Journal of Basics and Applied Sciences Research (JOBASR)
ISSN (print): 3026-9091, ISSN (online): 1597-9962
Volume 3(4) July 2025
DOI: https://dx.doi.org/10.4314/jobasr.v3i4.25

Android Applications Code Coverage Tools: A Comparative Study

Usman Asmau[1], Salihu Anka Ibrahim[2], Muazu Aminu Aminu.[3*], Eke Oke Ndukwe[4] & Usman Mohammed Abubakar[5]

[1]Department of Computer Science, Faculty of Computing, Nile University of Nigeria
[2]Department of Software Engineering, Faculty of Computing, Nile University of Nigeria
[3*]Department of Computer Science, Faculty of Natural and Applied Sciences, Umaru Musa Yar'adua University Katsina, Nigeria
[4]Wigwe University, Isiokpo, Rivers State, Nigeria
[5]Department of Computer Science, Federal University Gusau, Nigeria
[*]Corresponding Author Email: aminu.aminu@umyu.edu.ng

ABSTRACT
The widespread adoption of mobile applications necessitates robust quality assurance methods to ensure reliable software behavior. Among these methods, software testing plays a vital role, with code coverage serving as a key metric for evaluating test effectiveness. This study presents a comparative analysis of prominent code coverage tools specifically designed for Android applications. Through an extensive literature review and evaluation of thirteen tools—including Emma, Jacoco, COSMO, WallMauer, and ACVTool—this research highlights their instrumentation strategies, integration capabilities, validation methods, and reporting metrics. The study emphasizes the importance of granularity in coverage (e.g., method, line, and instruction), and the trade-offs between bytecode and source code instrumentation. The findings aim to guide developers and researchers in selecting appropriate tools for enhancing testing coverage in Android app development.

Keywords:
Android, Mobile Application, Code Coverage, Software Testing

INTRODUCTION

Software testing is a widely adopted practice for maintaining the quality of software systems. To assess and enhance the effectiveness of current test suites, code coverage metrics are frequently employed. Coverage serves as a key metric for evaluating the effectiveness of the testing process (Memon, Soffa, & Pollack, 2001; Usman, Ibrahim, & Salihu, 2020). Code coverage is a crucial metric employed in these techniques to assess their effectiveness, and it is often used as a fitness function to enhance outcomes in evolutionary and fuzzy-based methods (Eke, Salihu, & Usman, 2023; Shahid, Ibrahim, & Mahrin, 2011). Achieving a high percentage of test coverage for a specific program is one of the primary objectives of the software testing process.

Criteria for evaluating the testing process's completeness are represented by testing coverage. A variety of testing tasks can be carried out in order to achieve test coverage. Code testing coverage indicates whether sections of a program's code are examined by at least one test case (Pathy, Panda, & Baboo, 2015; Usman, Ibrahim, Sulaiman, & Salihu, 2024).

When testing Android apps, it helps to combine several granularities from instruction, method, and activity coverage to get better results (Usman, Boukar, Suleiman, & Salihu, 2024). Because activities and methods are essential to app development, the activity and method coverage numbers are clear and instructive (Azim & Neamtiu, 2013). The main interface for user interaction is called an activity, which consists of a number of methods and underlying code logic (Salihu & Ibrahim, 2016). All activity's methods have varying numbers of lines of code. Similar to this, activity coverage is a condition for identifying crashes that might happen while using the user interface of the application (Dashevskyi, Gadyatskaya, Pilgun, & Zhauniarovich, 2018). The program is more likely to find possible crashes the more coverage it looks at (Dashevskyi et al., 2018). Instruction coverage indicates how much of the code has been executed during testing. Therefore, enhancing both instruction and method coverage helps ensure that a greater portion of the application's functionalities linked to each activity are thoroughly explored and tested (Azim & Neamtiu, 2013; S. Yang, Huang, & Hui, 2019).

Likewise, activity coverage is essential for identifying crashes that may occur during interactions with the app's user interface. The greater the coverage achieved by the tool, the higher the likelihood of uncovering potential crashes (Dashevskyi et al., 2018).

Furthermore, in white-box testing, code coverage criteria are commonly set as targets, guiding the design of test cases until the desired level of coverage is achieved based on the chosen metrics (Usman, Ibrahim, & Salihu, 2018). However, there are numerous ways to define these criteria, which may vary in terms of analysis granularity such as component, method, or statement level and the specific aspects of the program code being exercised, including individual instructions, code blocks, control paths, data paths, and more (Pilgun et al., 2020).

The term code coverage when used without further clarification, typically refers to statement-level analysis, also known as statement coverage. This type of coverage indicates which program instructions are executed during testing and which remain untested. However, even at this level, variations in how an instruction is defined can lead to inconsistencies in interpreting the results (Auer, Arcuschin Moreno, & Fraser, 2024). In Java, for example, a single line of source code may correspond to multiple bytecode instructions, and the relationship between them can be complex due to factors like compiler optimizations. Another frequently used coverage criterion is decision coverage, which focuses on whether both possible outcomes of a decision—such as the true and false branches of an `if` statement—have been tested, as well as whether loops are evaluated both by executing and skipping their bodies (Memon et al., 2001). Because this level of analysis involves not just individual instructions but also control flow, measuring coverage becomes more complex and introduces additional challenges (Horváth et al., 2019; Salihu, Eke, Ibrahim, & Kusharki, 2023). The comparative analysis based on code coverage tools for Android applications is summarized in the paragraph that follows.

Horvath et al., (Horváth et al., 2019) explores issues in measuring code coverage for Java and compares results from tools using two types of instrumentation: source code and bytecode. It found that, due to key differences between the methods, source code instrumentation is better suited for calculating branch coverage. Moreover, the (Q. Yang, Li, & Weiss, 2006) surveys and compare 17 coverage-based testing instruments with an emphasis on coverage measurement, but not exclusively. They also surveyed other capabilities, such as test report customization, automatic test case development, debugging support, and software prioritization for testing. Furthermore, a survey of five code coverage tools is presented by (Shelke & Nagpure, 2014), and one of them was truly assessed for the features it suggested. Based on the established criteria, a comparative analysis is offered. The tools are compared based on the following criteria: supported languages, number of coverage criteria, instrumentation, and automation.

Upon closer examination, many of these improvements indeed introduce novel comparative study on based on android applications code coverage tools. However, authors often focus source code and bytecode, test report customization, automatic test case development, debugging support, and software prioritization, supported languages, number of coverage criteria, instrumentation, and automation underlying these domains. Consequently, they rarely connect this knowledge to our broader understanding of android applications code coverage tools. However, none of them focus on the evaluation in instrumentation strategies, integration capabilities, validation methods, and reporting metrics. As a result, the current paper will conduct a comparative study on android applications code coverage tools focusing the evaluation in instrumentation strategies, integration capabilities, validation methods, and reporting metrics to propose a future direction in the domain.

The paper is structured as follows: Section 2 outlines the methodology of the research, followed by Section 3 detailing the comparative analysis. Next, in Section 4, presents the results and discussions of the study. Finally, Section 5 offers conclusion.

MATERIALS AND METHODS

The approach employed to carry out the study is discussed in this section. The majority of recent software research publications have focused on Android apps; hence, this study focusses on code coverage tools for Android apps. We conducted a mini-literature survey on papers published in the last ten years on the well-known research databases and indexing systems of Google Scholar and Scopus in order to choose our subjects. Because Scopus and Google Scholar include all major publishers, including Elsevier, Springer, ACM, and IEEE publications, these two were deemed suitable. We found the most widely used mutation testing methods and resources by conducting a search using the paper title, keywords, and abstract from 2013 to the present.

Comparative Criteria

Code coverage shows how much of your code is tested. In Android development, you can create test coverage reports locally using tools like Emma, JaCoCo, and Cobertura (Dashevskyi et al., 2018). The process of generating and sending code coverage to any code coverage storage platform can be done automatically. The criteria used for comparison are: Average time, testing process, integration with testing tool, open source, is the tool validated? metrics measured by the tool. The criteria used for comparison are:

1. Instrumentation time

Coverage-testing tools use program execution monitoring to get coverage data. By adding probes to the program either before or during execution, execution can be tracked. Usually, a probe consists of a few lines of code that, when run, provide an event or record indicating that the program has completed its execution at the probe's location.

2. Testing strategy

Various methods are employed for measuring coverage when testing Android applications. In both methods, the system being tested and/or the runtime engine are instrumented, which means that measurement probes are positioned inside the system at particular points to allow for the gathering of runtime data without changing the system's behavior (Muazu, Hashim, Audi, & Maiwada, 2024). The first strategy is to instrument the source code, which entails changing the original code by adding probes, after which it is constructed and run through testing (Usman, et. al., 2023). The second approach involves instrumenting the system's compiled version, or bytecode (Salihu, Ibrahim, & Usman, 2018). There are two more methods here. The first option is to insert the probes immediately following the build, which essentially creates altered bytecode files. Second, when a class is loaded for execution, the instrumentation may happen at runtime.

Source code instrumentation allows precise control over what parts are instrumented, while bytecode instrumentation typically instruments entire classes at once. Compile time won't be impacted by online bytecode instrumentation, but runtime overhead comprises both the additional code execution time and instrumentation charges, which are typically incurred once every class load. Lastly, source code instrumentation results are directly linked to the sections of the source code, but bytecode-based results can occasionally be challenging to trace back to the source code.

3. Integration with testing tool

Code coverage tools help measure how much of your code is being tested by your test suite. Integration with testing tools ensures that you can see which parts of your code are covered during testing and identify untested code paths. Some of them relies on other tools to ensure their long-term viability and compatibility with newer app. For instance, AndroLog runs on Soot, BBoxTester is based on Emma.

4. License

A wide range of tools, many of which are open source and free, have emerged as a result of the growing need for code coverage assessment in mobile apps, where continuous integration necessitates ongoing code quality monitoring and regression testing.

5. Is the tool validated?

In software engineering, validation is performed to determine whether a system or method satisfies requirements and achieves its goals. Every study uses case studies, controlled or quasi-controlled trials, or comparisons with other methods to evaluate the precision or effectiveness of a new strategy or technique.

6. Metrics used in measuring the Tools

In order to quantify and analyze test coverage, numerous research articles concentrated on various coverage components. Statement, branch, block, decision, condition, method, class, package, requirement, and data flow coverage are some of the twelve different types of coverage items. It is evident from the collected papers that while other scholars have focused on various forms of coverage, only two have employed requirement coverage for test coverage analysis.

COMPARATIVE ANALYSIS

Our study focuses on comparing code coverage tools for the Android mobile apps that available mainstream research databases. This section presents an overview of the thirteen techniques/tools selected for the study and discusses the result. Summary of the

Emma ("EMMA: a free Java code coverage tool.," 2025) is a Java-based code coverage tool capable of instrumenting classes to measure various coverage types, including class, method, line, and basic block coverage. However, its limitation lies in supporting only Java archive (JAR) file formats, as it does not accommodate the Dalvik executable (DEX) format.

Jacoco ("Jacoco," 2025), is a free code coverage library for Java, which has been created by the EclEmma team from using and integrating existing libraries for many years. Jacoco measure instruction, branch, line, and method coverage.

AndroLog (Samhi & Zeller, 2024) is an innovative tool built on the Soot framework, aimed at delivering detailed coverage insights across various levels, such as classes, methods, statements, and Android components. Unlike some other tools, AndroLog places the responsibility of testing apps on the analysts and emphasizes simplicity as its core principle.

COSMO (Romdhana, Ceccato, Georgiu, Merlo, & Tonella, 2021) is a fully automated Android app instrumentation tool that works transparently at the source code level. It is publicly available and fully compatible with existing system-level testing tools and Android test generators. Experimental results demonstrate that COSMO can instrument the majority of apps effectively without affecting their execution behavior, while only introducing a minimal and acceptable runtime overhead.

WallMauer (Auer et al., 2024), a new code coverage tool that supports multidex, and avoids inconsistencies by rigorously instrumenting Dalvik byte-code directly. WallMauer solely requires an APK file as input and as

such it can be easily integrated into any existing testing environment. The code coverage report of WallMauer is textual, but the authors are planning to provide a graphical report, to help human testers understand it better.

ACVTool (Pilgun, Gadyatskaya, Dashevskyi, Zhauniarovich, & Kushniarou, 2018) can instrument Android APK files and generate a code coverage report, without needing access to the APK's original source code. The six phases that make up ACVTool's workflow for a single APK are Instrument, Install, Start, Test, Stop, and Report. Additionally, the program gathers crash data that make it easier to analyse software flaws. It takes 36 seconds on average to instrument an app with ACVTool, which is negligible for routine testing and analysis.

InsDal (Liu, Wu, Deng, Yan, & Zhang, 2017) is a tool designed to insert instructions at specific locations within Dalvik bytecode based on user-defined requirements. It optimizes the inserted code to prevent memory waste and needless overhead, and it carefully controls the registers to guard against unauthorized alteration of the original code's behavior. This user-friendly tool has been used in a variety of situations, including code coverage analysis and energy analysis. InsDal, built on top of ApkTool, operates at the smali level, offering only instrumentation at the class and method levels (Liu et al., 2017).

Cobertura ("Cobetura," 2025) is a free Java-based tool designed to measure the percentage of code exercised by tests. It helps pinpoint areas of a Java program that lack sufficient test coverage and is built upon the jcoverage framework.

CovDroid (Yeh & Huang, 2015), a black-box coverage system for android. Furthermore, an application with various test cases is used to demonstrate the concept that the coverage index can enhance the app's performance and serve as a metric for evaluating the quality of test cases, whether for app marketplaces or testing service providers. CovDroid executes its instrumentation at the smali level, inserting probes at the method level. The method devised by Huang et al. also functions at the smali level but necessitates additional modifications to the Android manifest, including integrating new permissions. The tool demonstrates a limited success rate in instrumenting apps, achieving only 36%.

BBoxTester, (Zhauniarovich, Philippov, Gadyatskaya, Crispo, & Massacci, 2015), is a black-box code coverage tool for Android apps. It converts Dalvik bytecode into Java bytecode using dex2jar, and then leverages Emma ("EMMA: a free Java code coverage tool.," 2025) for the instrumentation process. However, BBoxTester's approach necessitates modifying the Android manifest and adding app resources, making it less flexible and potentially intrusive.

Huang et al. (Huang, Chiu, Lin, & Tzeng, 2015) propose a general approach to measure the code coverage rate of dynamic analysis tools for Android that can be used with both online and offline implementations.

MALintent (Askar, Fleischer, Kruegel, Vigna, & Kim) is an open-source fuzzing framework that uses novel bug oracles to find security vulnerabilities in AndroidIntent handlers. MALintent is the first Intentfuzzer that applies grey box fuzzing on compiled closed source Android applications. The methods that work well with a variety of Android versions is presented, and bug oracles were able to identify a number of crashes, privacy-related vulnerabilities, and memory-safety problems in the most popular Android app, Google Play store.

ELLA (Anand, 2016), is a tool that allows Android APKs to be instrumented for a variety of uses, including recording which methods are used. Along with both online and offline tools, it can also capture the time-stamped trace of methods that have been executed, the values of arguments given at call sites, the values of formal parameters of methods, etc.

ELLA (Anand, 2016) and InsDal (Liu et al., 2017) measure code coverage only at the method level.

RESULTS AND DISCUSSION

In this section we and discusses results of the comparative analysis to provide a better understanding of the collected data. Table 1 shows the summary of the tools compared in the study.

Table 1 Summary of Reviewed Tools

Tool	Time	operation	Testing	Integration	License	Validation	Metrics	Report
Emma		Bytecode	Black box	--	yes	-	Class, method, line, block	Xml, text, html
Jacoco	40 sec	Bytecode	Black box	Soot	yes	yes	classes, methods, statements	Xml, html
AndroLog	34 sec	Bytecode	Black box	Soot	yes	yes	class, methods, statements,	-
COSMO	1 sec	Source code	Gray box	jacoco	yes	yes	methods and lines of code	HTML, CSV, XML
WallMauer	21 sec	Bytecode	Black box	Mate	yes	yes	Line, class, method	text
ACVTool	36 sec	Smali	Black box	Apktool	yes	Yes	Instruction, line, method, crash reports,	Html, Xml
InsDal	-	Smali	-	Apktool	yes	yes	Method, class	--
Cobertura		Bytecode	Whitebox	jcoverage		yes	Line, branch, method	Html, xml
CovDroid	-	Smali	Black box	-	yes	yes	method	jasmin structure
ELLA	-	Bytecode	Black box	-	yes	yes	Method	-
BBoxTester	13 sec	Bytecode		emma	yes	yes	Block, class, method	-
MALintent	7 sec	Bytecode	Gray box	eJavaVirtualMachine ToolsInterface(JVMTI)	yes	-	Block class	-
Huang et al.	-	smali	Black box	emma	yes	yes	Class Method Block, Line	--

1. Operation of the Tools

Most Android app coverage tools function at the bytecode level and do not inform developers about which source code lines remain untested, as illustrated in Figure 1. In cases where the source code is unavailable, coverage can be evaluated by instrumenting the app's Smali bytecode. However, for developers, identifying which source code lines are covered is typically more valuable than simply knowing the percentage of covered Smali code. Actually, a certain level of coverage of the source code does not always follow from a given level of coverage of the Smali code.

Furthermore, developers find it far more challenging to comprehend how to cover the Smali code sections that are not yet covered by the existing test suite than it is to reason directly on the exposed source code.

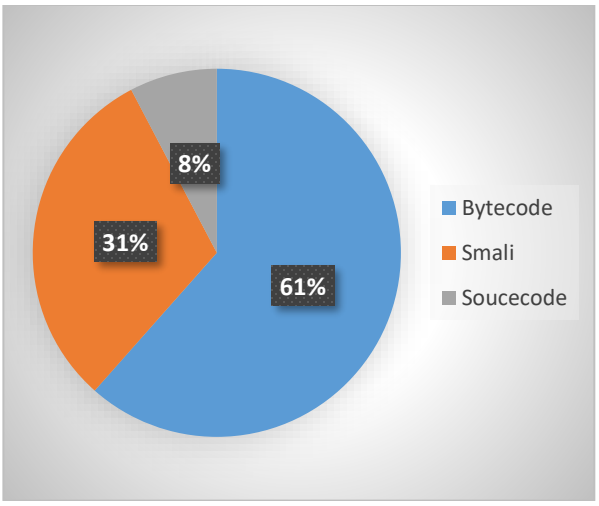

Figure 1: Operations of the Tools

2. Instrumentation Time

In terms of effectiveness, we can observe that the time required for app instrumentation is little in comparison to the time spent on app testing. Additionally, only one instrumentation of the apps is required for each testing session. Figure 2 show the average instrumentation time for most of the tools. Consequently, efficiency does not depend on this offline processing. BBOXTESTER (Zhauniarovich et al., 2015) was able to instrument 45 out of 52 applications of Dynodroid in 612 seconds, which is roughly 13 seconds.

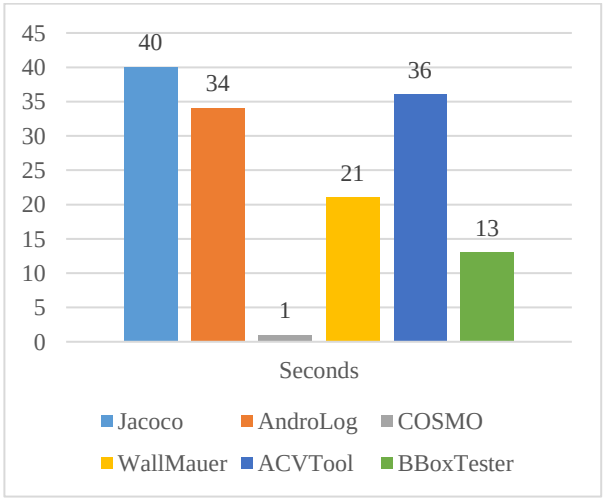

Figure 2: Instrumentation Time

3. Testing Strategy

Android app coverage can be obtained through two distinct types of instrumentation: black-box and white-box. Black-box instrumentation operates on the app's bytecode, whereas white-box instrumentation is applied directly to its source code. There are numerous black-box tools for measuring code coverage such as (Auer et al., 2024; Pilgun et al., 2018; Romdhana et al., 2021; Samhi & Zeller, 2024). However, they are unable to measure fine-grained source code coverage, as the coverage is computed at the bytecode level and cannot be accurately mapped back to specific source code lines, reducing its clarity for developers. In contrast, for general Java projects, source code coverage can be assessed using well-established methods. Tools for white-box code coverage ("Cobetura," 2025) measurement are included and maintained by Google in the Android SDK. Figure 3 show the percentage of testing strategy.

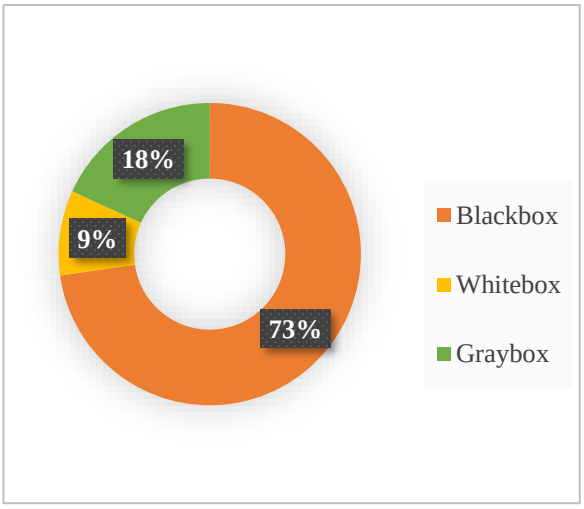

Figure 3: Testing Strategy

4. Evaluation Metric

A big part of software engineering is evaluation. It is the methodical approach to figuring out how well a process, method, or technique will work in the end. It is carried out in compliance with recognized metrics or measurements, such as code coverage in the software testing domain. Line, class, branch, method, instruction, and block are the most commonly used coverage evaluation criteria for determining how effective they are.

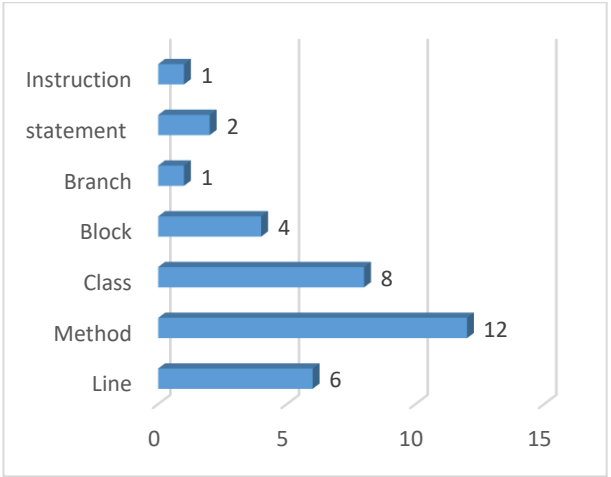

Figure 4: Evaluation Metric

5. Code Coverage Report

Most of the tools obtain the coverage report that can independently be used for any other downstream task in

different format. The most widely used format used for reporting coverage is xml and html. This is because they are easy to read and understand by both humans and machines, promoting transparency and interoperability. They are also not tense to any precise software or hardware, making it widely operational in different systems. The HTML report usually shows the application code in Smali with the relevant coverage details in an intuitive browser view. The same code coverage data that can be incorporated into automated testing tools is included in the report's xml and cvs versions. Figure 5 show the most widely used formats. And most of the tools use morethan one format like (Romdhana et al., 2021) use html, csv, xml, while ("EMMA: a free Java code coverage tool.," 2025) use html, text, and xml

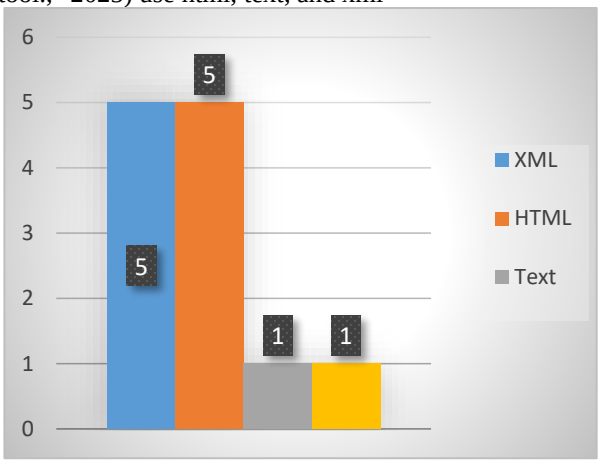

Figure 5: Code Coverage Report

6. Validation

All authors validated their tools using comparison with other existing techniques was also carried out. For instance AndroLog (Samhi & Zeller, 2024) is compared against existing tools COSMO (Romdhana et al., 2021), ACVTool (Pilgun et al., 2018), and BBoxTester (Zhauniarovich et al., 2015).

CONCLUSION

This comparative study of Android code coverage tools underscores the diverse approaches and capabilities available for measuring software test effectiveness in mobile applications. Tools like COSMO and WallMauer demonstrate advanced instrumentation methods, offering fine-grained analysis with minimal overhead, while others such as Emma and BBoxTester maintain simplicity and legacy compatibility. Our analysis reveals that although bytecode instrumentation dominates the Android testing landscape, source code-based tools provide more actionable insights for developers. The inclusion of metrics like method, instruction, and crash report coverage further distinguishes tools in terms of their practical applicability and diagnostic power. Additionally, the validation of tools through case studies and performance benchmarks ensures their credibility for real-world usage. Ultimately, the selection of an appropriate tool depends on factors such as required coverage granularity, integration needs, and available code access. This study offers a foundational reference to inform the choice of tools that best align with specific testing goals in Android application development.

REFERENCE

Anand, S. (2016). ELLA: a tool for binary instrumentation of Android apps. In: May.

Askar, A., Fleischer, F., Kruegel, C., Vigna, G., & Kim, T. (2025). MALintent: Coverage Guided Intent Fuzzing Framework for Android. In 32nd Annual Network and Distributed System Security Symposium, NDSS (pp. 24-28).

Auer, M., Arcuschin Moreno, I., & Fraser, G. (2024, April). Wallmauer: Robust code coverage instrumentation for android apps. In Proceedings of the 5th ACM/IEEE International Conference on Automation of Software Test (AST 2024) (pp. 34-44). https://doi.org/10.1145/3644032.3644462

Azim, T., & Neamtiu, I. (2013, October). Targeted and depth-first exploration for systematic testing of android apps. In Proceedings of the 2013 ACM SIGPLAN international conference on Object oriented programming systems languages & applications (pp. 641-660). https://doi.org/10.1145/2509136.2509549

Cobetura. (2025). Retrieved from https://sourceforge.net/projects/cobertura/

Dashevskyi, S., Gadyatskaya, O., Pilgun, A., & Zhauniarovich, Y. (2018, October). The influence of code coverage metrics on automated testing efficiency in android. In Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security (pp. 2216-2218). https://doi.org/10.1145/3243734.3278524

Eke, N. O., Salihu, I. A., & Usman, A. (2023, November). Comparative Analysis of Fully Automated Testing Techniques for Android Applications. In 2023 2nd International Conference on Multidisciplinary Engineering and Applied Science (ICMEAS) (pp. 1-6). IEEE. DOI: 10.1109/ICMEAS58693.2023.10429901

EMMA: a free Java code coverage tool. (2025). Retrieved from https://emma.sourceforge.net/Horváth, F., Gergely, T., Beszédes, Á., Tengeri, D., Balogh, G., & Gyimóthy, T. (2019). Code coverage differences of Java bytecode and

source code instrumentation tools. Software Quality Journal, 27, 79-123.

Huang, C. Y., Chiu, C. H., Lin, C. H., & Tzeng, H. W. (2015, June). Code coverage measurement for Android dynamic analysis tools. In 2015 IEEE International Conference on Mobile Services (pp. 209-216). IEEE. DOI: 10.1109/MobServ.2015.38

Jacoco. (2025). Retrieved from https://www.eclemma.org/jacoco/

Liu, J., Wu, T., Deng, X., Yan, J., & Zhang, J. (2017, February). InsDal: A safe and extensible instrumentation tool on Dalvik byte-code for Android applications. In 2017 IEEE 24th international conference on software analysis, evolution and reengineering (SANER) (pp. 502-506). IEEE. DOI: 10.1109/SANER.2017.7884662

Memon, A. M., Soffa, M. L., & Pollack, M. E. (2001, September). Coverage criteria for GUI testing. In Proceedings of the 8th European software engineering conference held jointly with 9th ACM SIGSOFT international symposium on Foundations of software engineering (pp. 256-267). https://doi.org/10.1145/503209.503244

Muazu, A. A., Hashim, A. S., Audi, U. I. I., & Maiwada, U. D. (2024). Refining a one-parameter-at-a-time approach using harmony search for optimizing test suite size in combinatorial t-way testing. IEEE Access. DOI: 10.1109/ACCESS.2024.3463953

Pathy, S., Panda, S., & Baboo, S. A. R. A. D. A. (2015). A review on code coverage analysis. International Journal of Computer Science & Engineering Technology (IJCSET), 6(10), 580-587.

Pilgun, A., Gadyatskaya, O., Dashevskyi, S., Zhauniarovich, Y., & Kushniarou, A. (2018, October). An effective android code coverage tool. In Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security (pp. 2189-2191). https://doi.org/10.1145/3243734.3278484

Pilgun, A., Gadyatskaya, O., Zhauniarovich, Y., Dashevskyi, S., Kushniarou, A., & Mauw, S. (2020). Fine-grained code coverage measurement in automated black-box android testing. ACM Transactions on Software Engineering and Methodology (TOSEM), 29(4), 1-35. https://doi.org/10.1145/3395042

Romdhana, A., Ceccato, M., Georgiu, G. C., Merlo, A., & Tonella, P. (2021, April). Cosmo: Code coverage made easier for android. In 2021 14th IEEE conference on software testing, verification and validation (ICST) (pp. 417-423). IEEE. DOI: 10.1109/ICST49551.2021.00053

Salihu, I. A., Usman, A. U., Eke, N. O., Ibrahim, R., & Kusharki, M. B. (2023, November). Mutation Testing Techniques for Android Applications: A Comparative Study. In 2023 2nd International Conference on Multidisciplinary Engineering and Applied Science (ICMEAS) (Vol. 1, pp. 1-5). IEEE. DOI: 10.1109/ICMEAS58693.2023.10429866

Salihu, I. A., & Ibrahim, R. (2016, November). Systematic exploration of android apps' events for automated testing. In Proceedings of the 14th International Conference on Advances in Mobile Computing and Multi Media (pp. 50-54). https://doi.org/10.1145/3007120.3011072

Samhi, J., & Zeller, A. (2024, July). AndroLog: Android Instrumentation and Code Coverage Analysis. In Companion Proceedings of the 32nd ACM International Conference on the Foundations of Software Engineering (pp. 597-601). https://doi.org/10.1145/3663529.3663806

Salihu, I. A., Ibrahim, R., & Usman, A. (2018, August). A Static-dynamic Approach for UI Model Generation for Mobile Applications. In 2018 7th International Conference on Reliability, Infocom Technologies and Optimization (Trends and Future Directions)(ICRITO) (pp. 96-100). IEEE. DOI: 10.1109/ICRITO.2018.8748410

Shahid, M., Ibrahim, S., & Mahrin, M. N. R. (2011). A study on test coverage in software testing. Advanced Informatics School (AIS), Universiti Teknologi Malaysia, International Campus, Jalan Semarak, Kuala Lumpur, Malaysia, 1.

Shelke, S., & Nagpure, S. (2014). The Study of various code coverage tools. International Journal of Computer Trends and Technology (IJCTT), 13(1).

Usman, A., Boukar, M. M., Suleiman, M. A., & Salihu, I. A. (2024). Test Case Generation Approach for Android Applications using Reinforcement Learning. Engineering, Technology & Applied Science Research, 14(4), 15127-15132. https://doi.org/10.48084/etasr.7422.

Usman, A., Ibrahim, N., & Salihu, I. A. (2018, February). Test case generation from android mobile applications focusing on context events. In Proceedings of the 2018 7th international conference on software and computer applications (pp. 25-30). https://doi.org/10.1145/3185089.3185099

Usman, A., Ibrahim, N., & Salihu, I. A. (2020). TEGDroid: Test case generation approach for android apps considering context and GUI events. International Journal on Advanced Science, Engineering and Information Technology, 10(1), 16.

Usman, A., Ibrahim, R., Sulaiman, M. A., & Salihu, I. A. (2024). An in-depth analysis of machine learning based techniques for automated testing of android applications. International Journal of Communication Networks and Information Security, 16(3), 663-683.

Usman, A., Boukar, M. M., Suleiman, M. A., Salihu, I. A., & Eke, N. O. (2023). Reinforcement learning for testing android applications: A review. In 2023 2nd International Conference on Multidisciplinary Engineering and Applied Science (ICMEAS) (Vol. 1, pp. 1-6). IEEE.

Yang, Q., Li, J. J., & Weiss, D. (2006, May). A survey of coverage-based testing tools. In Proceedings of the 2006 international workshop on Automation of software test (pp. 99-103). https://doi.org/10.1145/1138929.1138949

Yang, S., Huang, S., & Hui, Z. (2019). Theoretical Analysis and Empirical Evaluation of Coverage Indictors for Closed Source APP Testing. IEEE Access, 7, 162323-162332.

Yeh, C. C., & Huang, S. K. (2015, July). Covdroid: A black-box testing coverage system for android. In 2015 IEEE 39th annual computer software and applications conference (Vol. 3, pp. 447-452). IEEE. DOI: 10.1109/COMPSAC.2015.125

Zhauniarovich, Y., Philippov, A., Gadyatskaya, O., Crispo, B., & Massacci, F. (2015, August). Towards black box testing of android apps. In 2015 10th International Conference on Availability, Reliability and Security (pp. 501-510). IEEE. DOI: 10.1109/ARES.2015.70