.

To support the qualitative analysis of the generated topics from LDA, we decided to resort to a topic viewing tool called LDAvis. Sievert and Shirley (SIEVERT; SHIRLEY, 2014) present LDAvis as a web-based interactive visualization of topics estimated using Latent Dirichlet Allocation. They also propose a novel measure, relevance, by which to rank terms within topics to help in topic interpretation.

They define the relevance of a term w to a topic k given a weight parameter λ (where $0 \leq \lambda \leq 1$) as:

$$r(w, k|\lambda) = \lambda log(\phi_{kw}) + (1 - \lambda) log\left(\frac{\phi_{kw}}{p_w}\right) \tag{3.1}$$

where ϕ_{kw} denotes the probability of a term $w \in 1, ..., V$ for topic $k \in 1, ..., K$, where V denotes the number of terms in the vocabulary, and let p_w denote the marginal probability of term w in the corpus. λ determines the weight given to the probability of term w under topic k relative to its lift (measuring both on the log scale). Setting $\lambda = 1$ results in the familiar ranking of terms in decreasing order of their topic-specific probability, and setting $\lambda = 0$ ranks terms solely by their lift. In the same study they presented the "optimal" value of λ for topic interpretation as 0.6.

3.4 BASIC QUALITATIVE STUDY

The goal of this study is to understand how developers are dealing with the recent adoption of Kotlin as the official language for Android development, their perception about the advantages and disadvantages of its usage, and the most common problems faced by them. To this end, we performed a basic qualitative study.

Merriam and Tisdell (MERRIAM; TISDELL, 2016) points as possible motivations to conduct a basic qualitative study is to understand:

1. How people interpret their experiences?
2. How they construct their worlds?
3. What meaning they attribute to their experiences?

Therefore, the overall purpose of basic qualitative research is to understand how people make sense of their lives and their experiences. In this sense, we believe that a basic qualitative study is an appropriate method to gather relevant findings for this research.

In a basic qualitative study, data can be collected through interviews, observations, or document analysis. In this study, we conducted semi-structured interviews (MERRIAM; TISDELL, 2016; SJØBERG et al., 2008) with Android developers.

3.4.1 Sample Selection

The sample selection strategy adopted was intentional, *i.e.* based on the assumption that the researchers choose the most appropriate sample to learn about the phenomenon investigated (RUNESON; HöST, 2009), and convenience-based, *i.e.* non-probability sampling where the sample is taken from a group of people easy to contact or to reach (SAUMURE; GIVEN, 2008).

We selected a group of individuals with the purpose of meeting specific prescribed criteria. The criteria used for the sample selection was:

1. Developers need to have experience with Android and Kotlin development: Given the fact that this research aims to understand the perception of developers in relation to Kotlin specifically on the Android platform, it does not make sense to interview developers who are not included in this context;
2. There will be no restrictions on the level of experience of the developer: We decided to diversify the level of experience of the participants, because the difficulties and easiness faced by a developer may be related to the level of experience in that technology. In this way, we can understand both what most inexperienced and experienced developers think;
3. Individuals need to speak English or Portuguese (the languages in which researchers are fluent): This was a necessary requirement so that there was no difficulty in conducting and understanding the interviews.

Table 2 presents some characteristics of the participants that fulfilled the requirements and compromised with the researchers from the beginning to the end of the whole process.

Table 2 – Characteristics of research participants.

Id	Age	Role	Years in SD[a]	Years in AD[b]	Years in KD[c]
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

[a]Software Development

[b]Android Development

[c]Kotlin Development

Fonte: The author (2019)

The participants of this research include people with many levels of experience and different roles such as developers, engineers, testers and team leaders. They are distributed in two different countries, all of them selected according to the sample selection criteria previously established assessed from a volunteer form sent in the most varied channels of communication for developers: email groups, Slack and Facebook communities, and also the official Kotlin forum.[3] Following the ethical criteria presented in Section 3.4.4, altogether seven professionals were invited by the researchers, and voluntarily participated in the research. During the interviews, the participants reported their experiences with Kotlin development on Android.

3.4.2 Data Collection

The data collection was performed using semi-structured interviews (RUNESON; HöST, 2009). As Merriam states (MERRIAM; TISDELL, 2016), the semi-structured interview is in the middle, between structured and unstructured. In this type of interview, either all of the questions are more flexibly worded or the interview is a mix of more and less structured questions. Usually, specific information is desired from all the respondents, in which case there is a more structured section to the interview. But some part of the interview is guided by a list of questions or issues to be explored, and neither the exact wording nor the order of the questions is determined ahead of time. This format allows the researcher to respond to the situation at hand, to the emerging worldview of the respondent, and to new ideas on the topic.

[3] <https://discuss.kotlinlang.org>

Following this strategy, we interviewed seven developers. In total, approximately 215 minutes of individual interviews were performed. Three interviews were conducted in person, while the remaining ones were conducted via video-conference. All interviewees are professional developers. Among them, three professionals are Android Specialists (one is Leader Engineer), three others consider themselves as Android developers but they also work with other platforms such as iOS or hybrid frameworks. One of them work specifically with Android automation tests. 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). Also, six of them have strong experience with Java. We refer to them as D1–D7.

The interviews were grounded in all of our **RQs**. We started asking about the interviewee's background in software development. Then, we moved to specific questions, including overview questions about Kotlin, Java-Kotlin Interoperability, Functional Programming, and Tools (*i.e.* Android Studio, gradle). The interview script is available on Appendix B. All interviews were recorded and transcribed *ipsis litteris*.[4] Due to the usage of two languages (English and Portuguese) in the interviews, the coding process was done using the original transcription and interview language, then we translated only the codes used in this report.

3.4.3 Data Analysis

It is a process of making sense out of data. It can be limited to determine how best to arrange the material into a narrative account of the findings (MERRIAM; TISDELL, 2016).

The qualitative analysis of the data was based on the open, and axial coding. Corbin and Strauss (CORBIN; STRAUSS, 2014) 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. As we are open to anything possible at this point, this form of coding is often called open coding (MERRIAM; TISDELL, 2016). The research steps were assisted by the qualitative data analysis software, MAXQDA 18.2.0.[5]

MAXQDA has several features that assist in the analysis and presentation of results in qualitative research. Among the main ones used in this research are:

- Transcription of audio files;
- Reading, editing, and coding of the data;
- Creation of links between a specific part from one document to another document;
- Exporting of demographic information;
- Full-text searching engine;

[4] word by word
[5] <https://www.maxqda.com>

- Exporting reports files to text, excel, html, xml;
- Statistical analysis of qualitative data (descriptive statistics);

The open coding of the transcripts was performed with the selection of text segments relevant to the research (codes). The 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. From this, the codes were grouped into categories (axial coding (CORBIN; STRAUSS, 2014)). The names of our categories were derived from a mix of sources:

1. The researchers;
2. The literature;
3. Kotlin and Android documentation.

3.4.4 Ethics

In order to follow research ethics regulations, all participants agreed with a consent form following the Research Ethics Board (REB) constraints, available on Appendix C. All the consents were previously sent to the interviewers by email and the agreement is registered in the interview record.

4 RESULTS

In this chapter, we first describe the general results from the StackOverflow data analysis. Following this, we organize the results in terms of each **RQ**. All the questions presented in this chapter as examples are detailed in Appendix D.

4.1 GENERAL RESULTS

Table 3 summarizes the topics generated by Mallet command line interface for 15 topics and 10 top terms through 1,000 iterations:

Table 3 – LDA Results.

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

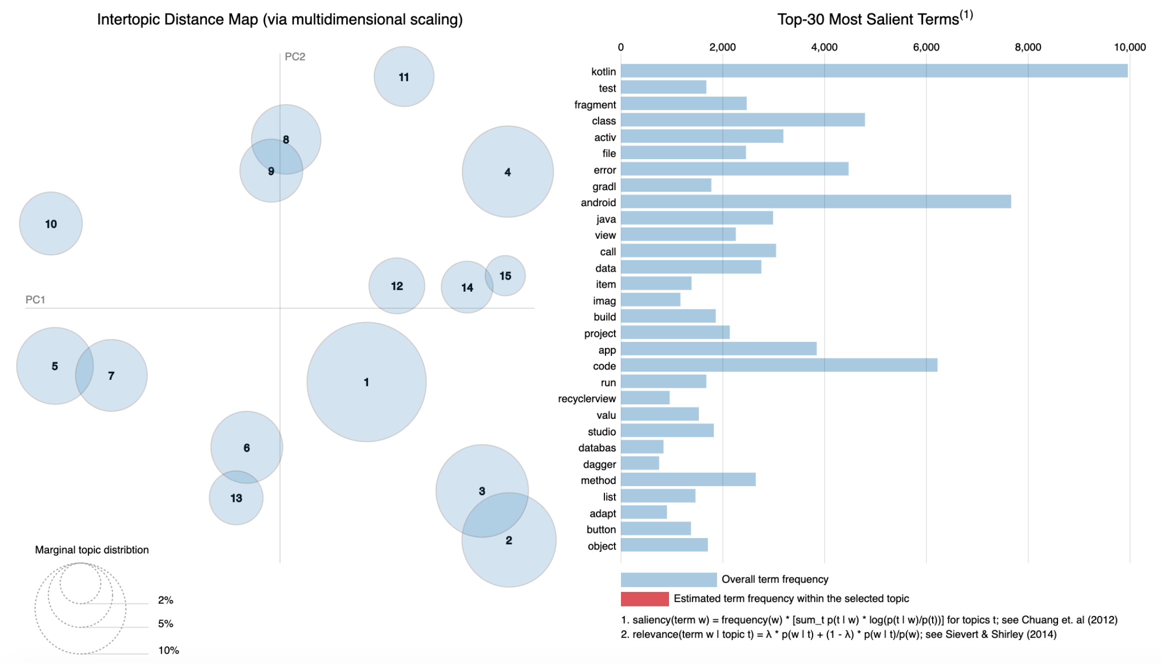
[a]The table is sorted by topic weight in descending order
[b]LDAvis column is used to identify topics in Figure 6

Fonte: The author (2019)

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. Its percentage does not sum up 9,405 and neither 100%, because one question can contain more than one topic. Hence, to compute this metric we did not take into consideration questions with weight lower than 10% for that topic. The topics are sorted by the topic weight in the third column. The LDAvis Id column is the identifier of the topic in the LDAvis tool (SIEVERT; SHIRLEY, 2014) that we used to support topic naming.

LDAvis is a topic visualization tool which supports qualitative analyses through a global visualization of the topics and how they differ from each other, while at the same time allowing for a deep inspection of the terms most highly associated with each individual topic (SIEVERT; SHIRLEY, 2014). Figure 6 shows the LDAvis output for our topic model with relevance adjusted to $\lambda = 0.6$.

Figure 6 – LDAvis: global topic view on the left, and the term bar charts on the right.



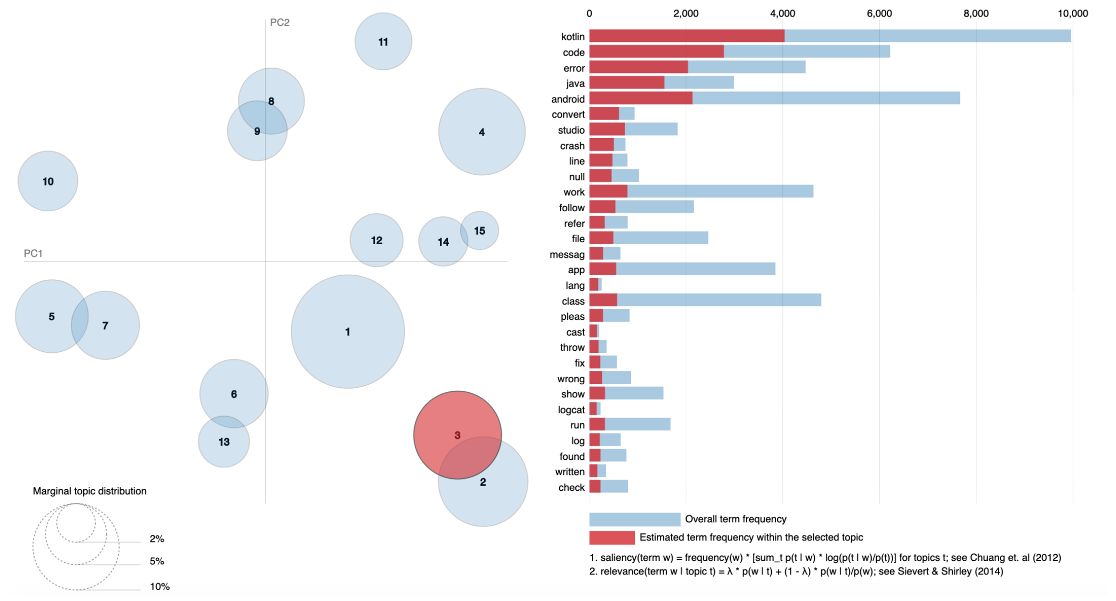
Fonte: The author (2019)

In Figure 6, the left panel shows the topics as circles, where the areas of the circles are proportional to the relative prevalence of the topic in the corpus. The number within is the identifier of the topic which corresponds to the LDAvis Id column from Table 3. The right panel shows the most relevant terms when some topic from the left panel is selected, which is helpful for interpreting a topic. Figure 7 shows the Java–Kotlin topic when selected.

In addition, selecting a term reveals the conditional distribution over topics for the selected term, as shown in Figure 8 when we select the term 'studio' for the topic Kotlin–Java. These interactions can be checked online.[1] As there is no selected topic in Figure 6, it only shows the overall term frequency for all topics.

[1] <https://www.victorlaerte.com/kotlin-stackoverflow-data-analysis/#topic=0&lambda=0.6&term=>

Figure 7 – LDAvis: selected topic in the right panel, and the estimated relevance and overall term frequency of each term in bar charts on the right.



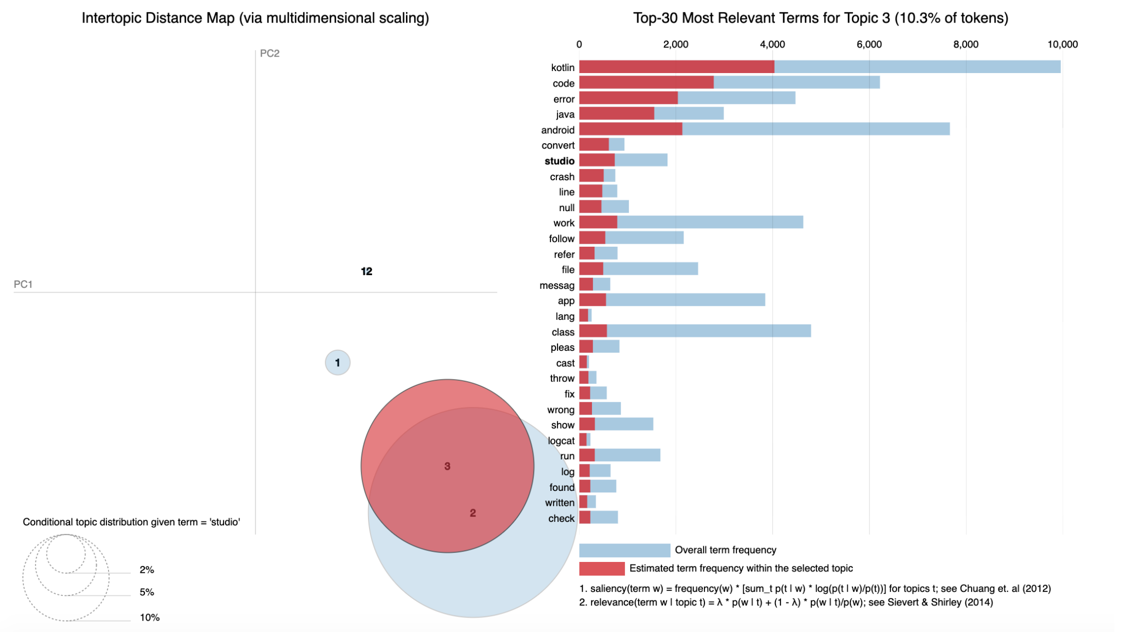
▭ Overall term frequency

🟥 Estimated term frequency within the selected topic

1. saliency(term w) = frequency(w) * [sum_t p(t | w) * log(p(t | w)/p(t))] for topics t; see Chuang et. al (2012)
2. relevance(term w | topic t) = λ * p(w | t) + (1 - λ) * p(w | t)/p(w); see Sievert & Shirley (2014)

Fonte: The author (2019)

Figure 8 – LDAvis: the distribution of the term 'studio' over the topics at the right panel.



▭ Overall term frequency

🟥 Estimated term frequency within the selected topic

1. saliency(term w) = frequency(w) * [sum_t p(t | w) * log(p(t | w)/p(t))] for topics t; see Chuang et. al (2012)
2. relevance(term w | topic t) = λ * p(w | t) + (1 - λ) * p(w | t)/p(w); see Sievert & Shirley (2014)

Fonte: The author (2019)

Regarding the basic qualitative study, we summarized our results in Table 4.

Table 4 – 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 Quality	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	StackOverflow	5	2.69
	Oficial Documentation	3	1.61
Similarities between Kotlin and Swift		11	5.91
		186	**100%**

Fonte: The author (2019)

The first column represents the main categories which groups related topics of the second column. Column three represents the number of codes, or segments, to that topic followed by its percentage over all codes. The last category ‘Similarities between Kotlin and Swift’ does not have any subdivision.

4.2 RESEARCH QUESTIONS

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

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

To answer this **RQ**, we rely on the StackOverflow data analysis where Table 3 summarizes the topics that developers are asking about Kotlin and Android. We named all the topics found with the support of LDAvis and describe all of them below.

- Top 5 topics - The topics with the highest weight are:

  1. ***General Questions (4,907)*** – this topic relies on general questions about the usage of Kotlin and/or Android platform, such as questions asking for a solution given a stack trace log or examples of usage, *e.g.*, *"How to download feature modules in an Android app?"* (Q3848), and *"What are the advantages of Kotlin programming language?"* (Q170).
  2. ***Java–Kotlin (3,402)*** – this topic contains questions that relate with Kotlin and Java regarding code conversion and also interoperation, *e.g. "Can we build Kotlin and Java Mix application?"* (Q5187), and *"Why do I receive "as non-null is null" error after Android Studio converts Java code into Kotlin code automatically?"* (Q5535).
  3. ***Classes, Objects, and Methods/Functions (2,813)*** – this topic addresses issues related to the structure of classes, objects, and methods of Kotlin. It contains questions that refer to the usage of properties, lambdas, constructors, or static attributes, *e.g. "In which situation val/var is necessary in Kotlin constructor parameter?"* (Q1624), and *"Is it possible to pass lambda to Intent?"* (Q9087).
  4. ***Build–Compilation (2,038)*** – this topic consists mainly of questions related to Gradle and Android Studio, but also questions about problems with versions, libraries, modules, and the Kotlin plugin that lead to compiling failures, *e.g. "Force Android Studio to use gradle 4.1"* (Q9381), and *"Android 3.1 build gradle 4.4 error occurred configuring project ':app' "* (Q4400).
  5. ***UI–Layout (1,712)*** – in the last topic of the top 5, we can find questions related to the construction of layouts in general, such as the use of visual Android components, custom layout creation, manipulating `layout.xml` files, or getting and handling view components in its controllers, *e.g. "How add TextView to View in kotlin"* (Q9383), and *"Android - How to change draw color of custom View?"* (Q2979).

- 6 - 10:

  6. ***UI–Navigation (1,528)*** – This topic also relies on UI questions, but it is more related to navigation mechanisms on Android, such as fragments transitions, navigation between activities, tabs interaction, and drawer layouts behaviors, *e.g. "Android, how to replace initial fragment?"* (Q6504).
  7. ***Data Types and Structures (1,440)*** – this topic include questions about basic Kotlin data types, such as strings, numbers, dates, arrays, and lists, *e.g. "How to append 2 strings in Kotlin?"* (Q6570).
  8. ***Google/Android Components (1,388)*** – This 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"* (Q1159).
  9. ***Background Tasks (1,373)*** – This topic refers to multithreading methods on Android and brings together various questions about coroutines, and also libraries like RxJava which is widely used to this purpose on Android, *e.g. "Android ViewState using RxJava or kotlin coroutines"* (Q3583).
  10. ***UI–Data View Layouts (1,199)*** – Again in UI components, this topic deals specifically with components for data visualization like RecyclerView, ListView, Spinner, *e.g. "How to show single item selected in recyclerview using kotlin"* (Q2808).

- 11 - 15: Among the topics with lowest weight remains:

  11. ***Connectivity (1,140)***, *e.g. "Retrofit parse result in Kotlin"* (Q3839).
  12. ***Multimedia Handling (908)***, *e.g. "How to save captured photos as jpg files on android camera2"* (Q5096).
  13. ***Data Storage (867)***, *e.g. "How to make primary key as autoincrement for Room Persistence lib"* (Q773).
  14. ***Dependency Injection (797)***, *e.g. "Dagger2 + Kotlin: lateinit property has not been initialized"* (Q5412).
  15. ***Testing (510)***, *e.g. "How can I run a single Android Test using Kotlin?"* (Q324).

Although developers are suffering from problems that are common to all Android developers, there are specific Kotlin issues regarding code conversion from Java to Kotlin, as well as interoperability between the two languages, how the Kotlin class members works, and also many compilation problems.

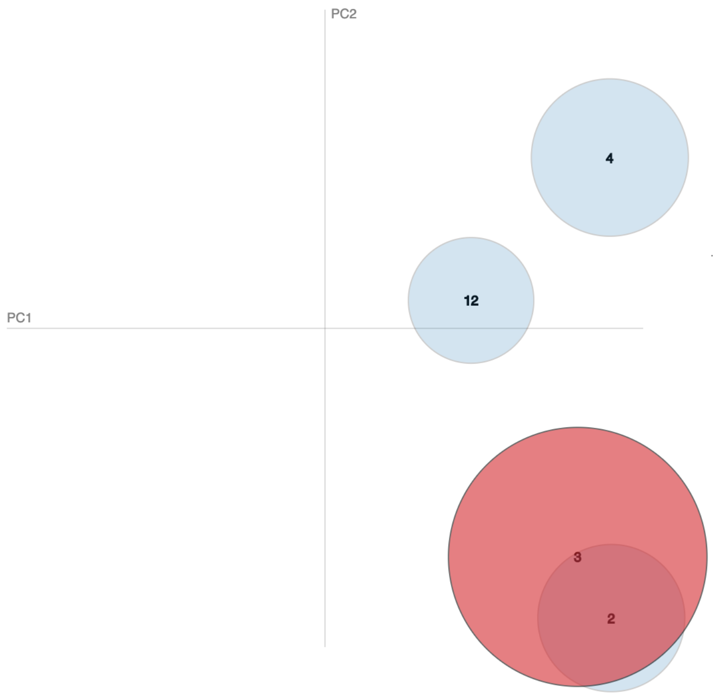
4.2.2 RQ2 - How are Android developers dealing with the Java-Kotlin interoperability?

We have studied the interoperability between Kotlin and Java mainly because this is one of the highlights of Kotlin creators for its adoption. Furthermore, another reason that led us to study this subject is that although the languages are interoperable, they have distinct characteristics such as the native null safety support in Kotlin that Java does not, so these difference in the languages can lead interesting results.

To answer this **RQ**, we first analyzed the LDA results and then went deeper into the subject by conducting interviews to acquire more evidence about it.

During the LDA analysis, the topic that most seemed relate with Kotlin-Java interoperation was the topic ***Java–Kotlin***, but due to LDAvis visualization it was possible to see that the *'java'* term is relevant to other three different topics: ***Classes, Objects, and Methods/Functions***, ***Build–Compilation***, and ***Data Storage***, as shown in Figure 9.

Figure 9 – Relevance of 'java' term between topics.



Fonte: The author (2019)

Then, we selected and manually investigated the 100 top questions of each topic to double check the classification. During the analysis of these questions, we removed the ***Data Storage*** topic due to the many number of false-positive questions, resulting in 300 questions in total. After examining the title, the question body, and the associated tags of these selected questions, we ended up with questions regarding problems with **Optionals**, **Kotlin Properties**, **Generics**, **Static Objects**, **Android API**, **Libraries**

Compatibility, and general problems that we did not address due to over-generality. We discuss these new topics below:

- Optionals: In this category, we grouped questions regarding problems with optionals and null pointer exceptions that the developers complain that crash their apps, *e.g. "Non-null assert is needed even after checking is not null in kotlin"* (Q3853), and *"How can I override a java method, and change the nullability of a parameter?"* (Q216).
- Kotlin Properties: This category groups questions about problems dealing with 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), and *"Kotlin: Setting a private Boolean in Java class via a Data class in Kotlin. Why am I not able to do this?"* (Q1486).
- Generics: This category groups questions about problems using Generics in Kotlin such as 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 are leading to questions such as: *"How to access static variable of java class in kotlin?"* (Q6396).
- Android API and Libraries Compatibility: These are questions about the usage of 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 that only accepts a Java class reference (Q4171).

Regarding the interviews, all interviews pointed the interoperation as an advantage to use Kotlin. Some of them call attention to the possibility of adopting Kotlin without the need for full codebase migration:

> *"The biggest benefit is that you can adopt Kotlin without having to change your whole project, as we have done in the company I work for. We had a project in Java that we decided to include Kotlin, and from now on, everything new is done in Kotlin. So, this increases the adoption curve and you have the ability to experiment by interacting with legacy code."* (D2)

> *"The main benefit is you can have a product that is already developed with Java and slowly migrate things to improve your product's technology stack and do things faster with Kotlin. So, you can keep the legacy application you have, but develop the new features in Kotlin. This is the main thing."* (D1)

> *"Since our system is a bit old and already developed in Java, people are migrating it to Kotlin. It is good that you can do it gradually, no need to go from scratch." (D3)*

> *"I think the main benefit, and I think it's a very huge one, is the interoperability, so you can keep an existing codebase in Java and introduce Kotlin, or use a Java library, or... Then you can have both, Kotlin and Java, in the same project. I think it's a very, very important benefit of Kotlin. I think it's the reason for the success of Kotlin because you don't throw away all the Java libraries, or you don't throw away all the old Java code, you can just call it from Java and vice-versa."* (D4)

Some of them also pointed out that, in order to keep full interoperability, there are several constraints that the language has to follow:

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

> *"(...) 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)

Among the main problems reported by the interviewees are about the optionals and the difficulty to deal with nullable objects:

> *"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)

> *"Java does not have this null safety control, so when you work with code coming from Java is dangerous. Even the Android SDK itself does not have much support for that."* (D5)

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)

> *"I think the biggest problems I've had in the project I worked on was this static method issue. Declaration of constants, static methods between the languages."* (D5)

> *"(...) we had to use the JVMStatic annotations and the construction annotations to be able to represent all the constructors that the Android SDK needs. It's complicated, or at least it's not very intuitive, and it's strange because it loses some of the elegance, the beauty of the language, using these annotations."* (D4)

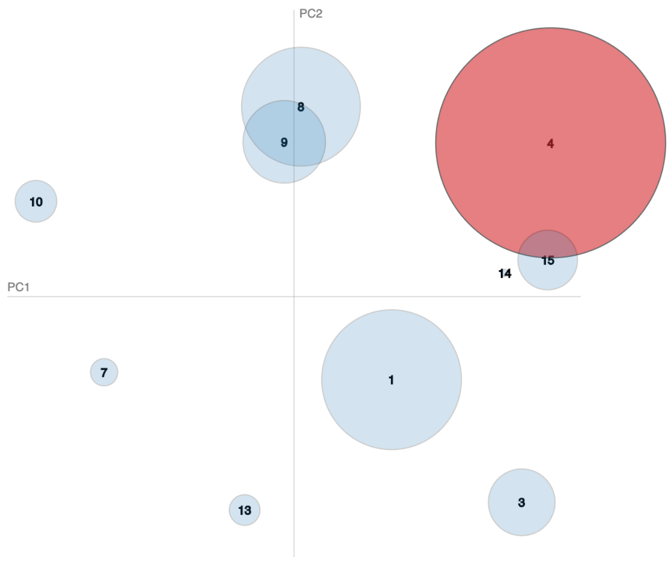
Despite of experiencing problems with optionals and nullability, as well as the use of static objects and overriding methods, among other less common problems, developers seems to consider the interoperability as a great benefit of adopting Kotlin.

4.2.3 RQ3 - How are Android developers dealing with the functional paradigm Kotlin?

We decided to study the functional programming in Kotlin because it was a point announced by JetBrains to bring a multi-paradigm modern approach to Android Platform (SHAFIROV, 2017). For many years, Android only had support to the Object-Oriented paradigm, so the adoption of functional programming can be very challenging even for experienced programmers, as shown by Chambers et al. (CHAMBERS et al., 2012). Hence the need to understand how developers are dealing with the new possibilities that the paradigm brings to the platform.

To answer this **RQ** we proceeded in a similar way to **RQ2**. We selected 100 questions from each topic that had the most relevant terms related to the functional paradigm according to LDAvis. Figure 10 shows the relevance for the term 'function' between the topics. The selected topics are: ***Classes, Objects, and Methods/Functions***, ***General Questions***, and ***Background Tasks***.

Figure 10 – Relevance of 'function' term between topics.



Fonte: The author (2019)

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 concepts such as **Higher-order Functions/Lambdas**, **Closures**, **Scope Functions**, **First Class Functions**, and general problems that we not addressed due to over-generality. We discuss these topics bellow:

- Higher-order Functions/Lambdas: This category contains questions regarding the usage of Higher-orderfunctions that are mainly achieved in Kotlin by the use of lambdas, *e.g. "How to pass a function as parameter in kotlin - Android"* (Q6181), and *"What is better approach of callback in Kotlin? Listener vs High-Order function"* (Q5381).
- Closures: This category contains questions regarding problems about scopes of enclosing functions, *e.g. "How to make 'this' a reference of Listener instead of the Activity in Kotlin?"* (Q3432), and *"Kotlin 'it' syntax in the context of Volley"* (Q1582).
- Scope Functions: This category relies on the special functions provided by Kotlin standard library. For instance, *"difference between kotlin also, apply, let, use, takeIf and takeUnless in Kotlin"* (Q2698).
- First Class Functions: These are mainly questions about the usage of function as first class citizens, *i.e.* assignee functions to variables or storing them in data structures, *e.g.* (Q1585).

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 Programming language, two of them did not respond or partially responded to the questions about functional programming because of the lack of experience in the subject. Some respondents related the functional programming as a factor that enhances the flexibility and modernity of language. We present some fragments from the interviews bellow:

> *"Yeah, I think Kotlin supports functional programming. Everything from the collections are very functional programming oriented. The lambdas, the 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)

> *"It is a language that is much more modern and already has the concepts of lambda, code blocks, closures ... What I can say is that it is very robust and modern."* (D6)

When asked about the problems faced with the functional paradigm, two interviewees mentioned difficulties to initially understand the Scope Functions, and one mentioned that had several problems using inline functions, but could not remember which ones exactly:

> *"(...) actually, I look for the old codes and I see that I used 'let' and 'also' in the wrong way, and also the 'apply' that I didn't know where to use it properly. Then today I look the code and see that I didn't use it right."* (D2)

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

Despite of the fact that the interviewees consider the functional paradigm as part of a modern approach and that it also helps in the readability of the code, they also state that the overuse of the paradigm can have the opposite effect causing the code to become more difficult to read:

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

> *"(...) it's a trade-off. Kotlin is so powerful that you can use 'let',' also', the Elvis operator to reduce and chain your code so much, thus that a simple code can become hard to understand."* (D6)

> *"Kotlin gives you practicality to create code blocks and in this block, if I don't explicitly define the name of the input variable, it will always be called 'it'. Then, if I nest functions, I can get lost in context with too many enclosed functions."* (D2)

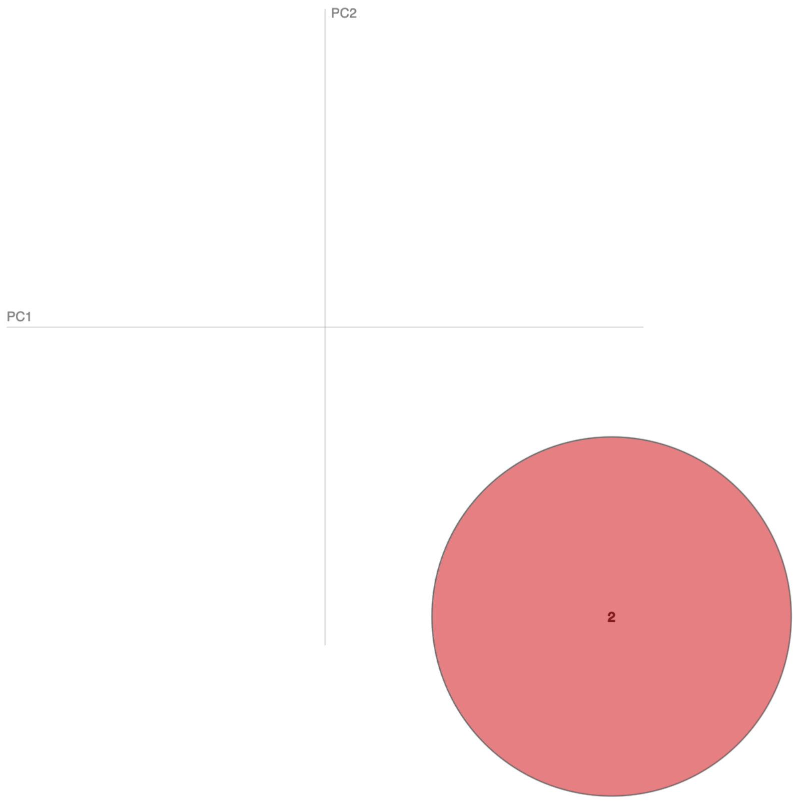
Despite of the fact that developers consider the functional paradigm as a factor that enhances the flexibility and modernity of language, they are facing problems regarding the usage of lambdas and closures. They also consider that its overuse can decrease code readability.

4.2.4 RQ4 - How are Android developers dealing wit the development environment tools available for Kotlin?

We decided to study the environment tools for Android because, as mentioned by Breslav (BRESLAV, 2018), creating a new programming language is more than just creating a new compiler. A whole ecosystem needs to be created, including tools that support other developers adopting it. Hence, we believe that by assessing the tools available for a programming language, we are also partially assessing how the adoption of such language is being perceived by the developers (ANSLOW; MARKSTRUM; MURPHY-HILL, 2011).

To answer this **RQ** we proceeded in a similar way to **RQ2** and **RQ3**. We selected 100 questions from the topic ***Build–Compilation*** which presents a big relevance for the terms 'gradle' and 'studio', and 100 more questions from the topic ***Java–Kotlin*** that shows a high relevance for the term 'studio' according to LDAvis and shown in Figures 11, and 12 respectively. We also take into consideration the relevance of other terms that refer to tools, such as 'plugin', 'librari', 'proguard' and 'kapt'.

Figure 11 – Relevance of 'gradle' term between topics.



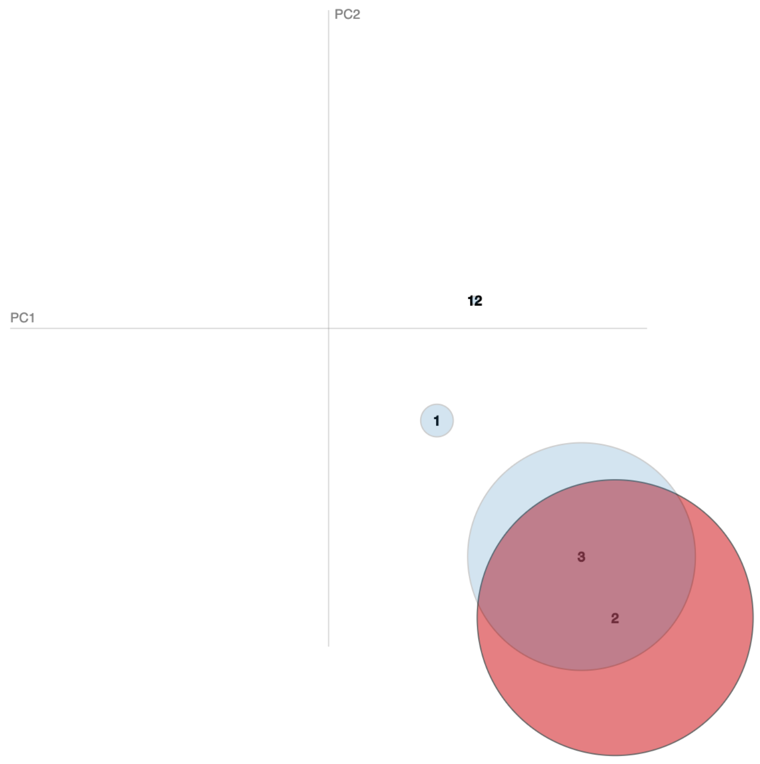
Fonte: The author (2019)

After the analysis of the 200 questions, we categorized the problems as the following:

- Android Studio code converter: This category groups questions regarding problems using the Android Studio code converter, a functionality that automatically translate Java code to Kotlin. Most of the questions report that compilation fails after code conversion, *e.g. "Kotlin Type mismatch after converted in Android Studio"* (Q3111), and *"Converting Java file to Kotlin now it won't compile - Internal compiler error"* (Q8770).

- Error After Upgrade: This category relies on compilation problems that appear after upgrading Android Studio, Kotlin, Kotlin Plugin or Gradle. *e.g. "Android unable to build project after updating kotlin runtime to 1.2.31"* (Q4689), and *"After update*

Figure 12 – Relevance of 'studio' term between topics.



Fonte: The author (2019)

to Android studio 3.1 i'm facing this erorr Could not find org.jetbrains.kotlin:kotlin-stdlib-jre8:1.2.0" (Q4341).

- Version Conflicts: This category relies on problems about version conflicts between libraries, the Kotlin plugin, the Kotlin language, and also Android tools. *"Warning "Kotlin plugin version is not the same as library version" (but it is!)"* (Q4784)[2].
- Kotlin Setup: This category contains questions about the difficulty faced by developers to setup Kotlin in their projects, e.g. *"I am unable to configure Kotlin in my android studio. getting error Error: Unable to find method 'BaseVariant-Data.getOutputs()Ljava/util/List;'."* (Q1002).
- kapt & databinding: The last category contains questions related to problems using Kotlin Annotation Processor (kapt) and Databinding library. Both terms presented high relevance in the LDAvis and manual analysis. (Q5941, and Q5294).

During the open coding process, we coded 36 segments from the interviews about Tools for Android/Kotlin development. The greatest number of problems reported by the interviewees relates to version upgrades of tools such as Gradle or Android Studio. The

[2] <https://youtrack.jetbrains.com/issue/KT-23744>

responses point out that even minor version upgrades can cause a lot of problems to the developers, leading their projects to stop compiling:

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

> *"Kotlin code was working... A simple code that was already working and then stopped working after we updated the gradle. It just stopped compiling."* (D3)

> *"When there's a new version of Android Studio or Gradle it's quite shocking and a lot of things stop working."* (D7)

Some respondents also report that fixing such issues is difficult because it is hard to identify where the problem really is, or what is causing the problem, whether it is a problem with Android Studio, Gradle, or Kotlin:

> *"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 the development with Kotlin, such as the automatic code converter from Java to Kotlin, and the project templates that already have Kotlin configured:

> *"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)

> *"The IDE already creates a full setup project for you, including a base architecture, and dependency plugins. So, I can't see any difficult to start code with Kotlin. [with Android Studio]"* (D6)

However, they point out that there is still a lot to improve on automatic code conversion, code suggestions, and fixing a few bugs:

> *"First I did the automatic code conversion, then I saw that Android Studio could not do all the conversions by itself, so I had to understand how is the Kotlin syntax to start covert it manually."* (D7)

> *"...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)

> *"I've seen problems that the IDE doesn't understand that the code is right, even it being right. For example, error messages that appear when you are implementing the parcelable interface and the IDE say that you have to create a CREATOR static object, but this is already correct the way it is."*[3] (D2)

Developers are facing many problems with tools for Kotlin development, specially regarding the Android Studio code converter, Gradle, and new versions of libraries. Interviewees report that even minor version upgrades can lead to various problems.

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

The goal of this **RQ** is to investigate the perception of the developers about Kotlin adoption for Android through the analysis of the interviews. In this section, we address the most relevant topics identified in the open coding process (Table 4) which were not previously analyzed.

In this **RQ**, we decided not to address the topics within the categories *Tools, Interoperability, and Classes and Objects* as they were already addressed by previous questions.

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

  > *"I think it's very concise. It's not as verbose as Java. I think it's 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 following it."* (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:

[3] <https://youtrack.jetbrains.com/issue/KT-19300>

> *"Kotlin is a language designed to be a much more functional language. You can define functions receiving other functions, this shows that it was thought in a way that you can program in a much simpler way than you do in Java [...] Since it is a language that already was born with functional paradigm it makes simpler to, for example, define high order functions, or extension functions, or... everything is much simpler."* (D7)
>
> *"You can even make a better, more modern architecture so you can modernize your code much easier and less costly."* (D3)
>
> *"It can improve if the person knows how to use with a new mindset. It's no use for you to program in Kotlin with Java style and not use the resources that Kotlin provides for you to develop in a secure way."* (D1)

- The category ***Performance, Productivity and Quality (27)*** was subdivided into the following topics: ***Readability/Legibility (13)***, ***Performance/Productivity (9)***, and ***QA (5)***. By analyzing these topics, it was possible to identify that developers consider the language to be more idiomatic, making it easier to read the code, and improving individual performance:

  > *"In my opinion, it is much easier to read a Kotlin code than a Java code for example."* (D3)
  >
  > *"We see a huge reduction of the codes to develop the same functionality [in Java], and this helped me a lot, I found it easy and I started to gain performance because of it."* (D7)

  However, they point out that such readability may begin to deteriorate with the excessive usage of the functional paradigm:

  > *"Kotlin brings many functions that improve readability, for example, aggregation functions that help you a lot... iterate over a list sometimes is not so clear in Java, you have too much code that decreases the code understanding, but it's necessary. I think Kotlin goes very 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 generate a code that is not so clear, for example, using too many 'let' or 'also' you can end up creating a code that is not easy to understand."* (D2)

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

> *"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 lambda, all those mini features that you can use in Android. I think it helps a lot in code quality, especially the null support."* (D4)

- The ***Documentation (8)*** with the topics ***StackOverflow (5)*** and ***Official documentation (3)*** show us that some interviewees believe that the official Documentation still lacks on 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 StackOverflow as a source of knowledge:

  > *"I think nowadays every developer uses StackOverflow. 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 consequently it will not be able to cover all use cases and needs. StackOverflow ends up being a tool, a very fast parallel platform for you to solve a problem. And the people from the community exemplify in a very simple and practical way."* (D6)

- The last category, ***Similarities between Kotlin and Swift (11)***, was only possible to identify due to the semi-structured interview method that we adopted. We did not subcategorize it because we only identified one subject regarding it. Four interviewees brought to light the similarities between Kotlin and Swift (language used in iOS, the main competitor of Android platform). Some of them report that, because they had experience with Swift and with the two languages being similar, this has made Kotlin adoption easier, reducing the learning curve. Others even point out as a reason for Kotlin adoption in his company, a startup, where according to him they need more multidisciplinary teams, and then 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)

  > *"There was a very cool situation that I experienced in the startup I worked for, which one of the reasons that led us to adopt Kotlin was the similarity with Swift. So what was the benefit we saw for our team? We were very small, only four mobile developers, two Android and two iOS. So if we decided to adopt a language that was a bit similar, we could somehow get closer to the platform of our teammates and then we were able to do code review... I was able to do code review in Swift, focused more on issues of code quality, clean code,*

not the platform itself, even because I have no deeper knowledge in Swift. This might be interesting to see if more multidisciplinary teams could also follow this approach." (D5)

5 DISCUSSION

This chapter discusses and analyses the results, putting them all together in an overall assessment, comparing with the existing literature, presenting its implications, and examining the threats to validity.

5.1 OVERALL ASSESSMENT

Developers consider Kotlin a modern language with many features that make it easy to adopt. Being able to use a new language without having to undergo an abrupt migration is an advantage pointed out by all the developers we interviewed. In fact, none of them cited a case where the code base was migrated 100% from Java to Kotlin. In most cases the code bases were partially migrated, or only the new things were developed in Kotlin, keeping the legacy in Java. Some of them have also stated that making use of the entire development environment from Java is a great facility. In this regard, we found many questions from developers about problems using null variables from Java, which has no nullability control, that was also reported by the interviewed developers.

Kotlin's multi-paradigm support is reported by respondents as a modern feature that brings more flexibility to development within the Android platform. Despite of this, they also consider that the overuse of the functional paradigm can make the code more complex and difficult to understand. This problem also appears in the analyzed questions, where it is possible to find questions regarding scope context within closures and also about the use of high order functions.

Additionally, many questions on StackOverflow pertain to problems in the toolset, specially the upgrade version of those, and gradle error messages that are either hard-to-understand or unhelpful.

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 features in the best way possible.

Regarding the documentation, developers believe that much can still be done to improve the support material of the language and point out StackOverflow as a platform of utmost importance to the developer community.

Finally, developers report that the similarity between Kotlin and Swift as one of the potentiating factors of Kotlin's adoption. One of the interviewees states that adopting it can greatly benefit multidisciplinary teams, especially in the startup's environment where, according to him, due to the scarcity of resources, much of the developers have to work on more than one platform such as Android and iOS.

5.2 LITERATURE ENFOLDING

In this section, the results of this research are compared the with the literature presented in Chapters 1 and 2 in order to identify similarities and differences.

According to JetBrains (BRESLAV, 2016; REBELLABS, 2013), and Google (CLERON, 2017), the reason for Kotlin's popularity is the flexibility, modernity, and the possibility to fully interoperate with Java. Our results converge to the same opinion, as the possibility of mixing both languages in a single code base as the main factor for its adoption, followed by the flexibility, and modernity, much for the possibility of using multiple programming paradigms, among them the functional paradigm as emphasized by Flauzino et al. (FLAUZINO et al., 2018). Despite of this, we had a complementary finding that, although interoperability is a great facilitator, it also leads developers to face many problems, being the second topic with more questions and reported by all the interviewees.

JetBrains (BRESLAV, 2016; REBELLABS, 2013), and Google (CLERON, 2017) also states that smart casting, higher-order functions, and extension functions allow developers to focus on making code more readable and less verbose. This statement is only partially supported by this research because despite of developers considering Kotlin as a language that helps readability, they believe that the overuse of functional capacities and operators, such as *elvis operator*, can cause the opposite effect, making the code difficult to read and understand. This might be explained by Chambers et al. (CHAMBERS et al., 2012) which 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, difficulty understanding the concept of higher-order functions (CHAKRAVARTY; KELLER, 2004), the type of functional expressions (JOOSTEN; BERG; HOEVEN, 1993), and the evaluation of iteration and recursive functions (EBRAHIMI, 1994; SEGAL, 1994).

Shafirov (SHAFIROV, 2017) affirms in the official Kotlin's Blog that developers do not need to install any extra plugin or worry about compatibility since JetBrains and Google improved the Kotlin support in the Android Studio IDE after its official announcement as an Android supported language. However, developers are still facing many problems regarding version compatibility, specially for Android Studio and Gradle, but also for other tools.

Our findings are also in agreement with Jangid (JANGID, 2017) regarding Kotlin's advantages (Java interoperability, null safety guarantee, less verbosity, smart casts, destructuring declarations, and good support in the IDE), and disadvantages (increase in compilation time, the small developer community, the increase in package size, and the not efficient Android Studio code converter).

In a large-scale empirical study with GitHub repositories involving more than 6 million lines of code, Flauzino et al. (FLAUZINO et al., 2018) identified that source code written in Kotlin usually have fewer code smells than code that are written in Java. The same find is presented by Banerjee et al. (BANERJEE et al., 2018). In this regard, 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.* (REBOUÇAS et al., 2016) research was about Swift, the similarity between both programming languages (Kotlin and Swift) came to light throughout the work. By comparing their findings with ours, it is possible to identify which in both studies the interviewees find languages easy to adopt. Another important similarity is that in both languages dealing with optional variables seems to cause problems to developers. Also, developers related problems with to the toolset, this can be explained by the immaturity of tools due to the short time since languages were released. Unlike their work, we did not find major problems with Kotlin compiler, only one of the interviewees raised the point that the compilation could be faster.

5.3 IMPLICATIONS

As a general implication of this work, we believe this is a valuable guide for companies and developers who want to make a preliminary analysis before adopting Kotlin. It is possible to assess if its adoption is worth for their context, and also assist decision making of how is the best way to adopt it. In addition, for those who have already adopted the Kotlin language, it is equally relevant for improving techniques, tools, processes, and mechanisms to make the best use of the language on the Android platform.

The implications of this work are detailed per research questions:

- **RQ1**: Companies and developers can know in advance what kind of problems they will face if they decide to adopt Kotlin. They may take preliminary steps to mitigate these problems, or even decide to not adopt the language if they find its advantages do not outweigh the potential problems of its adoption.

- **RQ2**: Companies and developers can understand the advantages and disadvantages of interoperating Java and Kotlin code, so it is easier for them to choose between a partial Kotlin adoption or a full code-base migration.

- **RQ3**: It is very important for developers a better understanding of the functional paradigm because until then the support for it was very limited in Android. As shown by Chambers *et al.* (CHAMBERS et al., 2012) even experienced developers face a number of conceptual problems learning functional programming. Researchers can also use these results for studying if the usage of Kotlin in functional programming disciplines can help the understanding of the functional paradigm.

- **RQ4**: Toolmakers can take advantage of the finds to create new tools and improve existing ones, e.g. developers reported several issues with Android Studio's automatic code converter making clear the need for a more efficient conversion tool.
- **RQ5**: Companies and developers can understand how Kotlin adoption has been perceived by the developers, taking into account productivity, readability, and code quality. They can create processes and mechanisms to maximize gains and mitigate losses, e.g. create static code analyzers to be integrated into the development process to prevent readability degradation due to the overuse of functional paradigm.

5.4 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 analyzed 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 analyzed in this study (SYSTEMS, 2012). We also used LDAvis analyses[1] to support and corroborate our manual analyses.

Thirdly, 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.

Finally, one can argue the fact that the study was performed with only seven developers as a limitation, but it does not invalidate the results of our research, because with the use of mixed methods and especially of the simultaneous triangulation strategy it was possible to adapt our analysis to explore the emergent results more effectively, consequently increasing the validity of the study. Nevertheless, we recognize that the lack of women in our sample is rather a limitation of our work.

[1] <http://www.victorlaerte.com/kotlin-stackoverflow-data-analysis/#topic=0&lambda=0.6&term=>

6 CONCLUSIONS AND FUTURE WORK

In this research, we conducted two studies to understand how developers are dealing with the recent adoption of Kotlin as an official language for the Android Platform. To this end, we first performed an in-depth analysis of StackOverflow questions related to Kotlin and Android, and interviewed seven Kotlin developers to complement our analysis.

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 the code quality, as well as readability and productivity. Among the main factors are: (i) Java interoperability which brings the flexibility of the Kotlin with the advantage to use well established libraries from Java; (ii) the null safety guarantee that decrease the chances of having to face null pointer exceptions; (iii) the less verbosity which increases the readability and understanding, (iv) the lambdas and higher-order functions (v) 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. Among them: (i) the use of null variables and optionals within Kotlin; (ii) the problems with the toolset, specially with Android Studio and Gradle; (iii) the degradation of readability with the overuse of the functional paradigm, and (iv) the interoperation with Java, which despite bringing advantages, it still requires a lot of work from the developers, *e.g.* the need to use annotations on static object objects, and method overriding.

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 that this study is the first step towards a better understanding of Kotlin within the Android platform. As future work, we plan to analyze Android open source projects to investigate in depth the Kotlin adoption and usage in the wild deeper, especially projects that use both languages, Java and Kotlin. We also encourage other researchers to conduct new studies on how Kotlin and Swift can be related in the mobile development context, comparing its optional variables usages. Other researchers could also investigate existing codebases to look at how Java to Kotlin migration is happening, how Kotlin is being introduced in existing systems, how this affects the structure of these systems, how does this impact bugs and code reviews, and whether reengineering of pre-existing Java code is happening. Since Kotlin is open source, we encourage researchers to contribute with new proposals for the language that could mitigate the problems presented in this research. In addition, new tools can be created to help developers regarding the toolset, but we strongly recommend researchers to focus on code conversion tools.

REFERENCES

ANDROID. *Android Developers - Use Java 8 language features*. 2017. Accessed: 2019-02-15. Disponível em: <https://developer.android.com/studio/write/java8-support>.

ANDROID. *Compiling with Jack*. 2017. Accessed: 2019-02-15. Disponível em: <https://source.android.com/setup/build/jack>.

ANDROID. *Android Developer - Android KTX*. 2019. Accessed: 2019-02-15. Disponível em: <https://developer.android.com/kotlin/ktx>.

ANDROID. *Android Developers - Develop Android apps with Kotlin*. 2019. Accessed: 2019-02-15. Disponível em: <https://developer.android.com/kotlin/index.html>.

ANSLOW, C.; MARKSTRUM, S.; MURPHY-HILL, E. Evaluation and usability of programming languages and tools: (plateau). In: *Proceedings of the 10th SIGPLAN Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software*. New York, NY, USA: ACM, 2011. (Onward! 2011), p. 119–120. ISBN 978-1-4503-0941-7. Disponível em: <http://doi.acm.org/10.1145/2048237.2048258>.

ATIF, A.; RICHARDS, D.; BILGIN, A. A student retention model: Empirical, theoretical and pragmatic considerations. In: . [S.l.: s.n.], 2013.

BANERJEE, M.; BOSE, S.; KUNDU, A.; MUKHERJEE, M. A comparative study: Java vs kotlin programming in android application development. *International Journal of Advanced Research in Computer Science*, v. 9, n. 3, p. 41–45, 2018. ISSN 0976-5697. Disponível em: <https://www.ijarcs.info/index.php/Ijarcs/article/view/5978>.

BARUA, A.; THOMAS, S. W.; HASSAN, A. E. What are developers talking about? an analysis of topics and trends in stack overflow. *Empirical Software Engineering*, v. 19, n. 3, p. 619–654, Jun 2014. Disponível em: <https://doi.org/10.1007/s10664-012-9231-y>.

BLEI, D. M.; NG, A. Y.; JORDAN, M. I. Latent dirichlet allocation. *J. Mach. Learn. Res.*, JMLR.org, v. 3, p. 993–1022, mar. 2003. ISSN 1532-4435. Disponível em: <http://dl.acm.org/citation.cfm?id=944919.944937>.

BRESLAV, A. *Kotlin Blog - Kotlin 1.0 Released: Pragmatic Language for JVM and Android*. 2016. Accessed: 2019-02-15. Disponível em: <https://blog.jetbrains.com/kotlin/2016/02/kotlin-1-0-released-pragmatic-language-for-jvm-and-android/>.

BRESLAV, A. *History of Kotlin*. 2018. Accessed: 2019-02-15. Disponível em: <https://www.coursera.org/lecture/kotlin-for-java-developers/history-of-kotlin-K8pZr>.

CHAKRAVARTY, M. M. T.; KELLER, G. The risks and benefits of teaching purely functional programming in first year. *J. Funct. Program.*, Cambridge University Press, New York, NY, USA, v. 14, n. 1, p. 113–123, jan. 2004. ISSN 0956-7968. Disponível em: <http://dx.doi.org/10.1017/S0956796803004805>.

CHAMBERS, C.; CHEN, S.; LE, D.; SCAFFIDI, C. The function, and dysfunction, of information sources in learning functional programming. *J. Comput. Sci. Coll.*, Consortium for Computing Sciences in Colleges, USA, v. 28, n. 1, p. 220–226, out. 2012. ISSN 1937-4771. Disponível em: <http://dl.acm.org/citation.cfm?id=2379703.2379745>.

CHANDRA, S. S.; CHANDRA, K. A comparison of java and c#. *J. Comput. Sci. Coll.*, Consortium for Computing Sciences in Colleges, USA, v. 20, n. 3, p. 238–254, fev. 2005. ISSN 1937-4771. Disponível em: <http://dl.acm.org/citation.cfm?id=1040196.1040228>.

CLARKE, A. C. Book. *Profiles of the future; an inquiry into the limits of the possible, by Arthur C. Clarke.* Rev. ed. [S.l.]: Harper Row New York, 1973. xvii, 237 p. p. ISBN 0060107928.

CLERON, M. *Android Developers Blog - Android Announces Support for Kotlin.* 2017. Accessed: 2019-02-15. Disponível em: <https://android-developers.googleblog.com/2017/05/android-announces-support-for-kotlin.html>.

CORBIN, J.; STRAUSS, A. *Basics of Qualitative Research: Techniques and Procedures for Developing Grounded Theory.* SAGE Publications, 2014. ISBN 9781483315683. Disponível em: <https://dx.doi.org/10.4135/9781452230153>.

CRESWELL, J.; CLARK, V. *Designing and Conducting Mixed Methods Research.* SAGE Publications, 2011. ISBN 9781412975179. Disponível em: <https://books.google.com.br/books?id=YcdlPWPJRBcC>.

EASTERBROOK, S.; SINGER, J.; STOREY, M.-A.; DAMIAN, D. Selecting empirical methods for software engineering research. In: _____. *Guide to Advanced Empirical Software Engineering.* London: Springer London, 2008. p. 285–311. ISBN 978-1-84800-044-5. Disponível em: <https://doi.org/10.1007/978-1-84800-044-5_11>.

EBRAHIMI, A. Novice programmer errors: language constructs and plan composition. *International Journal of Human-Computer Studies*, v. 41, n. 4, p. 457 – 480, 1994. ISSN 1071-5819. Disponível em: <http://www.sciencedirect.com/science/article/pii/S107158198471069X>.

FLAUZINO, M.; VERÍSSIMO, J.; TERRA, R.; CIRILO, E.; DURELLI, V. H. S.; DURELLI, R. S. Are you still smelling it?: A comparative study between java and kotlin language. In: *Proceedings of the VII Brazilian Symposium on Software Components, Architectures, and Reuse.* New York, NY, USA: ACM, 2018. (SBCARS '18), p. 23–32. ISBN 978-1-4503-6554-3. Disponível em: <http://doi.acm.org/10.1145/3267183.3267186>.

FORBES. *The World's Most Valuable Brands.* 2018. Accessed: 2019-02-15. Disponível em: <https://www.forbes.com/powerful-brands/list/>.

GOOGLE. *Google I/O Developer Keynote.* 2017. Accessed: 2019-02-15. Disponível em: <https://events.google.com/io2017/schedule/?section=may-17&sid=___keynote2___>.

JANGID, M. Kotlin – the unrivalled android programming language lineage. *Imperial Journal of Interdisciplinary Research*, v. 3, n. 8, 2017. ISSN 2454-1362. Disponível em: <http://www.imperialjournals.com/index.php/IJIR/article/view/5491>.

JOOSTEN, S.; BERG, K. V. D.; HOEVEN, G. V. D. Teaching functional programming to first-year students. *Journal of Functional Programming*, Cambridge University Press, v. 3, n. 1, p. 49–65, 1993.

KOTLIN. *Kotlin Docs - Comparison to Java Programming Language.* 2019. Accessed: 2019-02-15. Disponível em: <https://kotlinlang.org/docs/reference/comparison-to-java.html>.

KOTLIN. *Kotlin Language Official Website*. 2019. Accessed: 2019-02-15. Disponível em: <https://kotlinlang.org/>.

KOTLIN. *Kotlin Official Documentation*. 2019. Accessed: 2019-02-15. Disponível em: <https://kotlinlang.org/docs/reference/>.

MERRIAM, S. B.; TISDELL, E. J. *Qualitative research: A guide to design and implementation*. [S.l.]: John Wiley and Sons, 2016. ISBN 111900361X.

MORSE, J. Principles of mixed methods and multimethod research design. In: ______. [S.l.: s.n.], 2003.

NEWZOO. *Global Mobile Market Report - Free Version*. 2018. Accessed: 2019-02-15. Disponível em: <https://resources.newzoo.com/hubfs/Reports/Newzoo_2018_Global_Mobile_Market_Report_Free.pdf>.

PANCHAL, A. K. P. R. K. A comparative study: Java vs kotlin programming in android. *International Journal of Advanced Research in Computer Science*, v. 2, n. 9, 2016. ISSN 2456-4338. Disponível em: <https://www.ijiter.com/wp-content/uploads/papers/2016/A-comparative-study-Java-Vs-kotlin-Programming-in-Android.pdf>.

PORTER, M. F. Readings in information retrieval. In: JONES, K. S.; WILLETT, P. (Ed.). San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1997. cap. An Algorithm for Suffix Stripping, p. 313–316. ISBN 1-55860-454-5. Disponível em: <http://dl.acm.org/citation.cfm?id=275537.275705>.

REBELLABS. *JVM Languages Report extended interview with Kotlin creator Andrey Breslav*. 2013. Accessed: 2019-02-15. Disponível em: <https://jrebel.com/rebellabs/jvm-languages-report-extended-interview-with-kotlin-creator-andrey-breslav/>.

REBOUÇAS, M.; PINTO, G.; EBERT, F.; TORRES, W.; SEREBRENIK, A.; CASTOR, F. An empirical study on the usage of the swift programming language. In: *2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*. [S.l.: s.n.], 2016. v. 1, p. 634–638.

RUNESON, P.; HöST, M. Guidelines for conducting and reporting case study research in software engineering. *Empirical Softw. Engg.*, Kluwer Academic Publishers, Hingham, MA, USA, v. 14, n. 2, p. 131–164, abr. 2009. ISSN 1382-3256. Disponível em: <http://dx.doi.org/10.1007/s10664-008-9102-8>.

SAUMURE, K.; GIVEN, L. M. *The SAGE Encyclopedia of Qualitative Research Methods*. Thousand Oaks: SAGE Publications, Inc., 2008. Disponível em: <http://sk.sagepub.com/reference/research>.

SCHMAGER, F.; CAMERON, N.; NOBLE, J. Gohotdraw: Evaluating the go programming language with design patterns. In: *Evaluation and Usability of Programming Languages and Tools*. New York, NY, USA: ACM, 2010. (PLATEAU '10), p. 10:1–10:6. ISBN 978-1-4503-0547-1. Disponível em: <http://doi.acm.org/10.1145/1937117.1937127>.

SCHWERMER, P. *Performance Evaluation of Kotlin and Java on Android Runtime*. Dissertação (Mestrado) — KTH, School of Electrical Engineering and Computer Science (EECS), 2018.

SEGAL, J. Empirical studies of functional programming learners evaluating recursive functions. *Instructional Science*, v. 22, n. 5, p. 385–411, Sep 1994. ISSN 1573-1952. Disponível em: <https://doi.org/10.1007/BF00891962>.

SHAFIROV, M. *Kotlin Blog - Kotlin on Android. Now official.* 2017. Accessed: 2019-02-15. Disponível em: <https://blog.jetbrains.com/kotlin/2017/05/kotlin-on-android-now-official/>.

SHAH, Y.; SHAH, J.; KANSARA, K. Code obfuscating a kotlin-based app with proguard. In: . [S.l.: s.n.], 2018. p. 1–5.

SIEVERT, C.; SHIRLEY, K. LDAvis: A method for visualizing and interpreting topics. In: *Proceedings of the Workshop on Interactive Language Learning, Visualization, and Interfaces*. Baltimore, Maryland, USA: Association for Computational Linguistics, 2014. p. 63–70. Disponível em: <https://www.aclweb.org/anthology/W14-3110>.

SJØBERG, D. I. K.; DYBÅ, T.; ANDA, B. C. D.; HANNAY, J. E. Building theories in software engineering. In: _____. *Guide to Advanced Empirical Software Engineering.* London: Springer London, 2008. p. 312–336. ISBN 978-1-84800-044-5. Disponível em: <https://doi.org/10.1007/978-1-84800-044-5_12>.

STACKOVERFLOW. *Developer Survey Results.* 2018. Accessed: 2019-02-15. Disponível em: <https://insights.stackoverflow.com/survey/2018/>.

STATISTA. *Statista dossier about Smartphones.* 2019. Accessed: 2019-07-30. Disponível em: <https://www.statista.com/study/10490/smartphones-statista-dossier/>.

SYSTEMS, C. R. *Sample Size Calculator.* 2012. Accessed: 2019-02-15. Disponível em: <https://www.surveysystem.com/sscalc.htm>.

WIRTH, N. Programming languages: What to demand and how to assess them. *Software Engineering*, 01 1976.

APPENDIX A – PRE-PROCESSING STACKOVERFLOW DATA

To remove the content inside the tags <code>, <pre> and <blockquotes> we performed simple replacement using regex with Visual Studio Code:[1]

- `<code.*?/code>`
- `<pre.*?/pre>`
- `<blockquote.*?/blockquote>`

Then, we ran the routines below to perform the remaining pre-processing steps:

Code 33 – Pre-processing StackOverflow Data.

```kotlin
import org.jsoup.Jsoup
import java.io.File
import java.nio.charset.StandardCharsets
import java.nio.file.Files
import java.nio.file.Paths

fun main() {
    removeHtml()
    removeURLs()
    removePunctuation()
    removeStopWordsAndOneLetterWords()
    stem()
    nameAndLabel()
}

fun getReadFile(fileName: String): File = File(fileName)
fun getOrCreateWriteFile(fileName: String): File {
    val file = File(fileName)
    if (!file.exists()) {
        file.createNewFile()
    }
    return file
}

fun removeHtml() {
    val readFile = getReadFile("kotlin-posts-1.csv")

    if (readFile.exists()) {
        val lines = Files.lines(Paths.get(readFile.toURI()))
        val newFile = getOrCreateWriteFile("kotlin-posts-2.csv")

        lines.forEach {
            val text = Jsoup.parse(it).text()

            val newText = text.replace("\n", "").replace("\r", "")
            newFile.appendText("$newText\n")
```

[1] <https://code.visualstudio.com/>

```
        }
    }
}

fun removeURLs() {

    val readFile = getReadFile("kotlin-posts-2.csv")

    if (readFile.exists()) {
        val lines = Files.lines(Paths.get(readFile.toURI()))
        val newFile = getOrCreateWriteFile("kotlin-posts-3.csv")
        val regex = Regex(
            "((https?|ftp|gopher|telnet|file|Unsure|http):((//)|(\\\\))+[\\w\\d:#@
                %/;$()~_?+-=\\\\.&]*)")

        lines.forEach {
            val text = it.replace(regex, "")
            newFile.appendText("$text\n")
        }
    }
}

fun removePunctuation() {
    val readFile = getReadFile("kotlin-posts-3.csv")

    if (readFile.exists()) {
        val lines = Files.lines(Paths.get(readFile.toURI()))
        val newFile = getOrCreateWriteFile("kotlin-posts-4.csv")
        val regex = Regex("[\\p{P}&&[^\u0027]]")
        lines.forEach {
            val text = it.replace(regex, " ")
            newFile.appendText("$text\n")
        }
    }
}

fun removeStopWordsAndOneLetterWords() {
    val readFile = getReadFile("kotlin-posts-4.csv")

    if (readFile.exists()) {
        val lines = Files.lines(Paths.get(readFile.toURI()))
        val newFile = getOrCreateWriteFile("kotlin-posts-5.csv")

        val stopWords = listOf("a", "about", "above", "after", "again", "against", "
            all", "am", "an", "and", "any", "are", "aren’t", "aren‘t", "arent", "as"
            , "at", "be", "because", "been", "before", "being", "below", "between",
            "both", "but", "by", "can’t", "can‘t", "cant", "cannot", "could", "
            couldn’t", "couldn‘t", "couldnt", "did", "didn’t", "didn‘t", "didnt", "
            do", "does", "doesn’t", "doesn‘t", "doesnt", "doing", "don’t", "don‘t",
            "dont", "down", "during", "each", "few", "for", "from", "further", "had"
            , "hadn’t", "hadn‘t", "hadnt", "has", "hasn’t", "hasn‘t", "hasnt", "have
            ", "haven’t", "haven‘t", "havent","having", "he", "he’d", "he‘d", "he’ll
            ", "he‘ll", "he’s", "he‘s", "hes", "her", "here", "here’s", "here‘s", "
            hers", "herself", "him", "himself", "his", "how", "how’s", "how‘s", "
            hows","i", "i’d", "i‘d", "id", "i’ll", "i‘ll", "ill", "i’m", "i‘m", "im"
            , "i’ve", "i‘ve", "ive", "if", "in", "into", "is", "isn’t", "isn‘t", "
            isnt", "it", "it’s", "it‘s", "its", "itself", "let’s", "let‘s", "lets",
            "me", "more", "most", "mustn’t", "mustn‘t", "my", "myself", "no", "nor",
             "not", "of", "off", "on", "once", "only", "or", "other", "ought", "our"
            , "ours ourselves", "out", "over", "own", "same", "shan’t", "shan‘t", "
            she", "she’d", "she‘d", "she’ll", "she‘ll", "she’s", "she‘s", "shes", "
            should", "shouldn’t", "shouldn‘t", "shouldnt", "so", "some", "such", "
            than", "that", "that’s", "that‘s", "thats", "the", "their", "theirs", "
            them", "themselves", "then", "there", "there’s", "there‘s", "theres", "
            these", "they", "they’d", "they‘d", "they’ll", "they‘ll", "they’re", "
            they‘re", "they’ve", "they‘ve", "this", "those", "through", "to", "too",
             "under", "until", "up", "us", "very", "was", "wasn’t", "wasn‘t", "wasnt
            ", "we", "we’d", "we‘d", "we’ll", "we‘ll", "we’re", "we‘re", "we’ve", "
            we‘ve", "were", "weren’t", "weren‘t", "what", "what’s", "what‘s", "whats
            ", "when", "when’s", "when‘s", "where", "where’s", "where‘s", "wheres",
            "which", "while", "who", "who’s", "who‘s", "whos", "whom", "why", "why’s
            ", "why‘s", "with", "won’t", "won‘t", "wont", "would", "wouldn’t", "
            wouldn‘t", "wouldnt", "you", "you’d", "you‘d", "you’ll", "you‘ll", "you’
            re", "you‘re", "you’ve", "you‘ve", "your", "yours", "yourself", "
            yourselves")

        lines.forEach { line ->
            val text = line.toLowerCase()
            val words = text.split(" ").toMutableList()
            val newLine = StringBuffer()

            words.forEach { word ->
                if (word.length > 1 && !stopWords.contains(word)) {
                    newLine.append(word)
                    newLine.append(" ")
                }
            }

            newFile.appendText("$newLine\n")
        }
    }
}

fun stem() {
    val readFile = getReadFile("kotlin-posts-5.csv")

    if (readFile.exists()) {

        val output = getOrCreateWriteFile("kotlin-posts-6.csv")
        Stemmer.stem(readFile.absolutePath, output.absolutePath)
    }
}

fun nameAndLabel() {
    val readFile = getReadFile("kotlin-posts-6.csv")

    if (readFile.exists()) {
        val lines = Files.lines(Paths.get(readFile.toURI()), StandardCharsets.
            ISO_8859_1)
        val newFile = getOrCreateWriteFile("kotlin-posts-final.txt")

        var i = 0
        lines.forEach {
            val myString = it.prependIndent("q$i Question$i ").toByteArray(Charsets.
                ISO_8859_1)
            val text = String(myString, Charsets.UTF_8)
            newFile.appendText("$text\n")
            i++
        }
    }
}
```

APPENDIX B – INTERVIEW SCRIPT

The interview script was developed based on the Research Questions presented in section 3.1. As we used semi-structured interviews (RUNESON; HöST, 2009) the following interview script was used to gather specific desired information but the interview was not restricted by these questions.

Context of the Participant

- Can you please tell us your name?
- How old are you?
- How long have you been working as a developer?
- How long have you been working with Android development?
- What is your current role in the organization you work for?
- How long have you been working with Kotlin?
- How do you consider your knowledge level in Kotlin?
- Do you have any background with Java ?

Kotlin

- How was your first contact with Kotlin?
- What are the reasons that made you adopt Kotlin in Android Development?
- Do you think Kotlin improves code quality in any way?
- What do you most like in Kotlin development?
- What do you most dislike in Kotlin development?
- Can you remember one or more problems you faced in Kotlin development and had to resort to any specialized forum like StackOverflow?

Java-Kotlin Interoperability

- Kotlin is designed with Java Interoperability in mind. Existing Java code can be called from Kotlin in a natural way, and Kotlin code can be used from Java as well. In your opinion, how do you face the Kotlin and Java interoperability?

- Despite the fact that Kotlin and Java are interoperable, both languages are quite distinct. Have you ever experienced problems trying to interoperate Kotlin and Java?
- What you think could be improved in Kotlin-Java Interoperability?

Functional Programming

- Do you think that Kotlin provides support for functional programming in any way?
- Have you ever experienced problems with any functional programming practices in Kotlin?
- What do you think could be improved in Kotlin functional paradigm support?

Tools

- In your opinion does Android Studio provide a good experience for Kotlin Developers?
- Have you experienced any problems trying to develop Android Apps with Kotlin regarding Android Studio?
- Gradle is currently the most popular compilation automation system used by Android developers. To use Kotlin with Gradle some settings must be made, but may vary depending on the version of gradle and Android Studio you are using. Have you experienced problems using gradle?
- What do you think could be improved in Android Studio regarding Kotlin?

Closing

- Is there anything else you would like to discuss that was not addressed by the previous questions or you believe that can improve this interview?
- Do you have any question for me?

APPENDIX C – CONSENT FORM FOR RESEARCH PARTICIPATION

FEDERAL UNIVERSITY OF PERNAMBUCO
CONSENT FORM FOR RESEARCH PARTICIPATION

Study Title: An Empirical Study on the Usage of the Kotlin Programming Language for Android Development

Principal Investigator: Victor L. de Oliveira

Professor Advisor: Leopoldo Teixeira

I am a master student at the Federal University of Pernambuco, in the Center of Informatics (CIn). I am planning to conduct a research study with android developers, which I invite you to take part in. This form has important information about the reason for doing this study, what we will ask you to do if you decide to be in this study, and the way we would like to use information about you if you choose to be in the study.

Why are you doing this study?

You are being asked to participate in a research study about the usage of Kotlin Programming Language for Android Development. The purpose of the study is to understand how developers are dealing with the recent adoption of Kotlin as the official language for Android development: The perception about the advantages and disadvantages to using Kotlin for Android development; Most common problems faced by developers that choose Kotlin as Android programming language.

What will I do if I choose to be in this study?

You will be asked to describe your experience as an Android Developer and to describe your experience with Kotlin in deep.

Study time: Study participation will take approximately 40 minutes in one session. There may be a future contact with the interviewee to solve questions that may appear in the course of the research.

Study location: All study procedures will be conducted via video conference or in loco, depending on the availability of both parties. I would like to audio-record this interview to make sure that I remember accurately all the information you provide. I will keep these tapes in a private repository and they will only be used to extract data for this research. I may quote your remarks in presentations or articles resulting from this work. A pseudonym will be used to protect your identity unless you specifically request that you be identified by your true name.

What are the possible risks or discomforts?

Your participation in this study does not involve any physical or emotional risk to you beyond that of everyday life.

As with all research, there is a chance that confidentiality of the information we collect from you could be breached – we will take steps to minimize this risk, as discussed in more detail below in this form.

What are the possible benefits for me or others?

You are not likely to have any direct benefit from being in this research study. This study is designed to learn more about the Usage of the Kotlin Programming Language for Android Development. The study results may be used to help other people in the future.

How will you protect the information you collect about me, and how will that information be shared?

Results of this study may be used in publications and presentations. Your study data will be handled as confidentially as possible. If results of this study are published or presented, individual names and other personally identifiable information will not be used.

To minimize the risks to confidentiality, we will keep all confidential data in a private repository in a trustful storage provider.

We may share the data we collect from you for use in future research studies or with other researchers – if we share the data that we collect about you, we will remove any information that could identify you before we share it.

Financial Information

Participation in this study will involve no cost to you. You will not be paid for participating in this study.

What are my rights as a research participant?

Participation in this study is voluntary. You do not have to answer any question you do not want to answer. If at any time and for any reason, you would prefer not to participate in this study, please feel free not to. If at any time you would like to stop participating, please tell me. We can take a break, stop and continue at a later date, or stop altogether. You may withdraw from this study at any time, and you will not be penalised in any way for deciding to stop participation. If you decide to withdraw from this study, the researchers will ask you if the information already collected from you can be used.

Who can I contact if I have questions or concerns about this research study? If you have questions, you are free to ask them now. If you have questions later, you may contact the principal researcher at:

Name: Victor Laerte de Oliveira

Email: vlo2@cin.ufpe.br

APPENDIX D – QUESTIONS PRESENTED IN THE RESULTS

Q3848

Title: How to download feature modules in an Android app?

Body: Now that instant apps are a thing (and a great thing, in my opinion), I was wondering if there is a way to actually being able to download certain parts of your app (modules) in order to reduce the initial size of the APK. For example, in my app I have a module that relies on a 3rd party library that increases the size of the APK in around 15 MB, and it only applies to certain users, so I would prefer to not to include that functionality in the final APK and make it optional. So far, while this looks very similar to Instant Apps, I haven't been been able to find a solution to this problem, and I totally believe that many apps could benefit from something like this. **Tags:** java, android, kotlin, android-instant-apps.

Q170

Title: What are the advantages of Kotlin programming language?

Body: I am quite eager to know the advantages of Kotlin programming language over Java for Android application development as I would love to explore new things. If any one have any knowledge about it please do needful. Thank you. **Tags:** android, mobile, kotlin, android-developer-api.

Q5187

Title: Can we build Kotin and Java Mix application?

Body: I started learning kotlin , but as I had good work experience in making android application in java language.So i want to use my native java experience with Kotlin.So I had some queries: If we used some of the files in Kotlin language and some in java? **Tags:** android, kotlin.

Q5535

Title: Why do I receive "as non-null is null" error after Android Studio converts Java code into Kotlin code automatically?

Body: When I copy and paste Code B (Java Code) into Android Studio 3.1.2, I choose to convert to Kotlin code automatically. So I get the shown Code A in Kotlin, but with the following error. Why? Why is that error occurring when Android Studio converts the Java code into Kotlin code automatically? BTW, Code B (Java code) works well. Error Code A (Kotlin Code) Code B (Java Code) **Tags:** android, kotlin.

Q1624

Title: In which situation val/var is necessary in Kotlin constructor parameter?

Body: Right code: cannot be resolved code: cannot be resolved code screenshot The only difference is the "val" has been deleted and cannot be resolve. Which might be important is that it's a inner class. BUT This one class without "val/var" in constructor parameter is working: And if I add var/val before "queue: RequestQueue", I'll get suggestion: "Constructor parameter is never used as a property less. This inspection reports primary constructor parameters that can have 'val' or 'var' removed. Unnecessary usage of 'val' and 'var' in primary constructor consumes unnecessary memory." I am just confused about it. **Tags:** android, kotlin.

Q9087

Title: Is it possible to pass lambda to Intent?

Body: I was wondering if it would be possible to pass a lambda to Intent in kotlin, since lambdas are Serializable, but with this code I am getting error when creating a PendingIntent. error: **Tags:** android, serialization, lambda, kotlin.

Q9381

Title: Force Android Studio to use gradle 4.1

Body: I have downloaded a previous version of Android Studio (3.0.1) since I must use the android gradle plugin version 3.0.1 and gradle 4.1. The code, and the other modules it depends on, were written using Android Studio 3.2, kotlin 1.3.10 and gradle version 4.10.2 When I try to synchronize the project with the older version of Android Studio (3.0.1) or running it says the minimum gradle version is 4.6 even though the gradle wrapper version is 4.1 and the gradle plugin version is 3.0.1. The kotlin version for all dependencies was set on 1.2.41 This is the error: How can I compile such project using older the older version? **Tags:** android-studio, gradle, kotlin, android-gradle.

Q4400

Title: Android 3.1 build gradle 4.4 error occurred configuring project ':app'

Body: When i build app to android studio 3.1 with emulator api<26 not error, but when i build api>26 error. i have not to use kotlin because not import kotlin,but build api>26 error. > kotlin.KotlinNullPointerException (no error message). com.android.build .gradle.tasks.ir.InstantRunMainApkResourcesBuilder$ConfigAction.execute(InstantRunMainApkResourcesBuilder.kt:129) Build gradle: Build gradle app: **Tags:** android, gradle, kotlin, android-gradle.

Q9383

Title: How add TextView to View in kotlin

Body: I have button when click I want to show (View) and inside it textview I did view but I can't add to it textView inside it this is what I want and too I want corner radius for View by code **Tags:** android, kotlin.

Q2979

Title: Android - How to change draw color of custom View?

Body: I thought my code would change the draw color of my custom view - but it doesn't (screen is just black), so how do I do change the draw color of a view in Android? You have to do it like this: **Tags:** android, kotlin.

Q6504

Title: Android, how to replace initial fragment?

Body: I create this fragment and set initial fragment , if I don't set this initial fragment, the app will crash. And then I use this function to replace the fragment, this works fine, every fragment can be replaced by other fragment except , this fragment keep displaying on screen. How can I replace this fragment? **Tags:** android, android-fragments, kotlin.

Q6570

Title: How to append 2 strings in Kotlin?

Body: I am trying to concatenate 2 String but not sure how to go about it. this is my code: and i'm trying to append it with inside the i tried to make it in this way and this way and it did not work , it only shows me numbers not the text **Tags:** android, kotlin, android-context.

Q1159

Title: Android Notification Not Showing On API 26

Body: I recently updated my app to API 26, and notifications are no longer working, without even changing the code. Why isn't it working? Was there some change to the API that I'm not aware of? **Tags:** android, notifications, kotlin, android-8.0-oreo.

Q3583

Title: Android ViewState using RxJava or kotlin coroutines

Body: I'm trying to learn how to use RxJava in Android, but have run into a dead end. I have the following DataSource: I want to trigger an update downstream, whenever I change the value of , but this doesn't happen. It works when the is initialized, but not when I'm updating the value. Here's my ViewModel, from where I update the value: This is the code from the that does the subsciption: **Tags:** android, rx-java2, kotlinx.coroutines.

Q2808

Title: How to show single item selected in recyclerview using kotlin

Body: How can we mark single item is selected in using kotlin. When I select an item and after that click on other item then previously selected item should be dis-selected.Here is my adapter class in kotlin: **Tags:** android, kotlin.

Q3839

Title: Retrofit parse result in Kotlin

Body: I have a webservice that takes username and password in a post request and returns a token (JWT) and a code if http statuscode is 200. If statuscode is 403 then the code contains the details and token is null. On iOS it is working but now I'm trying to implement it in Kotlin and Retrofit. What I have created so far: 2 DTOs: (JWT handling will be next step) A client service: and the code that calls the service: The request itself is working. It returns a 200 with correct login data and 403 with incorrect. But the LoginResultDto is empty. How can I populate the result in LoginResultDto? **Tags:** android, kotlin, retrofit2.

Q5096

Title: How to save captured photos as jpg files on android camera2

Body: current when I click capture button, show camera preview capture image on Imageview.. but I want to save camera preview capture image in jpg format. in my storage. How to save my capture image? on '/sdcard/DCIM/' folder. this source show camera preview capture on ImageView. I want capture photo, save jpg file. how to save capture image? thanks. **Tags:** android, kotlin, android-camera2.

Q773

Title: How to make primary key as autoincrement for Room Persistence lib

Body: I am creating a Entity(Room Persistence lib) class Food, where i want to make as autoincrement. How can i set an autoincrement field? **Tags:** android, kotlin, android-room.

Q5412

Title: Dagger2 + Kotlin: lateinit property has not been initialized

Body: I'm trying to inject the ViewModelFactory into my Activity, but it keeps throwing this same error: lateinit property viewModelFactory has not been initialized. I can't find what I may be doing wrong. See the code above from my classes AppComponent.kt MainModule.kt MainActivity.kt TweetSentimentsApplication.kt **Tags:** android, kotlin, dagger-2.

Q324

Title: How can I run a single Android Test using Kotlin?

Body: I am using Kotlin 1.0.3 for Android Development in Unit Testing but when I try to run a single test it runs all tests of the class. Does anyone know how to avoid that behaviour? **Tags:** android, unit-testing, junit, kotlin.

Q3853

Title: Non-null assert is needed even after checking is not null in kotlin

Body: I have this code in Kotlin in android studio: As you can see I check registerDate is not null, but I have to put non-null assert after to eliminate null error: Is this a bug in Kotlin or is related to Android Studio? **Tags:** android, kotlin.

Q216

Title: How can I override a java method, and change the nullability of a parameter?

Body: I'm overriding a method from a Java library, and the parameter for the function is annotated as . However, when the method is called, the parameter frequently comes in with a value. When I override the method in Kotlin, it forces me to respect the annotation and mark the parameter as not nullable. Of course, Kotlin throws an exception at run time when the parameter comes in with a null value. Is there some way I can override the method in Kotlin and ignore the annotation? Specifically, I'm using the appcompat library for Android. The method is in AppCompatActivity.java The override in Kotlin: **Tags:** java, android, nullable, kotlin.

Q2698

Title: Why do some Java setter methods automatically become Kotlin properties but some don't??

Body: e.g. this WebSettings Java class. It has a Java method that turns into a Kotlin property as below, but there is also that does not turn into a Kotlin property . **Tags:** android, properties, kotlin, kotlin-interop.

Q1486

Title: Kotlin: Setting a private Boolean in Java class via a Data class in Kotlin. Why am I not able to do this?

Body: I have a Java class of the format: And I am overriding this class into a Data class in Kotlin, which is of the format: When I do this, the name initialization this way doesn't give a problem, but I can't initialize x in this way. The IDE complains that x is invisible. Why with x and not with name? I created a new variable in the Kotlin class with the name x with a custom getter and setter and it complains about an accidental override for the setter (That is understandable.). This means that the the Java setter and getter is visible in the Data class. So why is the setter not being used for x in the init block, like it is doing for name? **Tags:** java, kotlin, kotlin-android-extensions.

Q3215

Title: Kotlin Generics Error in Java

Body: Given the following three kotlin classes: I am unable to compile following lines in java code: Error says: I am still new to kotlin and this might be something very small but I can't seem to figure it out. I will appreciate any help. **Tags:** java. android, generics, kotlin, kotlin-generics.

Q6396

Title: How to access static variable of java class in kotlin?

Body: Eg: I have a java class First First.java Second.kt So help me to call the static variable TAG declared in First.java inside Second.kt kotlin class **Tags:** java, android, kotlin.

Q4171

Title: Intent in Kotlin

Body: Why can't I just write class. instead of writing class.java. Because is a kotlin class and I am getting an error when I write this, How can I fix it. **Tags:** java, android, android-intent, kotlin.

Q6181

Title: How to pass a function as parameter in kotlin - Android

Body: How to pass a function in android using Kotlin . I can able to pass if i know the function like : I want to pass any function like -> **Tags:** android, function, parameters, kotlin.

Q5381

Title: What is better approach of callback in Kotlin? Listener vs High-Order function

Body: Please explain a use cases and pros and cons of each approach. Use of interface. Use of high-order function. **Tags:** android, kotlin.

Q3432

Title: How to make 'this' a reference of Listener instead of the Activity in Kotlin?

Body: I have a situation similar to the following example and when I call it references the not the . Is there way to make this a reference of the listener? **Tags:** android, kotlin.

Q1582

Title: Kotlin 'it' syntax in the context of Volley

Body: Can someone explain to me how the Kotlin 'it' syntax in this snippet of code works? This code is very hard to read, someone sent it to me to fix my problem and it works like magic for multiple consecutive requests. I have used Volley before but this code is very confusing. I believe Kotlin is easier to read than Java but this particular code is very hard to understand. **Tags:** android android-volley kotlin

Q2698

Title: Why do some Java setter methods automatically become Kotlin properties but some don't?

Body: e.g. this WebSettings Java class. It has a Java method that turns into a Kotlin property as below, but there is also that does not turn into a Kotlin property . **Tags:** android, properties, kotlin, kotlin-interop.

Q1585

Title: How to differentiate between a bound callable member reference and a function of the same type in kotlin?

Body: When it comes to a method signature or definition, is there any way to differentiate and ? That is, to specify that a bound member reference is required and not a function instance, or vice versa. Here is the signature of the above example: On the same note, is it possible to differentiate between a constructor reference and a function which just returns a type? E.g. vs. (without kotlin.reflect, if possible) **Tags:** java, android, jvm, kotlin.

Q3111

Title: Kotlin Type mismatch after converted in Android Studio

Body: I have tried to convert the existing the android code to Kotlin code. However, it showed the following error The code before conversion. The code after conversion. **Tags:** android, android-recyclerview, kotlin, android-studio-3.0.

Q8770

Title: Converting Java file to Kotlin now it won't compile - "Internal compiler error"

Body: I'm working on converting a Java Android project to Kotlin in Android Studio and I'm getting problems. It worked fine when I had converted one file to Kotlin with the MainActivity still being in Java but after converting the MainActivity the code no longer compiles. It fails with a Gradle / Kotlin related error and the stack trace isn't very useful and doesn't point to anything in my code. MainActivity.kt source Any ideas please? Here's the stack trace: Full build log: <https://paste.pound-python.org/show/XgPRLyb3JLBdsCZkUFPJ/> Thanks :) **Tags:** java, android, android-studio, gradle, kotlin.

Q4689

Title: Android unable to build project after updating kotlin runtime to 1.2.31

Body: After updating kotlin runtime to 1.2.31 from 1.2.30 this morning, I was unable to build the project. Below are the build.gradle in project level. What I did is only changing the line of the ext.kotlin_version to '1.2.31'. The android studio also prompted me to upgrade the gradle plugin to 4.4, which i did but I don't know if it's related to this issue. And the errors are as below And at the end, it also said **Tags:** android, gradle, kotlin.

Q4341

Title: After update to Android studio 3.1 i'm facing this erorr Could not find org.jetbrains.kotlin:kotlin-stdlib-jre8:1.2.0

Body: After updating to Android Studio 3.1, I'm facing this error. Note: I'm using Java not Kotlin **Tags:** java, android, android-studio, kotlin, android-gradle.

Q4784

Title: Warning "Kotlin plugin version is not the same as library version" (but it is!)

Body: I have an Android studio project in which I have added a Java library module, which I call . My three Gradle build files look like this. project/build.gradle core/build.gradle app/build.gradle The problem I have is that, in , the line is giving me the warning . I have tried changing it to: But the warning is still there. The build still runs successfully, and I know I can surpress the warning without any problems and ignore it, but I really want to know why this is happening and how I can get rid of it. I am using Android Studio 3.0.1. Does anyone know of a solution to this? **Tags:** android, android-studio, gradle, kotlin.

Q1002

Title: unable to configure Kotlin

Body: i am unable to configure Kotlin in my android studio. getting error Error:Unable to find method 'com.android.build.gradle.internal.variant.BaseVariantData.getOutputs()Ljava/util/List;'. **Tags:** android, kotlin.