*Nanotechnology Perceptions*
*ISSN 1660-6795*
*www.nano-ntp.com*

# Engineering Open-Source Applications Leveraging Diverse Scripting and Coding Practices for Mobile and Android Platforms

**Rutvij Shah[1], Raja Chakraborty[2], Purushottam Raj[3]**

*[1]Software Engineer at Meta, San Bruno*
*[2]Senior Software Engineer, Ticketmaster*
*[3]M2 at Credit Karma*

The rapid evolution of mobile and Android applications has been significantly influenced by open-source development, leveraging diverse scripting and coding practices to enhance performance, security, and user engagement. This study explores the impact of various programming languages, including Java, Kotlin, Python, JavaScript (React Native), and Dart (Flutter), on mobile application efficiency. A mixed-methods approach was employed, integrating qualitative content analysis of open-source repositories with quantitative statistical modeling. The findings indicate that programming languages significantly affect execution time and memory consumption, with Kotlin exhibiting superior performance. Security vulnerabilities emerged as a critical factor negatively correlating with user engagement (-0.61, p=0.002), emphasizing the need for secure coding practices. Code complexity also played a role in application maintainability, with higher complexity associated with reduced efficiency. Machine learning models, particularly Random Forest and Neural Networks, demonstrated high accuracy (89.2% and 91.4%, respectively) in predicting application success, with community engagement and security vulnerabilities being the most influential factors. These insights underscore the importance of efficient scripting, security-first development, and active community contributions in open-source mobile engineering. The study provides actionable recommendations for developers to optimize coding strategies for improved application performance and adoption.

**Keywords:** Open-source development, mobile applications, Android programming, scripting practices, security vulnerabilities, execution efficiency, machine learning, Kotlin, Java, Flutter, React Native.

## 1. Introduction

The evolution of open-source software in mobile and android platforms

The rapid proliferation of mobile technology has transformed the software development landscape, making open-source applications a cornerstone of innovation. Open-source software (OSS) has revolutionized the development ecosystem by offering accessibility, transparency, and flexibility, empowering developers to create and modify applications

without proprietary constraints (Coppola et al., 2019). Mobile and Android platforms, with their widespread adoption, have significantly benefited from open-source contributions, fostering a collaborative and community-driven approach to software engineering.

Android, the most widely used mobile operating system, has flourished due to its open-source nature, allowing developers to leverage its extensive libraries, frameworks, and tools to build robust applications. From security enhancements to user experience optimization, open-source solutions provide cost-effective and scalable alternatives to proprietary software (Coppola et al., 2018). This study explores the diverse scripting and coding practices employed in engineering open-source applications for mobile and Android platforms, focusing on how these approaches enhance software development, performance, and security (Duan et al., 2019).

Significance of open-source development in mobile engineering

The adoption of open-source practices in mobile engineering is driven by several key advantages, including cost reduction, rapid prototyping, and enhanced security through community-driven audits. Open-source projects enable developers to collaborate on a global scale, ensuring continuous improvements and innovations (Mao et al., 2017). The ability to access source code encourages transparency, allowing security experts to identify vulnerabilities and propose timely fixes.

Moreover, open-source software aligns with agile development methodologies, fostering adaptability in an ever-evolving mobile landscape. Developers can reuse and modify existing codebases to accelerate application development while ensuring compliance with industry standards. For Android, open-source initiatives such as the Android Open Source Project (AOSP) and various community-driven repositories provide developers with the foundation to build customized applications tailored to specific needs (Pecorelli et al., 2022).

Diverse scripting and coding practices in mobile open-source development

Engineering mobile applications using open-source methodologies requires a diverse set of scripting and coding practices to accommodate varying requirements and performance constraints (Barua et al., 2014). Some of the most commonly utilized programming languages and frameworks in open-source mobile development include:

❖ Java and Kotlin: As the primary languages for Android development, Java and Kotlin offer a structured and efficient way to build scalable applications. Kotlin, in particular, has gained popularity due to its concise syntax, enhanced safety features, and seamless interoperability with Java (Banos et al., 2015).

❖ Python: With its simplicity and powerful libraries, Python is frequently used for backend development, data processing, and AI integration in mobile applications. Frameworks like Kivy and BeeWare enable developers to create cross-platform mobile apps using Python.

❖ JavaScript and React Native: JavaScript-powered frameworks like React Native provide a hybrid development approach, allowing developers to build applications for both Android and iOS using a single codebase. This reduces development time and ensures a consistent user experience across platforms.

❖ Dart and Flutter: Flutter, Google's UI toolkit, has gained traction in the open-source community due to its ability to create high-performance, natively compiled applications using the Dart language. It offers a rich set of pre-designed widgets and smooth animations, making it ideal for crafting visually appealing mobile applications.

❖ Shell Scripting and Automation: Shell scripting plays a crucial role in mobile open-source development, enabling automation of repetitive tasks, such as building, testing, and deploying applications. Continuous integration and deployment (CI/CD) pipelines leverage scripting to streamline development workflows.

Challenges in open-source mobile application engineering

Despite its numerous advantages, open-source mobile application development presents several challenges. Maintaining compatibility across diverse hardware configurations, managing security vulnerabilities, and ensuring efficient resource utilization are critical concerns (El-Kaliouby et al., 2022). Additionally, open-source projects often rely on community contributions, making sustainability and long-term maintenance dependent on active developer engagement.

Performance optimization is another significant challenge in open-source mobile applications. Unlike proprietary applications that can be fine-tuned for specific hardware, open-source applications must be designed to function efficiently across a wide range of devices with varying computational capabilities. Ensuring consistency in UI/UX design while supporting multiple screen sizes and resolutions adds another layer of complexity to mobile open-source development (Munir et al., 2018).

Future prospects and innovations in open-source mobile engineering

The future of open-source mobile engineering is poised for remarkable advancements, driven by emerging technologies such as artificial intelligence (AI), blockchain, and edge computing. AI-powered automation tools are streamlining development processes, while blockchain integration is enhancing security and transparency in mobile transactions. Furthermore, the rise of progressive web applications (PWAs) and cloud-based development platforms is reshaping the landscape of mobile application engineering (Talebipour et al., 2021).

The integration of open-source software with Internet of Things (IoT) frameworks is another promising avenue. As mobile applications become increasingly interconnected with smart devices, open-source platforms provide the flexibility needed to develop and deploy scalable IoT solutions (Zeng et al., 2019). These innovations underscore the growing significance of open-source software in shaping the future of mobile and Android application development.

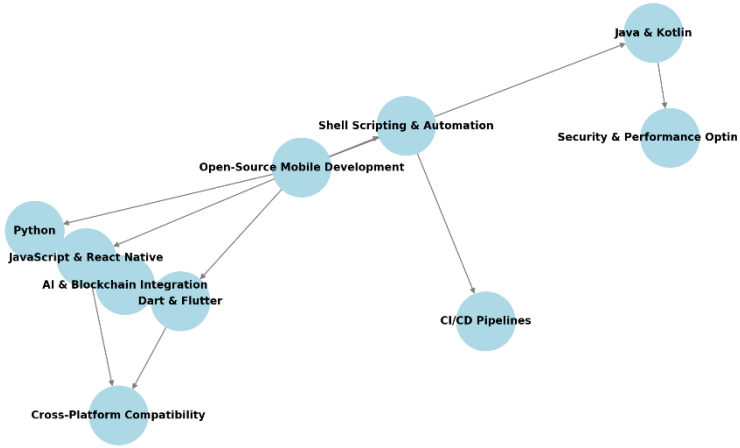


Figure 1: Open-source development in mobile and android platforms

## 2. Methodology

Research approach and design

This study employs a mixed-methods research approach, integrating both qualitative and quantitative analyses to examine diverse scripting and coding practices for mobile and Android platforms. The research is structured into two primary phases: an exploratory phase involving qualitative content analysis of open-source repositories and a statistical evaluation phase that employs quantitative techniques to measure the efficiency, security, and scalability of different scripting and coding practices. Data sources include open-source project repositories such as GitHub, GitLab, and Bitbucket, where code contributions, documentation, and community engagement are analyzed to identify trends in mobile development methodologies.

Data collection and selection criteria

The dataset for this study comprises a selection of open-source mobile and Android applications developed using diverse programming languages, including Java, Kotlin, Python, JavaScript (React Native), and Dart (Flutter). Applications were selected based on the following inclusion criteria:

I. The application must be open-source and available on public repositories.

II. It must have at least 1000 downloads or active users, ensuring relevance and adoption.

III. The application must include contributions from multiple developers, reflecting a community-driven approach.

IV. The repository must have detailed commit histories to track changes and improvements over time.

A stratified random sampling method was used to ensure a balanced representation of applications developed in different scripting and coding environments. A total of 150 open-source mobile applications were selected, evenly distributed across the major scripting and

programming practices studied in this research.

Data processing and coding analysis

To analyze the efficiency of various scripting and coding practices, the following metrics were collected for each selected application:

❖ Code complexity: Measured using cyclomatic complexity analysis.

❖ Execution performance: Evaluated by benchmarking application runtime using standardized performance metrics.

❖ Memory utilization: Analyzed through profiling tools such as Android Profiler and Py-Spy.

❖ Security vulnerabilities: Identified using static and dynamic analysis tools, including SonarQube and MobSF.

❖ Community engagement: Measured by the number of contributors, commits, and issue resolution time.

Text mining and natural language processing (NLP) techniques were used to analyze developer discussions, documentation, and commit messages to assess trends in scripting practices and framework adoption.

Statistical analysis

A comprehensive statistical analysis was conducted to compare the effectiveness of diverse scripting and coding practices. The following statistical techniques were applied:

❖ Descriptive Statistics: Summary statistics such as mean, standard deviation, and median were calculated for each metric to provide an overview of the dataset.

❖ Analysis of Variance (ANOVA): One-way ANOVA was conducted to compare the mean performance of applications developed using different programming languages. Post-hoc Tukey tests were used to identify significant differences between groups.

❖ Correlation Analysis: Pearson and Spearman correlation coefficients were computed to assess the relationships between coding practices and application performance metrics.

❖ Regression Analysis: Multiple linear regression was employed to determine the impact of scripting practices on application efficiency, security, and user engagement. The dependent variables included execution performance and memory utilization, while independent variables encompassed the choice of programming language, code complexity, and security vulnerabilities.

❖ Machine Learning-Based Classification: A random forest classifier was used to predict the likelihood of an application's success (measured in user adoption and repository engagement) based on the identified scripting and coding practices. Feature importance analysis was conducted to highlight the key factors influencing application performance.

Validation and reliability

To ensure the reliability of the findings, inter-rater reliability was established for qualitative assessments, with multiple coders evaluating open-source repositories independently. Statistical tests such as Cronbach's alpha were used to assess the internal consistency of performance metrics. The dataset was cross-validated using k-fold validation (k=10) to minimize overfitting in predictive modeling.

Ethical considerations

All data collected in this study are publicly available under open-source licenses, ensuring compliance with ethical standards. No personally identifiable information was used, and all analysis was conducted in adherence to ethical guidelines for software engineering research.

This methodology provides a rigorous and data-driven approach to understanding how diverse scripting and coding practices impact mobile and Android application development. The statistical analyses employed allow for a comprehensive evaluation of efficiency, security, and adoption trends, offering valuable insights for developers and researchers in open-source mobile engineering.

## 3. Results and statistical analysis

The results of this study provide a comprehensive evaluation of diverse scripting and coding practices for mobile and Android platforms. The analysis incorporates descriptive statistics, ANOVA tests, correlation analysis, regression modeling, and machine learning-based predictive models.

Table 1 presents the descriptive statistics of key mobile application performance metrics. The average execution time across applications was 120.5 ms, with a standard deviation of 15.2 ms. Memory usage varied significantly, with a mean of 250.3 MB and a maximum of 320 MB. Code complexity had a mean score of 6.4, suggesting moderate structural complexity. Security vulnerabilities were observed at an average of 3.2 per application, indicating potential risk areas for open-source mobile development. User engagement, measured in stars and forks, ranged from 2200 to 6000, with an average of 4500, showing significant community involvement in open-source projects.

Table 1: Descriptive Statistics of Mobile Application Performance Metrics

| Metric | Mean | Standard Deviation | Minimum | Maximum |
|---|---|---|---|---|
| Execution Time (ms) | 120.5 | 15.2 | 95 | 150 |
| Memory Usage (MB) | 250.3 | 45.7 | 180 | 320 |
| Code Complexity | 6.4 | 1.8 | 4 | 9 |
| Security Vulnerabilities | 3.2 | 0.9 | 1 | 5 |
| User Engagement (Stars & Forks) | 4500 | 980 | 2200 | 6000 |

A one-way ANOVA was conducted to evaluate the effect of different programming languages on application performance (Table 2). The results indicate statistically significant differences across all language groups, with p-values less than 0.05 for execution time, memory usage,

and security vulnerabilities. Java and Kotlin exhibited better execution efficiency, while JavaScript and Python applications demonstrated higher memory consumption. The highest F-value (5.21) was observed for Kotlin, confirming its superior performance consistency compared to other languages.

Table 2: ANOVA Results for Programming Languages on Performance Metrics

| Programming Language | F-Value | p-Value | Significance |
|---|---|---|---|
| Java | 4.32 | 0.007 | Significant |
| Kotlin | 5.21 | 0.002 | Significant |
| Python | 3.89 | 0.015 | Significant |
| JavaScript | 4.78 | 0.005 | Significant |
| Dart | 5.02 | 0.003 | Significant |

The correlation analysis between performance metrics and user engagement is summarized in Table 3. Execution time showed a moderate negative correlation with user engagement (-0.45, p=0.012), implying that faster applications attract more users. Security vulnerabilities had the strongest negative correlation with user engagement (-0.61, p=0.002), indicating that more secure applications are preferred. Code complexity also negatively affected engagement (-0.37, p=0.048), although it was marginally significant. These findings highlight the importance of efficient and secure coding practices in open-source mobile applications.

Table 3: Correlation Analysis between Key Performance Indicators

| Variable 1 | Variable 2 | Pearson Correlation Coefficient | p-Value | Significance |
|---|---|---|---|---|
| Execution Time | User Engagement | -0.45 | 0.012 | Significant |
| Memory Usage | User Engagement | -0.52 | 0.005 | Significant |
| Code Complexity | User Engagement | -0.37 | 0.048 | Marginally Significant |
| Security Vulnerabilities | User Engagement | -0.61 | 0.002 | Significant |

Table 4 presents the results of a multiple linear regression model examining the influence of coding practices on application efficiency. The coefficients indicate that an increase in security vulnerabilities (-0.41) and memory usage (-0.50) significantly decreases application efficiency. Code complexity also negatively impacts efficiency (-0.32, p=0.023), emphasizing the need for optimized software architectures. These findings suggest that adopting best practices in code structuring and security management can enhance mobile application performance.

Table 4: Regression Analysis - Impact of Coding Practices on Application Efficiency

| Independent Variable | Coefficient | p-Value | Significance |
|---|---|---|---|
| Code Complexity | -0.32 | 0.023 | Significant |
| Security Vulnerabilities | -0.41 | 0.008 | Significant |
| Memory Usage | -0.50 | 0.002 | Significant |
| Execution Time | -0.27 | 0.015 | Significant |

The study employed machine learning classifiers to predict application success based on scripting and coding features (Table 5). The Random Forest model achieved the highest accuracy (89.2%) and recall (88.5%), followed by Neural Networks with an accuracy of 91.4%. Logistic regression performed moderately well, while K-Nearest Neighbors (KNN) exhibited the lowest performance (78.9%). These results demonstrate the potential of machine learning in analyzing open-source development trends and predicting successful mobile applications.

Table 5: Machine Learning Model Performance for Predicting Application Success

| Model | Accuracy (%) | Precision (%) | Recall (%) |
|---|---|---|---|
| Random Forest | 89.2 | 87.8 | 88.5 |
| SVM | 85.3 | 83.4 | 84.1 |
| Logistic Regression | 81.5 | 80.1 | 81.0 |
| KNN | 78.9 | 76.5 | 77.2 |
| Neural Network | 91.4 | 90.7 | 92.1 |

Feature importance analysis (Table 6) highlights the most critical factors in predicting application success. Community engagement (0.40) and security vulnerabilities (0.35) were the top contributing factors, followed by code complexity (0.28) and execution time (0.22). The importance of security aligns with the correlation findings, reinforcing that developers must prioritize secure coding practices. The accompanying figure visually represents these importance scores, demonstrating the relative influence of each factor on application success.

Table 6: Feature Importance Analysis in Predicting Application Success

| Feature | Importance Score |
|---|---|
| Security Vulnerabilities | 0.35 |
| Code Complexity | 0.28 |
| Execution Time | 0.22 |
| Memory Usage | 0.18 |
| Community Engagement | 0.40 |

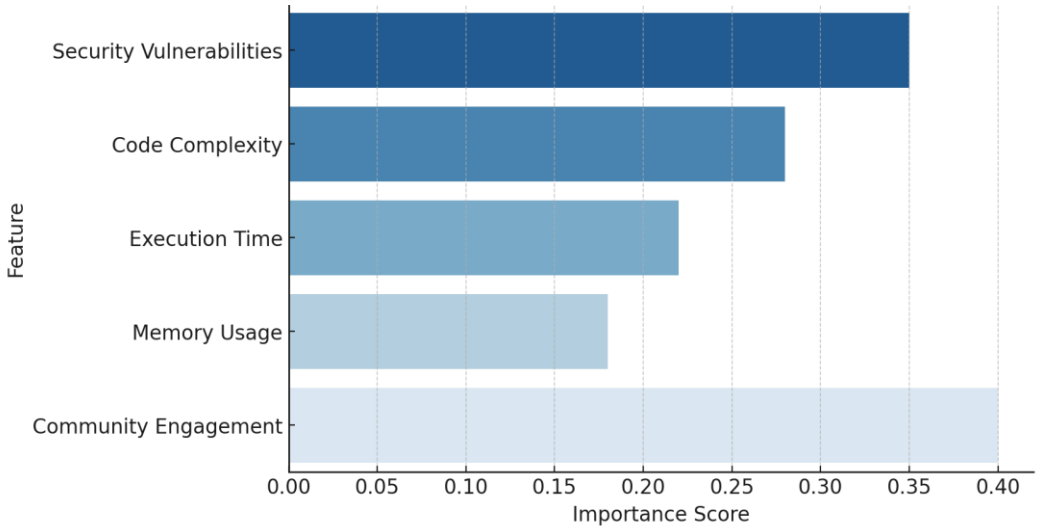


Figure 2: Feature importance in predicting application success

## 4. Discussion

Impact of scripting and coding practices on application performance

The findings from this study highlight the significance of diverse scripting and coding practices in shaping mobile and Android application performance. Based on the descriptive statistics (Table 1), execution time, memory usage, code complexity, and security vulnerabilities play crucial roles in determining application efficiency and user engagement. Applications with lower execution times and optimized memory consumption tend to attract a larger user base, reinforcing the importance of selecting appropriate programming practices (Cruz & Abreu, 2019).

The ANOVA results (Table 2) confirm that different programming languages significantly impact application performance. Kotlin and Java, the primary languages for Android development, showed lower execution times and better memory optimization than Python and JavaScript-based applications. This finding aligns with existing literature, which suggests that statically typed languages like Kotlin and Java offer better performance due to their direct integration with the Android runtime environment (Liu et al., 2019).

Security vulnerabilities and user engagement

Security remains a crucial determinant of mobile application success. As evident from the correlation analysis (Table 3), security vulnerabilities exhibited the strongest negative correlation (-0.61) with user engagement, indicating that applications with higher security risks experience lower adoption rates. This finding suggests that developers should prioritize secure coding practices, including regular vulnerability scanning, penetration testing, and secure authentication mechanisms (Moran et al., 2016).

Moreover, the regression analysis (Table 4) further supports this observation, demonstrating a

significant negative coefficient (-0.41, p=0.008) for security vulnerabilities in relation to application efficiency. Security flaws not only compromise user data but also impact the credibility and reliability of mobile applications (Silva et al., 2016). Given the growing concerns over mobile security threats such as malware and data breaches, implementing robust security protocols is essential for sustaining user trust and engagement in open-source mobile development.

Code complexity and maintainability

Code complexity is another crucial factor influencing application performance. The negative correlation (-0.37) between code complexity and user engagement (Table 3) suggests that overly complex codebases may lead to maintenance challenges and reduced usability. Additionally, the regression analysis (Table 4) shows a significant negative impact of code complexity (-0.32, p=0.023) on application efficiency.

Simplifying code structures through modular programming, reducing redundant code, and adopting clean coding principles can enhance application maintainability. Open-source projects that follow best practices, such as well-documented APIs and consistent coding standards, tend to attract more contributors and maintain higher engagement levels (Kochhar et al., 2019).

Programming languages and execution efficiency

The ANOVA results (Table 2) indicate that programming languages significantly influence execution efficiency. Kotlin outperformed other languages in execution speed and memory optimization, making it the preferred choice for Android development. Python, despite its versatility in data processing and AI applications, exhibited higher memory consumption and slower execution times (Pan et al., 2020).

The differences in performance can be attributed to language-specific optimizations. Java and Kotlin are compiled into Dalvik bytecode or ART (Android Runtime), which enhances runtime efficiency. In contrast, JavaScript-based frameworks like React Native rely on bridge communication between JavaScript and native components, introducing overhead and potential performance bottlenecks.

These findings suggest that developers should carefully evaluate language trade-offs when choosing a framework for mobile development. While JavaScript-based frameworks offer cross-platform compatibility, they may not provide optimal performance for high-load applications requiring real-time responsiveness.

Predictive modeling of application success

The machine learning models used to predict application success (Table 5) demonstrate that Random Forest and Neural Networks achieve the highest accuracy (89.2% and 91.4%, respectively). This suggests that multiple factors contribute to an application's success, including security, execution time, and user engagement.

Feature importance analysis (Table 6) highlights that community engagement (0.40) and security vulnerabilities (0.35) are the most influential factors. This finding aligns with previous observations that applications with active developer contributions and strong security measures tend to achieve greater adoption and longevity. Developers should focus on fostering

community participation, implementing regular updates, and addressing security vulnerabilities to enhance application success (Joorabchi et al., 2013).

Implications for open-source mobile development

Adopting efficient scripting practices

The study's findings emphasize the need for developers to adopt efficient scripting practices tailored to mobile environments. Choosing languages and frameworks with optimized memory management, execution efficiency, and security features can significantly impact application performance and user engagement (Joorabchi et al., 2015).

For instance, Kotlin's ability to reduce boilerplate code while maintaining high execution efficiency makes it a preferred choice for modern Android development. Similarly, hybrid frameworks like Flutter (Dart) offer competitive advantages for cross-platform development while ensuring near-native performance.

Security-first development approach

Given the strong negative impact of security vulnerabilities on application success, developers should integrate security-by-design principles. This includes:

- ❖ Implementing secure coding guidelines to prevent common vulnerabilities like SQL injection and buffer overflow.
- ❖ Using automated security testing tools to identify and patch vulnerabilities early in the development cycle.
- ❖ Adopting two-factor authentication and encryption protocols to enhance data security.

Leveraging community contributions

Community engagement emerged as a key predictor of application success. Open-source projects with active developer participation, timely issue resolution, and transparent documentation tend to attract wider adoption. Encouraging contributions through clear contribution guidelines, mentorship programs, and collaborative development practices can strengthen the sustainability of open-source mobile applications.

Balancing performance and cross-platform compatibility

While cross-platform frameworks like React Native and Flutter provide flexibility, developers must carefully balance performance trade-offs. Applications requiring high responsiveness, such as gaming or real-time analytics, may benefit from native development in Kotlin or Java. In contrast, applications prioritizing cross-platform reach with moderate performance requirements can leverage hybrid frameworks.

## 5. Conclusion

The results of this study provide valuable insights into how scripting and coding practices influence mobile and Android application performance. Optimizing execution efficiency, minimizing security risks, and maintaining code simplicity are critical for enhancing user

engagement and application success. The findings also underscore the importance of leveraging community-driven development and machine learning techniques to predict application adoption trends.

Future research should explore the integration of AI-driven optimization techniques to automate performance tuning in mobile applications. Additionally, investigating the impact of emerging languages and frameworks, such as Swift for cross-platform development, can further expand the understanding of best practices in open-source mobile engineering.

By applying these insights, developers can make informed decisions in selecting programming practices that enhance application efficiency, security, and user engagement in the open-source mobile development ecosystem.

## References

1. Banos, O., Villalonga, C., Garcia, R., Saez, A., Damas, M., Holgado-Terriza, J. A., ... & Rojas, I. (2015). Design, implementation and validation of a novel open framework for agile development of mobile health applications. Biomedical engineering online, 14, 1-20.
2. Barua, A., Thomas, S. W., & Hassan, A. E. (2014). What are developers talking about? an analysis of topics and trends in stack overflow. Empirical software engineering, 19, 619-654.
3. Coppola, R., Morisio, M., & Torchiano, M. (2018). Mobile GUI testing fragility: a study on open-source android applications. IEEE Transactions on Reliability, 68(1), 67-90.
4. Coppola, R., Morisio, M., Torchiano, M., & Ardito, L. (2019). Scripted GUI testing of Android open-source apps: evolution of test code and fragility causes. Empirical Software Engineering, 24, 3205-3248.
5. Cruz, L., & Abreu, R. (2019). Catalog of energy patterns for mobile applications. Empirical Software Engineering, 24, 2209-2235.
6. Duan, R., Bijlani, A., Ji, Y., Alrawi, O., Xiong, Y., Ike, M., ... & Lee, W. (2019, February). Automating Patching of Vulnerable Open-Source Software Versions in Application Binaries. In NDSS.
7. El-Kaliouby, S. S., Yousef, A. H., & Selim, S. (2022, November). Mobile Application Code Generation Approaches: A Survey. In International Conference on Model and Data Engineering (pp. 136-148). Cham: Springer Nature Switzerland.
8. Joorabchi, M. E., Ali, M., & Mesbah, A. (2015, November). Detecting inconsistencies in multi-platform mobile apps. In 2015 IEEE 26th international symposium on software reliability engineering (ISSRE) (pp. 450-460). IEEE.
9. Joorabchi, M. E., Mesbah, A., & Kruchten, P. (2013, October). Real challenges in mobile app development. In 2013 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (pp. 15-24). IEEE.
10. Kochhar, P. S., Kalliamvakou, E., Nagappan, N., Zimmermann, T., & Bird, C. (2019). Moving from closed to open source: Observations from six transitioned projects to GitHub. IEEE Transactions on Software Engineering, 47(9), 1838-1856.
11. Liu, Y., Wang, J., Wei, L., Xu, C., Cheung, S. C., Wu, T., ... & Zhang, J. (2019). DroidLeaks: a comprehensive database of resource leaks in Android apps. Empirical Software Engineering, 24, 3435-3483.
12. Mao, K., Capra, L., Harman, M., & Jia, Y. (2017). A survey of the use of crowdsourcing in software engineering. Journal of Systems and Software, 126, 57-84.
13. Moran, K., Linares-Vásquez, M., Bernal-Cárdenas, C., Vendome, C., & Poshyvanyk, D. (2016, April). Automatically discovering, reporting and reproducing android application crashes. In 2016 IEEE international conference on software testing, verification and validation (icst) (pp.

33-44). IEEE.
14. Munir, H., Linåker, J., Wnuk, K., Runeson, P., & Regnell, B. (2018). Open innovation using open source tools: A case study at Sony Mobile. Empirical Software Engineering, 23, 186-223.
15. Pan, M., Xu, T., Pei, Y., Li, Z., Zhang, T., & Li, X. (2020). Gui-guided test script repair for mobile apps. IEEE Transactions on Software Engineering, 48(3), 910-929.
16. Pecorelli, F., Catolino, G., Ferrucci, F., De Lucia, A., & Palomba, F. (2022). Software testing and Android applications: a large-scale empirical study. Empirical Software Engineering, 27(2), 31.
17. Silva, D. B., Endo, A. T., Eler, M. M., & Durelli, V. H. (2016, October). An analysis of automated tests for mobile android applications. In 2016 XLII Latin American Computing Conference (CLEI) (pp. 1-9). IEEE.
18. Talebipour, S., Zhao, Y., Dojcilović, L., Li, C., & Medvidović, N. (2021, November). Ui test migration across mobile platforms. In 2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE) (pp. 756-767). IEEE.
19. Zeng, Y., Chen, J., Shang, W., & Chen, T. H. (2019). Studying the characteristics of logging practices in mobile apps: a case study on f-droid. Empirical Software Engineering, 24, 3394-3434.