MobileTestPro: Testing Framework for Mobile Application

Jorge Romero-Collaguazo[1], Danilo Martinez[1,4], Mireya Zapata[2,3(✉)], and Xavier Ferre[5]

[1] Departamento de Ciencias de la Computación, Universidad de las Fuerzas Armadas ESPE, Av. Rumiñahui, Sangolquí, Ecuador
[2] Centro de Investigación en Mecatrónica y Sistemas Interactivos (MIST), Universidad Tecnológica Indoamérica, Quito, Ecuador
mireyazapata@uti.edu.ec
[3] Carrera de Ingeniería Industrial, Facultad de Ingenierías, Universidad Tecnológica Indoamérica, Quito, Ecuador
[4] Instituto Superior Tecnológico Wissen, Av. 10 de Agosto s/n y Jose Ma. Sanchez, 010107 Cuenca, Ecuador
[5] ETSI Informáticos, Universidad Politécnica de Madrid, Campus de Montegancedo s/n, 28660 Boadilla del Monte (Madrid), Spain

Abstract. In the last decade, both the mobile market and app development have seen exponential growth, with millions of apps in online stores and billions of downloads recorded. Several development processes are used for app development, with agile approaches being the best fit for the market model. However, there is a weakness in testing activities and tasks. To provide a tool to improve software quality, we present a proposal for a framework for software testing in apps. Our proposal combines a series of tools used to evaluate an application in key areas such as consistency, security, compatibility, stress, among others, the development activities that provide a deliverable, and Mobile Ilities. To evaluate the feasibility of using the proposal, a toy app was used. The results reflect situations where the app's behavior is not desired and its relationship with mobile ilities. The main contribution of the paper is to guide app developers in evaluating apps and contributing to improving app quality.

Keywords: Framework · Testing · Mobile applications · App · Mobile Devices

1 Introduction

Technological advances in the mobile field have grown rapidly in the last decade. Both devices and applications have become an essential part of our daily lives. For the development of mobile applications, several process proposals seek to obtain applications quickly and with high quality. Proposals such as Mobile-D [1] and IFFMAD [2] demonstrate an interest in improving mobile development. However, Abrahamsson [1], Martinez [2] and Jabangwe [3] agree that there is

© The Author(s), under exclusive license to Springer Nature Switzerland AG 2025
O. Gervasi et al. (Eds.): ICCSA 2025, LNCS 15650, pp. 68–80, 2025.
https://doi.org/10.1007/978-3-031-96962-1_5

a weakness in the testing process of mobile applications. Mobile app testing is primarily based on manual testing, where testers evaluate the functionality and user interface on different devices [4]. However, manual testing alone is not sufficient to address the increasing complexity and diversity of mobile applications due to new challenges arising such as support for multiple platforms, screen sizes, resolutions, OS versions, and network configurations. Moreover, the rapid release of updates and new functionalities makes the adoption of agile methodologies in mobile app development the best choice of development approach. In this paper, we present MobileTestPro, a proposal for testing mobile applications based on techniques and approaches used both academia and industry. Through the review of existing methodologies and tools, and best practices, we seek to establish a framework that allows development teams to improve the quality of their mobile applications, enabling a superior user experience while maintaining high standards of security and performance. The proposed framework focuses on performing fast, repeatable, and scalable tests on different mobile devices and platforms. Aspects of mobile testing such as usability, stress, accessibility, compatibility, security, and energy consumption are addressed, providing specific recommendations to evaluate and improve each of these aspects. The proposed framework was validated through the evaluation of a toy application. Section 2 presents a background. Section 3 then details the proposed framework. The following section describes the evaluation of the feasibility of the proposal through the evaluation of one application developed by students of the computer science degree, and Sect. 5 discusses the results obtained. Finally, Sect. 6 presents conclusions and lines of future work.

2 Background

This section describes a series of definitions that contribute to understanding our proposal.

2.1 Development Frameworks

A software development framework is a pre-established structure or platform that contains set of basic common tools and components that are used to help you create your software application. They offer reusable code, libraries, and conventions to prevent developers from writing the same code each time from scratch enabling them to work more efficiently, consistently, and scalable [5]. The most important aspects of a frameworks are Code re-usability, consistency, build on modularity, security, and maintain ease. There are a several kins of frameworks, the mos popular are frameworks for domain Specific for example Web Development Frameworks like Django (Python), Ruby on Rails (Ruby), Angular (JavaScript) in respective backends, front-end developments. Mobile Development Frameworks likes IFFMAD, Flutter (Dart), React Native (JavaScript), Xamarin (C#). Backend Development Frameworks such as Spring (Java), Express (JavaScript), laravel frameworks (PHP). Finally, we have Framework for frontend development for examples React. js, Vue. js, Angular.

Frameworks can cut down on development time because common features and tools are already implemented. They address the global quality of your software in enforcing best practices and providing out of the box debugging, testing.

2.2 Software Life Cycle and Testing

According to [6] software life cycle is "the period that begins when a software product is conceived and ends when the software is no longer available for use. The software life cycle typically includes a concept phase, requirements phase, design phase, implementation phase, test phase, installation and checkout phase, operation and maintenance phase, and, sometimes, retirement phase" Each of these phases involves a series of activities and tasks that contribute to the development of the application. The development of mobile applications has a series of restrictions that are defined by the mobile environment itself, which Martinez et al. [2] call Mobile Ilities. However, the development phases are mostly maintained. The principal aim is to develop quality software, although all phases contribute to this objective, testing activities take on greater importance because they are the ones that help to identify errors, failures, and faults in the applications. Software testing definition according to SWEBOK [7] consists of the dynamic verification that a program provides expected behaviors on a finite set of test cases, suitably selected from the usually infinite execution domain.

2.3 IFFMAD and Mobile Ilities

Mobile ilities are the mobile aspects that must be considered during the mobile application development process. [2] propose a set of eight Mobile Ilities that address since connectivity problems to platforms and device diversity. This sections presents a description of each eight Mobile Ilities.

- **Compatibility**: The application must ensure availability despite a lack of data connectivity or bandwidth limitations.
- **Interoperability**: The application should consider resource sharing and communication between internal and external applications using appropriate APIs.
- **Flexibility**: The application should consider adaptability to different mobile contexts and user configurations such as screen brightness level, connectivity preferences, power consumption, etc.
- **Energy**: The application should reduce the power consumption of the mobile device battery, for example, by reducing the number of network connections.
- **Device Heterogeneity**: The application must behave properly on both high-end and low-end devices, regardless of processing capacity, ensuring effective integration with hardware such as HD cameras, GPS, NFC and Bluetooth, fingerprint scanner, etc. It is also necessary to consider the user experience on small screens.

– **Platforms**: The application should ensure that users of different platforms have a common baseline user experience.
– **Data Security**: The application must manage internal and external threats, and vulnerability caused by lack of user knowledge, device attacks, or intentional misuse.

3 MobileTestPro

The proposed framework is composed of 3 blocks (see Fig. 1). The first bock groups the development activities that generate some deliverables that can be evaluated to identify failures. The development activities considered in the proposal are those related to requirements, design and construction according to SWEBOK v3.0 [7] and suggested in the Integrated Framework For Mobile Application Development IFFMAD proposed by Martinez et al. [2].

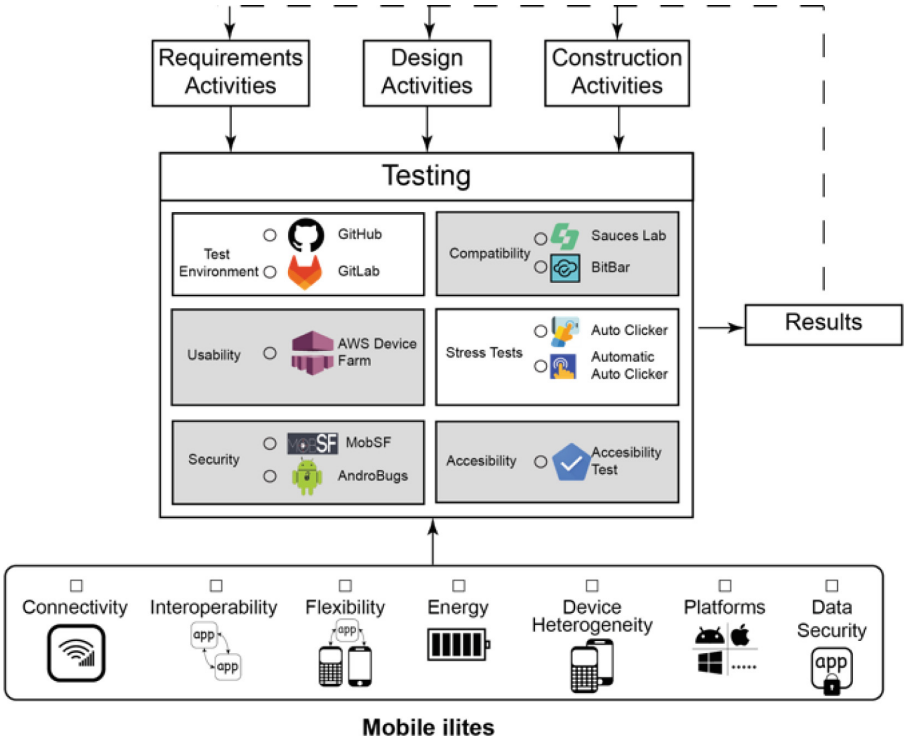


Fig. 1. MobileTestPro Framework proposal.

The second block groups the suggested tools to carry out the evaluations of the inputs obtained by the development activities. Finally, the Mobile Ilites [2]

these are the relevant property in the development of mobile applications to be taken into account during the development of an app. In the following sections describe the proposed activities and tools for testing applications and explain their key components in detail.

3.1 Energy Consumption Test

Mobile devices are battery operated, allowing the user a degree of freedom to move anywhere [8]. However, if the battery power consumption increases, the degree of freedom decreases, therefore limiting the user's mobility. In this context, power consumption tests help us identify those applications that make use of different components of the device that increase power consumption and can lead to overheating, which can cause the application to stop being used by the user. In our proposal, we recommend AccuBattery [9], because this tool evaluates the energy consumption in real time while the applications are running, resulting in the average consumption and the discharge speed as can be seen in Sect. 4. With this information, the application that consumes the most energy is identified, and the necessary actions can be taken to optimize energy consumption.

3.2 Usability Test

Usability is a key quality attribute of a mobile app, given the high level of competition in the app market. The International Organization for Standardization (ISO) defines usability as "the extent to which a product can be used by specified users to achieve specified goals with effectiveness, efficiency and satisfaction in a specified context of use" [10]. Usability testing is a challenge because it requires having a representative set of users using the app to carry out specific tasks to achieve specified goals. Even if there is the need of real representative users, some usability testing tasks can be automated. We have chosen for our framework the tool AWS Device Farm. It allows to use remote sessions to conduct usability tests with target users. The tests can be carried out with a variety of devices, and both in Android and iOS operating systems. Ensuring a seamless experience across devices and operating systems contributes to the overall usability of a mobile app, and an automated environment able to test in a wide range of devices contributes to this aim.

3.3 Security Test

The information stored on a mobile device is an asset that can be subject to attacks taking advantage of the internal and external vulnerabilities of the app, as well as those threats caused by the user's lack of knowledge. We propose the MobSF [11] and AndroBugs [12] tools to identify and mitigate potential vulnerabilities in the application. This kind of testing allows for the detection of errors in both code and incorrect configurations that could be exploited by

attackers seeking to ensure the protection of user information and compliance with privacy and data security regulations. Maintaining user trust and avoiding potential legal consequences arising from security breaches is a priority in developing secure and reliable applications.

3.4 Compatibility Test

The diversity of mobile devices, especially those using Android, is extensive. The main differences identified are storage capacity, and screen length, among others. Hence, it is difficult to guarantee that an application will run on the majority of devices on the market. Compatibility testing is performed using tools such as Sauce Labs [13] and BitBar [14] to ensure that the application works correctly on a wide range of devices and operating systems. This is essential to ensure that the app is accessible and functional for all users, regardless of the device they use. By reaching a broad and diverse user base, the app's reach and effectiveness in the market is maximized.

3.5 Stress Test

Stress testing, using tools such as Auto Clicker and Automatic Auto Clicker, is crucial to assess application performance under extreme conditions. These tests help identify bottlenecks and ensure that the application maintains its performance and stability even under high user load or adverse situations. Robust performance is vital to ensure user satisfaction and continued application reliability in critical environments.

3.6 Accessibility Test

Finally, for accessibility testing, Google's Accessibility Scanner [15] is proposed. Although accessibility testing is a complex activity per se, this tool allows you to check some basic elements that can be automated. This makes this task easier, allowing all users to be taken into account, including those with disabilities. Accessibility testing seeks to ensure that the application complies with accessibility regulations, promoting inclusion and guaranteeing equal access to technology for all users.

3.7 Process Flow

The process flow for using the framework is shown in Fig. 2.

The process flow start with the selection of artifacts to be tested, which can be inputs derived from requirements, design, or build activities. During the initiation phase, we identify quality standards, potential risks, and project requirements. Them, in the planning phase, we define the test plan for an iteration, ensuring compliance with the expected quality levels. During the design phase, we prepare the test cases and resources necessary to ensure readiness and optimization for execution. Execution executes and monitors tests to verify that

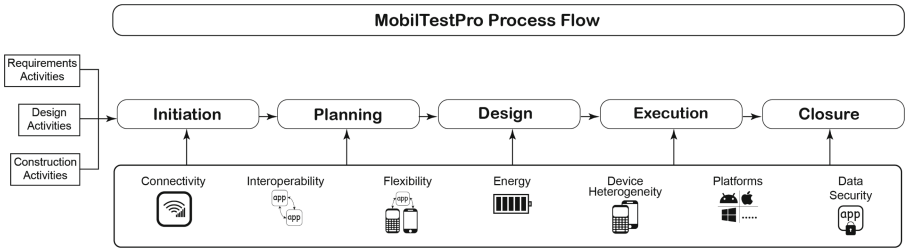


Fig. 2. MobileTestPro Framework proposal.

the software meets acceptance criteria and quality standards. Finally, the closure phase consolidates and communicates the results, metrics, and information obtained from the testing process.

4 Results

To evaluate the feasibility of using the proposed framework, a toy application (tic-tac-toe) developed by computer science students from a university in Ecuador was used. For the evaluation of this application, compatibility, stress, accessibility, energy consumption, and security tests will be considered. The version of Android 12 is used, in this case available in the different simulators since it is free. The results obtained are shown in the following sections.

4.1 Compatibility Test

Table 1 shows the results obtained in the evaluation of application compatibility on three devices that vary in CPU type, internal storage capacity and RAM.

We identified problems on the devices Motorola Google Nexus 6 7.1.1 y LG Google Nexus 5 7.1.1 -EU, el APK where it could not be installed because an SDK update is required. The message displayed was:

"*Faul [INSTALL_FAILED_OLDER_SDK: Error de análisis durante installPackageLI: /data/app/vmdl2018386247.tmp/base.apk (on line of file XML binary #7). The built-in version of SDA was #25 and version #26 is required.*"

The test helped verify compatibility with devices using SDK (Software Development Kit) versions higher than 25, verifying the correct operation of the application. However, some devices were unable to install the application due to limitations in the update capacity of their software versions.

4.2 Stress Test

With the aim to evaluate the robustness and stability of the applications under control conditions, Clicker was used to simulate intensive and repetitive use of the application. Failures and performance bottlenecks are identified, emphasizing

Table 1. Compatibility Results

Name	OS	GPU	API	RAM	CPU	Storage	Result
Google Pixel 3a	Android 12	Adreno 615	32	4 GB	8-core Kryo 385 (2.8 GHz / 1.7 GHz)	64 GB	✓
LG Nexus 5	Android 7.1.1	Adreno 330	25	2 GB	4-core Krait 400 (2.3 GHz)	16 GB	✗
Motorola Nexus 6	Android 7.1.1	Adreno 420	25	3 GB	4-core Krait 450 (2.7 GHz)	32 GB	✗

the Mobile Ilities of Flexibility and Interoperability. Taking into account that Auto Clicker works from Android versions 7.0 and higher, a physical device with Android version 12 was used. The stress test is performed 3 times. Parameters such as test duration, interval, number of cycles, and amount of time for each cycle will be considered. The first test lasted 3 min where normal stress conditions were simulated, a correct operation of the application could be observed where the parameters taken into account were very low. In the second test that lasted 6 min, high stress conditions were simulated and it could be observed that the time per cycle is 200 ms, therefore, the application could not function correctly, it did not process the clicks. Finally, in the last 9-minute test under continuous and not very complex stress conditions, the application worked correctly as shown in the Table 2.

Table 2. Stress Test

Test Duration (min)	Interval (min)	Number of Cycles	Amount of Time (ms)	Results
3	1	10	300	✓
6	1	20	200	✗
9	1	30	300	✓

In real-world environments, with a normal number of clicks, the application operates adequately due to its limited set of functionalities, reducing the possibility of user errors. However, if the number of clicks increases beyond the normal range, the application tends to not respond adequately.

4.3 Accessibility Test

Accessibility Scanner by Google was used to verify that all user interface elements are accessible via screen readers. This allows us to identify improvements to the app such as enlarging small touch areas, increasing the contrast between text and images, and providing content descriptions for graphical elements without labels. Testing was performed on a physical device with Android version 6.0 or higher. Testing was performed on the application screen, and no suggestions for improving the main screen are identified in the results. On the second screen, it is noted that the item shown in Fig. 3 may not have a label to be identified by the screen reader.



Fig. 3. Accessibility Scanner suggestion in second screen.

Table 3 shows all the suggestions made.

Table 3. Accesibility Scanner Suggestion

Screen	Condition	Suggestion
1	Game Started	No suggestions for improvement identified
2	Game not Started	This item may not have a label that can be read by an on-screen reader
2	Game Started	It is possible that this element has more than one label that can be read by an on-screen reader

4.4 Energy Consumption Test

AccuBattery [9] was used to measure the power consumption of the application during its active use. This test is associated with the Mobile Ilitie of Energy. The test was performed on a physical device with Android version 7. During the test AccuBattery runs in the background and the average consumption is calculated while the application is running. The energy consumption was analysed in time periods of 5, 10 and 15, during which time the energy wear is observed as can be seen in Table 4.

Table 4. Energy Consumption by time

Time	Battery Consumption	Battery Consumption Speed
5 min	0.8%, 21.6 mAh	10.8%/h, 299.5 mA (21.6 mAh)
10 min	0.8%, 21.6 mAh	10.8%/h, 299.5 mA (21.6 mAh)
15 min	0.8%, 21.6 mAh	10.8%/h, 299.5 mA (21.6 mAh)

An analysis was then conducted to identify optimization opportunities to extend the battery life of devices without compromising user experience. The results obtained in the test suggest that the consumption and drain of the battery during the use of the application is minimal. This low energy consumption indicates that the application, apparently, is well optimized in terms of energy efficiency, which is a positive feature and seeks to ensure a long user experience.

4.5 Security Test

The application under review, by its nature, does not handle sensitive data that requires a high level of security. However, in order to test the suggested tool, this test was carried out to verify its correct operation in this area. The Mobile Ility associated with this test is Data Security. MobSF (Mobile Security Framework) is an open source platform for penetration testing, malware analysis and security assessment of mobile applications (Android/iOS/Windows). MobSF was used for vulnerability detection and remediation. The analysis of the results that the application gave us are in Table 5.

The analysis identifies security issues that make the app vulnerable. Installing older versions increases the app's vulnerability, enabling debugging mode makes it easier to connect debuggers to access sensitive information. Finally, accessible data backup allows the extraction of app data from the device. It is also suggested to update the app to a version compatible with Android 10 or higher, disable debugging mode in the app, and restrict access to the data backup.

5 Discussion

Testing activities attempt to demonstrate to the developer and the client that the software meets requirements and to identify situations where the software's behavior is incorrect, undesirable, or not in accordance with its specifications [16]. In the mobile context, in addition to the app meeting the functional requirements, it must satisfy requirements that are not necessarily explicit, but that influence both the development of the app and its performance Martinez et al. [2] calls these particularities Mobile Ilities. Our proposal presents a framework that groups together both testing activities and the tools that best fit the mobile field, which are linked to Mobile Ilities. Table 6 shows the relationship that has been identified between Mobile Ilities and the tests proposed in the framework.

Table 5. MobSF Results

Affair	Gravity	Description
The application can be installed on a vulnerable version of Android (Android 8.0, minSdk = 26)	Warning	This app can be installed on an older version of Android that has multiple vulnerabilities. Supports Android version ≥ 10, API 29 to receive reasonable security updates
Debugging enabled for the application	High	Debugging has been enabled in the application, making it easier for reverse engineers to plug a debugger into it. This allows them to dump a stack trace and access debugging helper classes
App data can be backed up [android:allowBackup=true]	Warning	This flag allows anyone to back up their application data via ADB. Allows users who have enabled USB debugging to copy application data off the device

Table 6 shows that Mobile Ilities is related to at least one testing activity proposed in our framework, except for connectivity, because our toy app do not require an internet connection to exchange information. However, connectivity is very important because a large number of apps need to be connected to a network for normal operation. A lack of connectivity could cause the app to malfunction and even become inoperable. The proposed framework comprises a series of development activities that generate an artifact, however, for this study only the construction activities that generate a prototype were used. The wide variety of Android and SDK versions are some of the restrictions we did not encounter, so the application presented problems in older versions of Android, especially those lower than version 10, and SDK versions lower than 25. The problems identified are related to compatibility and security related to the mobile capabilities of Heterogeneity Device and Data Security. Stress tests identify that the application responds adequately under normal and average conditions, while under extreme conditions it fails and does not respond to interactions on the screen. Related to accessibility of the application, there are not many new features, except for the lack of labels for some elements and the existence of elements with the same description. It is recommended to carry out tests with real users to obtain a more complete perspective. The application's energy consumption is minimal and homogeneous during all the tests performed, which suggests that the algorithms are optimized and the interface design is balanced and uses few light colors. This test is linked to Mobile Ility Energy. In terms of data security, vulnerabilities have been identified related to older versions of Android that are more vulnerable. Enabled debugging mode makes it easier to access sensitive information and accessible data backup allows data to be extracted from the device, which could lead to the theft of sensitive user information.

Table 6. Relationship beetwen Mobile Ilities and Test activities proposal

	Test					
Mobile Ilities	Compatibility	**Stress**	Accessibility	Energy Consumption	Security	Usability
Connectivity						
Flexibility		✗	✗			✗
Interoperability		✗				✗
Energy				✗		
Device Heterogeneity	✗		✗			✗
Platforms	✗		✗			
Data Security					✗	

6 Conclussion and Future Works

We have proposed a framework for app evaluation, which suggests the different tests that should be performed on apps related to Mobile Ilities. The feasibility of the proposal is assessed by applying it to the development of a toy app by students of the software engineering degree. Following the recommendations of the framework, 5 tests were carried out using various testing tools that allowed us to identify certain deficiencies in the application. The main contribution of the paper is to guide app developers in evaluating apps and contributing to improving app quality. Furthermore, the relationship between Mobile Ilities and app testing activities is presented. As future work, more complex tests such as partition equivalence, decision tables, mutations and checking whether the application works correctly with any type of technology used for test development if required can be increased. In addition, it is suggested to implement automated usability tests using tools such as Espresso and Appium to detect possible problems in the user flow. To optimize accessibility tests, specialized tools such as Ax for Android could be incorporated. In addition, strengthening security assessments by integrating static and dynamic analysis with tools such as SonarQube and OWASP ZAP would allow vulnerabilities to be identified more thoroughly. Device connectivity tests should also be implemented. The proposal must be evaluated using a real-world application, which requires defining efficiency, effectiveness, and user satisfaction metrics to compare with other app evaluation approaches and contribute to software product improvement.

References

1. Abrahamsson, P., Hanhineva, A., Hulkko, H.: Mobile-d: An agile approach for mobile application development. In: Companion to the 19th Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications, pp. 174–175 (2004). http://dl.acm.org/citation.cfm?id=1028736
2. Martinez, D., Ferre, X., Guerrero, G., Juristo, N.: An agile-based integrated framework for mobile application development considering ilities. IEEE Access **8**, 72461–72470 (2020). https://doi.org/10.1109/ACCESS.2020.2987882

3. Jabangwe, R., Edison, H., Duc, A.N.: Software engineering process models for mobile app development: a systematic literature review. J. Syst. Softw. **145**, 98–111 (2018). https://doi.org/10.1016/j.jss.2018.08.028
4. Zhan, W., Yan, G.: Testing of mobile applications: a review of industry practices. https://www.bth.se, retrieved September 4, (2024)
5. Villalobos, G.M., Darío, G., Sánchez, C., Alberto, D., Gutiérrez, B.: Diseño de framework web para el desarrollo dinámico de aplicaciones. Scientia et Technica **16**(44) (2010). http://www.pdf4free.com
6. IEEE: 610.12-1990 IEEE standard glossary of software engineering terminology (1990). https://doi.org/10.1109/IEEESTD.1990.101064
7. IEEE Computer Society: Guide to the Software Engineering Body of Knowledge (SWEBOK Version 3.0). IEEE (2014)
8. Perrucci, G.P., Fitzek, F.H.P., Widmer, J.: Survey on energy consumption entities on the smartphone platform. In: IEEE Vehicular Technology Conference, pp. 1–6 (2011). https://doi.org/10.1109/VETECS.2011.5956528
9. AccuBattery: User manual – accubattery help center, https://accubattery.zendesk.com/hc/en-us/sections/202339749-User-manual. retrieved September 4, 2024
10. International Organization for Standardization: Ergonomics of human-system interaction—part 210: Human-centred design for interactive systems (2019)
11. Abraham, A.: Mobile security framework. https://mobsf.live/about, retrieved September 4, 2024
12. AndroBugs: Androbugs framework. https://www.androbugs.com/, retrieved September 4, 2024
13. Sauce Labs: Sauce labs: Cross browser testing, selenium testing & mobile testing (2024). https://saucelabs.com/
14. SmartBear: Bitbar dashboard (2024). https://cloud.bitbar.com/#testing/dashboard
15. Google: Accessibility scanner - apps on google play (2024). https://play.google.com/store/apps/details?id=com.google.android.apps.accessibility.auditor
16. Sommerville, I.: Software Engineering. Pearson, 7th edn. (2003)