How are Contracts Used in Android Mobile Applications?

David R. Ferreira
Faculty of Engineering, University of Porto
Porto, Portugal
david.regatia@gmail.com

Alexandra Mendes
HASLab / INESC TEC & Faculty of Engineering, University of Porto
Porto, Portugal
alexandra@archimendes.com

João F. Ferreira
INESC-ID & IST, University of Lisbon
Lisbon, Portugal
joao@joaoff.com

ABSTRACT

Formal contracts and assertions are effective methods to enhance software quality by enforcing preconditions, postconditions, and invariants. However, the adoption and impact of contracts in the context of mobile application development, particularly of Android applications, remain unexplored. We present the first large-scale empirical study on the presence and use of contracts in Android applications, written in Java or Kotlin. We consider 2,390 applications and five categories of contract elements: conditional runtime exceptions, APIs, annotations, assertions, and other. We show that most contracts are annotation-based and are concentrated in a small number of applications.

CCS CONCEPTS

• **General and reference → Empirical studies**; *Reliability*; • **Software and its engineering → Software evolution**.

KEYWORDS

design by contract, Android, assertions, Kotlin, Java

ACM Reference Format:
David R. Ferreira, Alexandra Mendes, and João F. Ferreira. 2024. How are Contracts Used in Android Mobile Applications?. In *2024 IEEE/ACM 46th International Conference on Software Engineering: Companion Proceedings (ICSE-Companion '24), April 14–20, 2024, Lisbon, Portugal.* ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/3639478.3643536

1 INTRODUCTION

Data from 2023 shows that Android represents approximately 43% of the overall operative system market share [10]. Therefore, faults in Android apps can impact a very large number of users. Also, with an increasing number of apps in critical areas such as health and finance, faults can have a huge negative impact. It is thus important to use software reliability techniques when developing these apps.

One of these techniques is Design by Contract (DbC) [6], under which software systems are seen as components that interact amongst themselves based on precisely defined specifications of client-supplier obligations (*contracts*). Many have advocated DbC as an efficient technique to aid the identification of failures, improve code understanding, and improve testing efforts, which directly or indirectly contribute to more reliable software. This has led to a number of empirical studies on the use of contracts in a variety of contexts [1–3, 5, 9]. However, *there are no previous studies on the presence and usage of contracts in Android applications nor any study that includes the Kotlin language.*

In this extended abstract, we present the first large-scale empirical study of contract usage in Android mobile applications written in Java or Kotlin. A longer version of this abstract presents more findings and considers evolution and safe usage of contracts [4].

Table 1: Contract elements considered in this study

category	examples
CREs (74 constructs)	`AccessControlException`, `IndexOutOfBoundsException`, `IllegalArgumentException`, `EmptyStackException`, ...

APIs (31 constructs)	`org.apache.commons.lang.Validate.*`
	`org.apache.commons.lang3.Validate.*`
	`com.google.common.base.Preconditions.*`

Assertions (6 constructs)	`assert` (Java), `assert` (Kotlin) `check(), checkNotNull()` (Kotlin) `require(), requireNotNull()` (Kotlin)

Annotations (136 constructs)	`org.jetbrains.annotations.*`
	`edu.umd.cs.findbugs.annotations.*`
	`android.annotation.*`
	`androidx.annotation.*`
	`javax.annotation.*` (JSR305)

Other (1 construct)	`@ExperimentalContracts` (Kotlin)

2 CONTRACTS IN ANDROID APPLICATIONS

Our notion of contract follows from the theory of *design by contract* [6], where preconditions, postconditions, and invariants are used to document (and specify) state changes that might occur in a program. Preconditions and postconditions are associated with methods and constrain their input and output values. Invariants are associated with classes and properties and constrain all the public methods in a given class. Preconditions represent the expectations of the contract, and postconditions represent its guarantees. Invariants represent the conditions that the contract maintains.

Java and Kotlin do not provide a native and standardized approach for contract specification, but developers can take advantage of language features and libraries to specify preconditions, postconditions, and class invariants in both languages. Similar to Dietrich et al. [2], we group these constructs into five categories: conditional runtime exceptions (CREs), APIs, annotations, assertions, and other. Since we focus on Android applications, we include contract elements that are specifically used by Android developers (e.g., Android annotations and specific Android runtime exceptions). Table 1 summarizes the classification and provides some examples; we consider a total of 248 constructs.

ICSE-Companion '24, April 14–20, 2024, Lisbon, Portugal

ACM ISBN 979-8-4007-0502-1/24/04.
https://doi.org/10.1145/3639478.3643536

Table 2: Number of contracts by construct and category.

Construct	Category	Java Contracts	Kotlin Contracts
cond. runtime exc.	CRE	14,887	2,071
unsupp. op. exc.	CRE	308	116
java assert	assertion	2,217	-
kotlin assert	assertion	-	2,370
guava precond.	API	1,121	9
commons validate	API	3	0
spring assert	API	1	0
JSR303, JSR349	annotation	0	0
JSR305	annotation	2,133	13
findbugs	annotation	0	0
jetbrains	annotation	1,596	98
android	annotation	7,013	3,414
androidx	annotation	86,212	13,811
kotlin contracts	others	-	1

3 DATASET

The dataset used is composed of real-world applications obtained from F-droid[1], written in Java or Kotlin. We consider all applications for which 1) the source code is hosted in GitHub; 2) the source code is either Java or Kotlin; 3) the GitHub project is not archived; 4) the GitHub project has had a commit since 2018. We clone all the Github projects. *Every file that is neither Java nor Kotlin is removed* from the dataset. From the initial list of 4,070 projects in the F-Droid index retrieved on May 21, 2023, we got 3,215 hosted in GitHub, 3,141 non-duplicated, and 2,390 projects after filtering by the inclusion criteria. Out of these, 1,767 are Java applications and 623 are Kotlin.

4 RESULTS

Table 2 shows the frequency of each construct. Annotations are the most popular category (this aligns with literature that supports annotations increasing popularity [12]). We also note that while Java's second most popular category is CREs, in Kotlin, it is assertions. This is explained by the inclusion of the four language's standard library methods listed in Table 1, where *require()* alone counts 901 total occurrences.

Table 3: Gini coefficient by category.

Category	Java	Kotlin
assertion	0.70	0.71
API	0.80	0.37
annotation	0.88	0.76
CRE	0.77	0.67
others	-	1.00

Finding 1: Most contracts are annotation-based, accounting for 88.31% in Java and 77.44% in Kotlin of the total of contracts found.

Table 2 also shows that the usage of *APIs* is very low in both languages, especially in Kotlin, where only nine instances were found. The known industry skepticism around adding third-party dependencies to projects, which may lead to maintainability and support issues in the future, may explain this finding [11].

Finding 2: The use of APIs to specify contracts is very rare.

Table 3 presents each category's *Gini coefficient*. A *Gini coefficient* of 0 means that all applications have the same number of contracts; a *Gini coefficient* of 1 means that a single program has all the contracts. All coefficients in the table are higher than 0.50, except for Kotlin's API usage. Almost all coefficients are high, meaning that although some applications use contracts intensively, the majority do not use them significantly. This aligns with Dietrich et al.'s results [2].

Finding 3: Although some applications use contracts intensively, the majority do not use them significantly.

5 CONCLUSION

Contracts are concentrated in a small number of applications. Still, when applications use contracts, annotation-based approaches are the most frequent, with the *androidx.annotation* package being the most popular. The use of APIs to specify contracts is rare. A longer version of this extended abstract presents additional findings and considers evolution and safe usage of contracts [4]. Future work includes the use of annotations to improve Android analysis tools [7, 8], and the development of tools that can help increase the adoption of DbC [13].

ACKNOWLEDGMENTS

This work is financed by National Funds through the Portuguese funding agency, FCT - Fundação para a Ciência e a Tecnologia, within project UIDB/50021/2020 (DOI:10.54499/UIDB/50021/2020) and project LA/P/0063/2020, DOI 10.54499/LA/P/0063/2020 | https://doi.org/10.54499/LA/P/0063/2020.

REFERENCES

[1] C. Casalnuovo, P. Devanbu, A. Oliveira, V. Filkov, and B. Ray. 2015. Assert Use in GitHub Projects. In *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering (ICSE). Proceedings*, Vol. 1. Los Alamitos, CA, USA.

[2] J. Dietrich, D. J. Pearce, K. Jezek, and P. Brada. 2017. Contracts in the Wild: A Study of Java Programs. In *31st European Conference on Object-Oriented Programming (ECOOP 2017) (Leibniz International Proceedings in Informatics (LIPIcs), Vol. 74)*. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany.

[3] H.-C. Estler, C. A. Furia, M. Nordio, M. Piccioni, and B. Meyer. 2014. Contracts in Practice. In *FM 2014: Formal Methods. 19th International Symposium. Proceedings: LNCS 8442*. Cham, Switzerland, 230 – 46.

[4] David R. Ferreira, Alexandra Mendes, and João F. Ferreira. 2024. Contract Usage and Evolution in Android Mobile Applications. arXiv:2401.14244 [cs.SE]

[5] P. Kochhar and D. Lo. 2017. Revisiting Assert Use in GitHub Projects. In *Proceedings of the 21st International Conference on Evaluation and Assessment in Software Engineering*. 298–307.

[6] B. Meyer. 1992. Applying 'design by contract'. *Computer* 25, 10 (1992), 40 – 51.

[7] Ricardo B Pereira, João F. Ferreira, Alexandra Mendes, and Rui Abreu. 2022. Extending Ecoandroid with automated detection of resource leaks. In *Proceedings of the 9th IEEE/ACM International Conference on Mobile Software Engineering and Systems*. 17–27.

[8] Ana Ribeiro, João F. Ferreira, and Alexandra Mendes. 2021. Ecoandroid: An Android studio plugin for developing energy-efficient Java mobile applications. In *2021 IEEE 21st International Conference on Software Quality, Reliability and Security (QRS)*. IEEE, 62–69.

[9] T. W. Schiller, K. Donohue, F. Coward, and M. D. Ernst. 2014. Case Studies and Tools for Contract Specifications. In *Proceedings of the 36th International Conference on Software Engineering* (Hyderabad, India) *(ICSE 2014)*. Association for Computing Machinery, New York, NY, USA, 596–607.

[10] StatCounter Global Stats. 2023. Operating System Market Share Worldwide. https://gs.statcounter.com/os-market-share#monthly-202208-202209-bar [Online; accessed 3-February-2023].

[11] Y. Wang, B. Chen, K. Huang, B. Shi, C. Xu, X. Peng, Y. Wu, and Y. Liu. 2020. An Empirical Study of Usages, Updates and Risks of Third-Party Libraries in Java Projects. In *2020 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. 35–45.

[12] Z. Yu, C. Bai, L. Seinturier, and M. Monperrus. 2021. Characterizing the Usage, Evolution and Impact of Java Annotations in Practice. *IEEE Transactions on Software Engineering* 47, 5 (2021), 969–986.

[13] Álvaro Silva, Alexandra Mendes, and João F. Ferreira. 2024. Leveraging Large Language Models to Boost Dafny's Developers Productivity. arXiv:2401.00963 [cs.SE]

[1]https://f-droid.org (accessed 6 June 2023)