2023 IEEE International Conference on Software Maintenance and Evolution (ICSME) | 979-8-3503-2783-0/23/$31.00 ©2023 IEEE | DOI: 10.1109/ICSME58846.2023.00077

PyAnaDroid: A fully-customizable execution pipeline for benchmarking Android Applications

Rui Rua
HASLab/INESC TEC, Portugal
Universidade do Minho, Portugal
Braga, Portugal
rui.a.rua@inesctec.pt

João Saraiva
HASLab/INESC TEC, Portugal
Universidade do Minho, Portugal
Braga, Portugal
saraiva@di.uminho.pt

***Abstract*—This paper presents *PyAnaDroid*, an open-source, fully-customizable execution pipeline designed to benchmark the performance of Android native projects and applications, with a special emphasis on benchmarking energy performance. *PyAnaDroid* is currently being used for developing large-scale mobile software empirical studies and for supporting an advanced academic course on program testing and analysis. The presented artifact is an expandable and reusable pipeline to automatically build, test and analyze Android applications. This tool was made openly available in order to become a reference tool to transparently conduct, share and validate empirical studies regarding Android applications. This document presents the architecture of *PyAnaDroid*, several use cases, and the results of a preliminary analysis that illustrates its potential.**
Video demo: https://youtu.be/7AV3nrh4Qc8

***Index Terms*—Mobile Testing; Energy Consumption; Benchmarking;**

I. Introduction

Improving energy efficiency in mobile devices and apps is a major concern for software developers, given the power limitations of such devices [1]. Consequently, extensive research in green software within the mobile ecosystem has focused on analyzing energy consumption through various approaches, including estimating app or code energy consumption [2]–[5], detecting anomalous coding patterns [6]–[8], and defining green/red APIs [5], [9], [10].

Although the scientific community has made significant efforts to provide insights into mobile app efficiency and programming practices, these studies typically rely on independent and non-reusable procedures, tools, and knowledge [11], [12]. Consequently, each new study often requires recreating the entire process, resulting in time-consuming setup, verification, and validation of data and results, which are seldom reused in subsequent studies.

To address these issues, we introduce *PyAnaDroid*, an open-source framework designed to optimize, enhance transparency, and improve replicability in empirical studies evaluating Android software artifacts' performance. Following the recent interest and necessity of evaluating mobile software energy performance, our tool is specially focused on energy performance, aiming to be used by both researchers and developers to analyze Android source code performance. *PyAnaDroid* offers a configurable workflow that analyzes source code, extracts metrics, and provides instrumentation capabilities. It also supports various testing frameworks, instrumentation techniques, profilers, and other application analysis tools, facilitating empirical studies by developers and researchers. The framework validates test results and conditions, ensuring result consistency and adherence to specified test conditions.

PyAnaDroid should not be considered a test framework or an energy profiler per se, but rather a versatile tool integrating state-of-the-art test frameworks and energy profilers for dynamic analysis of Android applications. Users can easily configure *PyAnaDroid* to process a set of applications, specify testing frameworks, test conditions, device parameters, and assert results validation.

To demonstrate *PyAnaDroid*'s capabilities, this paper presents preliminary results obtained through its execution. We utilized *PyAnaDroid* to analyze an initial set of 245 Android projects extracted from open-source repositories. All collected data, metrics, and execution procedures are openly available. This analysis aims to evaluate *PyAnaDroid*'s effectiveness in tasks such as building and analyzing Android projects and comparing the performance of different testing frameworks to achieve various test objectives.

II. Related Work

The Android ecosystem offers a wide range of alternatives for testing applications. Automating application execution using automated execution procedures, such as the ones provided by testing frameworks, is the most common method followed by the scientific community to evaluate the dynamic execution of Android [12], [13] apps: Correctness [14], security [15], or performance [16], [17] are among the main requirements evaluated by the community in their studies.

Using testing frameworks to exercise applications in order to evaluate source code performance is a common approach in the literature. When it comes to running a large number of tests on a set of variable-size apps, collecting or replicating app-specific real user inputs is impractical. Consequently, developers and researchers use testing frameworks to emulate user interaction. One of the most commonly used frameworks is UI/Application Exerciser Monkey (Monkey for short) [18], which has been used in many previous works [3], [4], [10], [14], [19] to help detect energy/performance-related issues.

2576-3148/23/$31.00 ©2023 IEEE
DOI 10.1109/ICSME58846.2023.00077

Unit testing frameworks such as JUnit or Robotium are used in scenarios where the source code is available [2], [14] or for developing app-specific scenarios.

In the literature, there are a few attempts to standardize procedures for benchmarking performance (more concretely, energy consumption) of Android apps. GreenDroid [2] was one of the first tools to automate the process of building and running Android projects to obtain estimates of their energy consumption. PETRA [3] was also included in a pipeline that performed source code instrumentation and was used alongside Monkey to exercise applications while monitoring energy consumption. Malavolta et al. [20] also presented a profiler-independent tool for the automation of measurement-based experiments on Android devices. This tool was conceived to be profiler-independent and used to replicate the performance-based study of Android apps using a black-box approach, being able to process native and web-based apps. However, all these tools have limited applicability. None of these tools is able to automatically instrument and build modern applications with state-of-the-art tools. GreenDroid is only able to exercise Java apps using unit tests and profile applications with Power Tutor [21] (a deprecated profiler). PETRA was only able to exercise apps with Monkey and is not suitable for conducting large-scale empirical studies, since it relies on an GUI to conduct the testing and profiling process. The tool presented in Malavolta et al. is not able to automatically build Android projects or execute apps with different testing frameworks.

III. The *PyAnaDroid* Framework

This section provides a overview of the *PyAnaDroid* framework, presenting its main components and features and demonstrating how it can assist practitioners in analyzing and benchmarking Android applications. Motivated by the open-source movement, this tool is openly available for the community to use and contribute, aiming to standardize and make transparent the empirical experimental process of analyzing applications' energy consumption. *PyAnaDroid* itself is inspired by an open-source tool, GreenDroid [2], which was the first execution flow that involved instrumenting, building, and running apps on physical device using JUnit tests, while also estimating energy consumption.

PyAnaDroid is a highly configurable pipeline designed for empirical studies involving dynamic analysis of software. It consists of multiple blocks or modules, each representing a common static/dynamic analysis task in such studies. These modules are implementations of state-of-the-art tools or procedures widely used in the literature. The tool can be easily extended in order to support new workflows or modules, by extending or implementing the modules or workflow interface. *PyAnaDroid* allows generating and replicating workload over a set of applications, being able to monitor the execution of this workload and produce reports regarding the application's performance. In addition to providing several implementations of each of its blocks, the framework can easily support other relevant blocks for app analysis, such as refactoring or code repairing blocks. Furthermore, *PyAnaDroid* includes essential features for dynamic analysis studies, including test condition enforcement to ensure tests are conducted under specific device conditions and test validation to detect anomalies by enforcing a set of post-execution checks.

The *PyAnaDroid* architecture follows recent state-of-the-art guidelines [22], being able to perform both white-box and black-box testing, depending on the selected testing procedure. It is also able to automatically instrument Android projects in order to include tracing calls to obtain dynamic metrics and integrate performance profiling tools. It can also automatically build Android projects and install the resulting APK(s) on a given device. The installed apps can be automatically executed by the supported testing frameworks while their execution is being monitored by an energy profiler. Finally, it can analyze and validate the execution of each test, and present the final results and metrics to the user.

PyAnaDroid was developed to be easily integrated with the typical Android development environment and can be used to interact with any Android device or emulator. Furthermore, it was designed to be easily installable and configurable, resulting in a plug-and-play experience on any machine with Android SDK configured. Furthermore, it is available as a Pypi package in order to be easily installed with a single command in any Python development environment, relying solely on Android SDK tools and open-source libraries.

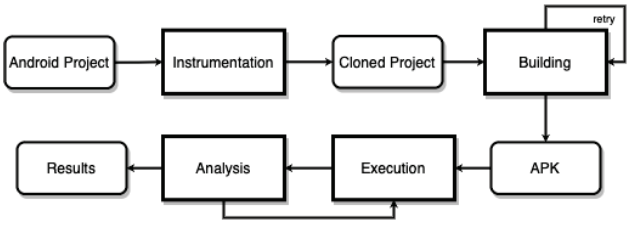

Fig. 1. *PyAnaDroid* default workflow

The framework offers several implementations of each block in the execution pipeline, allowing users to replace software components with minimal effort or develop new parts to integrate into the pipeline. Fig. 1 contains the default workflow of the pipeline when an Android project is provided as input. Each stage of *PyAnaDroid* is represented by a square, while the inputs/outputs of each block are represented by the rounded blocks. These blocks can be interchanged so as to define different execution pipelines. The framework supports different types of input, which we will present in Section III-A and originate different execution pipelines.

In the default workflow, when an Android project is provided, the project enters the Instrumentation stage, which clones the project and transforms the project's sources (Section III-B). Afterward, the cloned project is built using the building block (Section III-C). The expected output of such block is one or more APKs, consumed by a process illustrated in Figure 1 as *Execution*, which is detailed in Section III-D. This process can have several iterations, which can be validated by the *Analysis* block described in Section III-E.

A. Input

This section describes the first block of the execution pipeline. *PyAnaDroid* supports 3 different types of inputs to consume, whose type is inferred by the arguments passed to the CLI interface or the class constructor: *packages name(s)* (name of the packages of the apps to be tested), *APK(s)* (path of the APKs to test), and *projects' directory* (the directory with Android projects to be processed). In addition to these three types of inputs that are processed by the execution pipeline, the framework also supports several input options that allow configuring aspects related to pipeline blocks, test framework configuration parameters, type of building to be performed, etc.

B. Instrumentation

Currently, *PyAnaDroid* is able to perform source code instrumentation, both at pre-processing and compile-time. Such instrumentation is performed before the building stage.

PyAnaDroid is able to perform two different types of Java and Kotlin source code instrumentation, available through 2 different instrumentation blocks, such as applications' methods. The first implementation consists of instrumenting methods with calls to TrepnLib [23] in order to estimate the power and resources consumption of such instrumented blocks, as well as log other relevant events. The latter, Annotation-Oriented, aims to perform method tracing and temporal localization of the execution of methods in the device's system logs. The latter is automatically performed by injecting annotations on the application's methods using a compile-time plugin.

C. Building Tools

This module performs a function that is often a major obstacle to perform empirical studies from source code with a large number of applications: building third-party Android apps from source. Many studies in the literature involving automated testing in Android rely on open-source repositories to obtain its corpus of applications [24]. However, when one intends to obtain and analyze a significant corpus of applications, a procedure to automate their building is essential. Unfortunately, many of these applications are configured to be built only on specific environments or devices, requiring a tedious manual setup to be built and executed. *PyAnaDroid* offers a building block that allows not only to automate the build process of several Android projects but also to solve problems of build or integration with the current development environment, instrumentation libraries or profilers. Furthermore, the framework contains a single implementation of the building block, that provides support for the Gradle [25] building tool: the default building system for building native Android apps. This block tries to build Android projects in an iterative procedure, where it parses the output of the building commands and repair errors based on a set of pre-defined heuristics. Among the building errors solved by the current implemented block are missing build tools, deprecated configurations, invalid local or NDK properties, dependencies' incompatibilities, among others. Besides being able to solve these building errors, the block is also able to inject custom configurations in the building scripts, such as, for instance, *lint* or *dex* options.

D. Dynamic Execution

This *PyAnaDroid* stage is composed of one or more procedures that involve the interaction and management of the test device with two block implementations: the *testing frameworks* block and the *profiler* block. The typical test workflow consists of unlocking the device screen (if locked), clearing device logs, and then starting the profiler and the monitoring process. Afterward, the app is opened and the test is executed. At the end of the execution, the app is closed and the profiler is turned off. The device logs are cleared and the monitoring results are exported and extracted from the device, right before the application cache being cleared too. Finally, the results are validated using an Analysis block, repeating the test if this block rejects its results, optionally until a maximum of pre-defined times is reached.

1) ***Testing frameworks****: PyAnaDroid* takes advantage of testing frameworks to exercise the Android applications and respective code. Our tool currently supports several of the most common frameworks that allow to easily interact and execute applications on Android devices. These frameworks can be used in the form of interchangeable blocks whose configurations and test parameters can be configured with minimal effort through configuration files. Currently, the tool supports 9 testing frameworks, of several testing types. Beside of the supported testing frameworks, *PyAnaDroid* also supports the execution of custom tests whose invoking command(s) can be passed via command-line. The test suites to be executed can be configured via configuration files, as well as other testing parameters (number of executions, timeouts, etc). Table I shows the testing frameworks currently supported by *PyAnaDroid*.

TABLE I
TESTING FRAMEWORKS SUPPORTED BY *PyAnaDroid*

Framework	Requires source	Description
Monkey [18]	No	generates pseudo-random system events
JUnit [26]	Yes	Unit testing
UI Automator [27]	No	functional UI testing
Robotium [28]	No	UI testing via component abstraction
Espresso [29]	Yes	UI testing of view components
RERAN [30]	No	Record and Replay framework
DroidBot [31]	No	Model-based UI framework
App Crawler [32]	No	GUI and system events
MonkeyRunner [33]	No	functional and regression testing

In addition to these frameworks, others that have not yet been tested based on JUnit might also be supported, such as dexmaker [34]. Other frameworks can easily be added by implementing the interface defined for the testing block.

2) ***Profilers****:* To monitor apps' execution, *PyAnaDroid* provides a block that aims to define an interface for communication and management of monitoring tools. Currently, 3 different implementations of such interface are provided, that integrate 3 different energy profilers, namely: Trepn Profiler [35], E-Manafa [36] and GreenScaler [4]. By default, *PyAnaDroid* uses E-Manafa, since it it the most suitable

device-independent solution to be used with state-of-the-art devices. This open-source profiler that allows obtaining performance measurements on Android devices, such as energy consumption, runtime, CPU frequency, or resource usage. Furthermore, it can be used on any device that has these services (all devices running version 9.0 or later of the Android platform).

E-Manafa is an open-source profiler that allows obtaining performance measurements on Android devices, such as energy consumption, runtime, CPU frequency, or resource usage. To obtain these metrics, it uses as a data source, the power profile file present on each device, the *batterystats* service (to obtain data on resource usage) and *perfetto* [37] (to sample the CPU frequency). Furthermore, it can be used on any device that has these services (all devices running version 9.0 or later of the Android platform).

E. Analysis Tools

Currently, there are already several implementations of this block in the pipeline, which are executed consecutively or concurrently to obtain results from the analysis of several elements, such as source code, logs, devices, and services. The metrics and results provided by each one of the analyzers are compiled in results files and reports in JSON format. Some of the most relevant analyzers are analyzers to analyze logs collected from the device during application execution to detect errors that occurred during app executions, analyzers for validate results and produce reports of the monitoring process with the energy profilers or source code and APK analyzers to derive static metrics from the project source code. For instance, *PyAnaDroid* provides an implementation of the SCC Analyzer, that uses *scc* [38] to extract metrics such as the languages used, Lines Of Code, Cyclomatic Complexity [39], among others.

IV. Examples and Results

In this section, we briefly describe the procedure and results of a preliminary dynamic analysis performed over a large set of 245 Android projects using *PyAnaDroid*, whose execution did not involved any human interaction. With this preliminary study, we intend to demonstrate some use cases of our framework, as well as to evaluate the advantages of using this tool to carry out large-scale studies. Among the presented use cases are operational acceptance testing and comparing the effectiveness of testing approaches and frameworks to detect run-time errors, explore energy hot-spots or maximize code coverage. The entire analytical process was made openly available [1], with the respective instructions for its replication. Most of the collected results and respective analysis are provided in an online appendix [2].

The conducted empirical process carried out to process the Android projects and respective applications involved transforming and building each of these projects 3 times, which means that it took approximately the same time to build 3 times the number of applications when executed sequentially. The test process consisted of executing the tool over Android Projects via 4 different commands.

To demonstrate the advantages of using *PyAnaDroid* to automate the project building process, our framework was executed with 3 different building approaches:

- ***naive*** approach: Sets up Gradle wrapper and properties file to automate the building process. Intends to simulate a limited workflow that could be included in an Android project building automation process (as designed in [2]) and . It aims to perform a workload equivalent to the manual of work of performing a minimal setup effort and invoke the Gradle building commands.
- ***default*** approach: capabilities of the previous approach plus the ability to automatically solve several building errors (missing build tools, deprecated configurations, invalid local/NDK properties, etc).
- ***default+I*** approach: Equivalent to the previous approach, but with source code instrumentation to evaluate the instrumentation process' intrusiveness.

The results obtained by applying the 3 building approaches aim to perform 2 evaluations corresponding to 2 potential use cases: **a)** if open-source Android applications have their code ready to be built and executed outside the development environment; **b)** The accuracy of black-box testing frameworks in detecting runtime errors on Android apps. Consequently, the last option presented (***default+I***) was also combined with the execution of tests with Monkey and Droidbot frameworks. Since these two frameworks allow black-box testing of applications using various test loads and parameters, running applications with 2 different frameworks allows the possibility of detecting errors in applications as well as to establish comparisons between the effectiveness of test frameworks for automating the execution of Android applications.

In short, the empirical process can be summarized by the execution of 4 commands of the framework:

```
$ pyanadroid -d <apps_dir> -nv -i None -p None -bo
$ pyanadroid -d <apps_dir> -i None -p None -bo
$ pyanadroid -d <apps_dir> -t Monkey -p EManafa
$ pyanadroid -d <apps_dir> -t Droidbot -p EManafa
```

All the options used are described in the *PyAnaDroid* documentation and in the online appendix. The -d option precedes the location of the projects under analysis. The -nv option of the first command runs the process with the ***naive*** builder, while the -i and -p options specify the instrumentation and profiler. Using the -bo (build only) option present in the first 2 commands, the processing workflow of each project ends after the building phase. The -t option specifies the testing framework for the execution process. The tool rebuilt the projects in the second and third commands since they were built with different configurations. In the last command, the projects are not rebuilt, since the instrumentation and build process is the same.

This study considered an initial diverse set of randomly selected 250 apps with release packages present on GitHub. In order to gather our dataset, we used the information present in AndroZoOpen [40], by using a custom crawler [41] that

[1]Execution process used in this work: https://shorturl.at/qrHSX
[2]**Online appendix:** https://shorturl.at/ekyIU

is able to download and filter the projects of the dataset. The data selection and filtering process are described in the online appendix. Static metrics collected regarding the projects' sources are also available in tabular format.

TABLE II
BUILDING APPROACHES ABILITY

Approach	#Projects	%
naive	113	46,12%
default	129	52,66%
default+i	102	41,63%

TABLE III
DYNAMIC EXECUTION METRICS

Metric	Scope	Monkey	Droidbot	Unit
Energy	Test	17.91	204.67	J
Power	Test	1.91	1.85	W
Energy	Method	0.0023	0.12	J
Run-time	Test	9.36	110.79	s
Method coverage	Test	5.06	11.42	%
Errors	Test	79.5	156	#
Java Exceptions	Test	1	7	#

Table II shows the results of applying the 3 building strategies to our initial set of Android projects. All numerical values presented in this table, independently of the corresponding unit, are presented in terms of median. Our results show that *PyAnaDroid* improves building ability by approximately 14% over the *naive* alternative. Building with instrumentation(*default+i*) proved to be more intrusive for the building applications (since it requires build files instrumentation to include a compile-time plugin for the instrumentation process), having only managed to build 102 applications. By observing the significant failure rate of the *naive* approach and by comparing with *default* and *default+i*, it is possible to observe that the usage of *PyAnaDroid* does not significantly increases the failure rate. This evidence also reveals that building open-source Android projects typically raises configuration and replication issues, with this problems being caused by either missing or erroneous environment and project configurations, such as incompatibility issues, missing dependencies, etc.

Table III presents a set metrics extracted by the analysis block of the framework after dynamically executing the apps. The measurements presented in this Table represent the median values obtained during the testing procedure, accounting for the energy consumption of the applications during the testing procedure and the performance overhead of the testing framework being executed alongside the application. The results allow us to draw comparisons regarding the accuracy and effectiveness of the selected testing frameworks to meet certain testing objectives. For instance, for the selected configurations for each framework, we can observe that Monkey consumes more energy per second (Power, Watts), but less energy overall, since generated its inputs more quickly. However, since the testing sessions with Droidbot were longer (caused mostly by its startup time), Droidbot significantly consumed more energy overall. Nevertheless, Droidbot was able to detect more Java Exceptions and Errors, while also providing more method coverage. This evidence suggests that this framework might be more suitable for performing functional testing and more effective to explore applications' code. Nevertheless, to obtain meaningful and representative results for the supported testing frameworks, a more elaborated study must be conducted to demonstrate the potential use case for the *PyAnaDroid* framework. For instance, our tool is suitable to conduct studies aiming to estimate and compare the performance overhead of testing frameworks [42].

V. CONCLUSIONS

This work presents *PyAnaDroid*, a tool to automatically build, analyze and run Android applications, using different testing strategies. As far as the authors acknowledge, there is no other capable of performing native source code instrumentation and performing automatic building and execution of the code with different profilers and testing frameworks. The tool is completely open-source and its operation and results can be easily validated and extended by the community. *PyAnaDroid* is under continuous development and supporting an advanced course in software testing and analysis. In addition to the development by the authors, it is expected that the scientific community and Android programmers can also contribute to its development.

The results presented in this work demonstrate that it is possible to use a generic pipeline that can transform and analyze source code, build applications and execute them automatically, with a high degree of customizability. *PyAnaDroid* can be used by practitioners and researchers for several purposes, such as detection of bottlenecks or inefficient coding practices, calibration of energy monitoring tools, comparison of testing frameworks and energy profilers, mutation and regression testing, among others. Furthermore, the performance benchmarking step is optional, being also suitable for automating the dynamic execution of non-performance-related tasks over sets of applications, such as accessibility checking, UI testing, among others.

VI. ACKNOWLEDGMENTS

This work is financed by National Funds through the Portuguese funding agency, FCT - Fundação para a Ciência e a Tecnologia, within project UIDP/50014/2020. The first author is also financed by FCT grant SFRH/BD/146624/2019.

REFERENCES

[1] G. Pinto and F. Castor, "Energy efficiency: a new concern for application software developers," *Communications of the ACM*, vol. 60, no. 12, pp. 68–75, 2017.

[2] M. Couto, J. Cunha, J. P. Fernandes, R. Pereira, and J. Saraiva, "Greendroid: A tool for analysing power consumption in the android ecosystem," in *2015 IEEE 13th International Scientific Conference on Informatics*, Nov 2015, pp. 73–78.

[3] D. D. Nucci, F. Palomba, A. Prota, A. Panichella, A. Zaidman, and A. D. Lucia, "Petra: A software-based tool for estimating the energy profile of android applications," in *2017 IEEE/ACM 39th International Conference on Software Engineering Companion (ICSE-C)*, May 2017, pp. 3–6.

[4] S. Chowdhury, S. Borle, S. Romansky, and A. Hindle, "Greenscaler: Training software energy models with automatic test generation," *Empirical Softw. Engg.*, vol. 24, no. 4, p. 1649–1692, Aug. 2019.

[5] A. Pathak, Y. C. Hu, M. Zhang, P. Bahl, and Y.-M. Wang, "Fine-grained power modeling for smartphones using system call tracing," in *Proceedings of the Sixth Conference on Computer Systems*, ser. EuroSys '11. New York, NY, USA: ACM, 2011, pp. 153–168.

[6] L. Cruz and R. Abreu, "Performance-based guidelines for energy efficient mobile applications," in *2017 IEEE/ACM 4th International Conference on Mobile Software Engineering and Systems (MOBILESoft)*, May 2017, pp. 46–57.

[7] R. Pereira, M. Couto, J. Cunha, J. P. Fernandes, and J. Saraiva, "The influence of the java collection framework on overall energy consumption," in *2016 IEEE/ACM 5th International Workshop on Green and Sustainable Software (GREENS)*. IEEE, 2016, pp. 15–21.

[8] R. Rua, M. Couto, A. Pinto, J. Cunha, and J. Saraiva, "Towards using memoization for saving energy in android," in *Proceedings of the XXII Iberoamerican Conference on Software Engineering, CIbSE 2019, La Habana, Cuba, April 22-26, 2019*, 2019, pp. 279–292.

[9] L. Zhang, C. Stover, A. Lins, C. Buckley, and P. Mohapatra, "Characterizing mobile open apis in smartphone apps," in *2014 IFIP Networking Conference*, June 2014, pp. 1–9.

[10] M. Linares-Vásquez, G. Bavota, C. Bernal-Cárdenas, R. Oliveto, M. Di Penta, and D. Poshyvanyk, "Mining energy-greedy api usage patterns in android apps: An empirical study," in *Proceedings of the 11th Working Conference on Mining Software Repositories*, ser. MSR 2014. New York, NY, USA: ACM, 2014, pp. 2–11.

[11] S. R. Choudhary, A. Gorla, and A. Orso, "Automated test input generation for android: Are we there yet? (e)," in *Proceedings of the 2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, ser. ASE '15. Washington, DC, USA: IEEE Computer Society, 2015, pp. 429–440.

[12] P. Kong, L. Li, J. Gao, K. Liu, T. F. Bissyandé, and J. Klein, "Automated testing of android apps: A systematic literature review," *IEEE Transactions on Reliability*, vol. 68, no. 1, pp. 45–66, 2019.

[13] J. W. Lin, N. Salehnamadi, and S. Malek, "Test automation in open-source android apps: A large-scale empirical study," in *2020 35th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, 2020, pp. 1078–1089.

[14] X. Li, Y. Yang, Y. Liu, J. P. Gallagher, and K. Wu, "Detecting and diagnosing energy issues for mobile applications," in *Proceedings of the 29th ACM SIGSOFT International Symposium on Software Testing and Analysis*, ser. ISSTA 2020. New York, NY, USA: Association for Computing Machinery, 2020, p. 115–127.

[15] M. Egele, D. Brumley, Y. Fratantonio, and C. Kruegel, "An empirical study of cryptographic misuse in android applications," in *Proceedings of the 2013 ACM SIGSAC Conference on Computer Communications Security*, ser. CCS '13. New York, NY, USA: Association for Computing Machinery, 2013, p. 73–84.

[16] R. Rua, T. Fraga, M. Couto, and J. a. Saraiva, "Greenspecting android virtual keyboards," in *Proceedings of the IEEE/ACM 7th International Conference on Mobile Software Engineering and Systems*, ser. MOBILESoft '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 98–108.

[17] C. Wilke, C. Piechnick, S. Richly, G. Püschel, S. Götz, and U. Aßmann, "Comparing mobile applications' energy consumption," in *Proceedings of the 28th Annual ACM Symposium on Applied Computing*, ser. SAC '13. New York, NY, USA: Association for Computing Machinery, 2013, p. 1177–1179.

[18] Google. (2021) Ui/application exerciser monkey. Last visit: 2023-02-03. [Online]. Available: https://developer.android.com/studio/test/monkey

[19] Y. Hu, J. Yan, D. Yan, Q. Lu, and J. Yan, "Lightweight energy consumption analysis and prediction for android applications," *Science of Computer Programming*, vol. 162, 05 2017.

[20] I. Malavolta, E. M. Grua, C.-Y. Lam, R. de Vries, F. Tan, E. Zielinski, M. Peters, and L. Kaandorp, "A framework for the automatic execution of measurement-based experiments on android devices," in *2020 35th IEEE/ACM International Conference on Automated Software Engineering Workshops (ASEW)*, 2020, pp. 61–66.

[21] L. Zhang, B. Tiwana, R. P. Dick, Z. Qian, Z. M. Mao, Z. Wang, and L. Yang, "Accurate online power estimation and automatic battery behavior based power model generation for smartphones," in *2010 IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*, 2010, pp. 105–114.

[22] A. Schuler and G. Anderst-Kotsis, "Towards a framework for detecting energy drain in mobile applications: An architecture overview," in *Companion Proceedings for the ISSTA/ECOOP 2018 Workshops*, ser. ISSTA '18. New York, NY, USA: ACM, 2018, pp. 144–149. [Online]. Available: http://doi.acm.org/10.1145/3236454.3236504

[23] R. Rua, M. Couto, and J. Saraiva, "Greensource: A large-scale collection of android code, tests and energy metrics," in *2019 IEEE/ACM 16th International Conference on Mining Software Repositories (MSR)*, May 2019, pp. 176–180.

[24] P. Kong, L. Li, J. Gao, K. Liu, T. F. Bissyandé, and J. Klein, "Automated testing of android apps: A systematic literature review," *IEEE Transactions on Reliability*, vol. 68, no. 1, pp. 45–66, 2019.

[25] Gradle. (2023) Gradle. [Online]. Available: https://gradle.org

[26] JUnit. (2022) Junit. [Online]. Available: https://junit.org/junit5/

[27] Google. (2022) Write automated tests with ui automator. [Online]. Available: https://shorturl.at/AEKN2

[28] RobotiumTech. (2022) Robotium: User scenario testing for android. [Online]. Available: https://github.com/RobotiumTech/robotium

[29] Google. (2022) Espresso. [Online]. Available: https://developer.android.com/training/testing/espresso

[30] L. Gomez, I. Neamtiu, T. Azim, and T. Millstein, "Reran: Timing- and touch-sensitive record and replay for android," in *2013 35th International Conference on Software Engineering (ICSE)*, 2013, pp. 72–81.

[31] Y. Li, Z. Yang, Y. Guo, and X. Chen, "Droidbot: a lightweight ui-guided test input generator for android," in *2017 IEEE/ACM 39th International Conference on Software Engineering Companion (ICSE-C)*, 2017, pp. 23–26.

[32] Google. (2021) App crawler. Last visit: 2023-02-03. [Online]. Available: https://developer.android.com/training/testing/crawler?hl=en

[33] ——. (2022) Monkeyrunner. [Online]. Available: {https://developer.android.com/studio/test/monkeyrunner}

[34] Linkedin. (2018) Dexmaker. [Online]. Available: https://github.com/linkedin/dexmaker

[35] Google. (2016) Trepn profiler android app. Last visit: 2023-02-03. [Online]. Available: https://play.google.com/store/apps/details?id=com.quicinc.trepn

[36] R. Rua and J. a. Saraiva, "E-manafa: Energy monitoring and analysis tool for android," ser. ASE22. New York, NY, USA: Association for Computing Machinery, 2022. [Online]. Available: https://doi.org/10.1145/3551349.3561342

[37] Google. (2022) Perfetto: System profiling, app tracing and trace analysis. [Online]. Available: https://perfetto.dev

[38] (2022) Sloc cloc and code (scc). [Online]. Available: https://github.com/boyter/scc#sloc-cloc-and-code-scc

[39] T. J. McCabe, "A complexity measure," in *Proceedings of the 2Nd International Conference on Software Engineering*, ser. ICSE '76. Los Alamitos, CA, USA: IEEE Computer Society Press, 1976, pp. 407–.

[40] P. Liu, L. Li, Y. Zhao, X. Sun, and J. Grundy, "Androzooopen: Collecting large-scale open source android apps for the research community," ser. MSR '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 548–552. [Online]. Available: https://doi.org/10.1145/3379597.3387503

[41] (2022) Androzooopencrawler. [Online]. Available: https://github.com/greensoftwarelab/androzoopen-crawler

[42] L. Cruz and R. Abreu, "Measuring the energy footprint of mobile testing frameworks," in *Proceedings of the 40th International Conference on Software Engineering: Companion Proceeedings*, ser. ICSE '18. New York, NY, USA: ACM, 2018, pp. 400–401. [Online]. Available: http://doi.acm.org/10.1145/3183440.3195027