Quality Characteristics of Mobile Learning Applications

Dinu MIHAIL-VĂDUVA
Bucharest University of Economic Studies
dinu.mihailvaduva@stud.ase.ro

Software quality evaluation for mobile application is a subject with major implications onto educational evaluating process. The place where teacher and his educable person build a temporary social relation is connected to an environment where success is correlated with a mobile-driven mindset. A suitable examination of such mobile learning - environment implies quality characteristics like functional correctness and testability and should be conducted with proper tools if we desire to use mobile learning applications for implementing into a collaborative scenario toward complex classroom events. In this article we are making a raised mark above quality characteristics of mobile learning applications because this is the starting point to acquire a certain comprehension concerning a better utilization of such methods for obtaining knowledge. We construct a potential bridge between two quality characteristics of mobile learning applications to show how to transfer practical results from one quality characteristic toward another to sustain an experimental study with adjacent outline involving mobile programming laboratory curricula. We focus our attention above a software quality metric, the so-called statement code coverage (CC), to obtain a synergic effect between a black-box approach and a white-box technique in order for developing and testing Android mobile applications. For reaching our desideratum we studied scientific papers regarding the complexity of quality characteristics applied to mobile learning applications, from a general to a particular view, to shed light above previous gathering observations concerning our mission. We have a genuine desire to reveal technical instructions needed to be executed in order to obtain a reproducible pattern of a manually testing process, concerning an Android mobile application, conducted by a beta-testing team of users.
Keywords*: Mobile Applications, Mobile Learning, Android, Quality Metrics, Econometrics*

1 Introduction

A general overview among quality characteristics of mobile learning represented by mobile applications implies a profound knowledge about standardized specifications related to software dedicated to be used inside a particularly hardware like those represented by mobile devices. Looking for more details about quality characteristics of mobile applications attached to any social activity in a way that could be orientated toward education we find a zone where previous general specifications of mobile applications suffer a transposition determined by a large variety of preferences or choices encounter at the level of user groups. Our path of research follows a direction from general specifications of quality characteristics allocated to mobile applications through discrete quality characteristics encounter at the level of mobile learning applications. In our article the experimental study is orientated toward a practical activity engaged to establish values of statement code coverage obtained through a beta testing process of evaluating the quality characteristic of **functional correctness and testability**. We consider that **functional correctness and testability** is a milestone for a system of quality characteristics of m-learning applications and our research involving such quality characteristics is motivated by the fact that the results could be implemented into a laboratory experiment executed at the level of a programming mobile discipline.

This paper is structured as follows. The first section is a general overview of the subject chosen about quality characteristics regarded mobile application toward mobile learning objective. The second section shows the salient facts of software quality characteristics of mobile application, discrete quality

DOI: 10.12948/issn14531305/23.4.2019.07

characteristics of mobile learning application and the reason for why we proceed with an evaluation about the statement code coverage metric in our research. The third section reveals our efforts to extract from scientific literature technical details about the statement code coverage evaluation, dedicated frameworks to measure it. The fourth section shows a methodology to investigate, with ACVTool, an evaluation about a quality percentage regarding statement code coverage for our own Android application [1], virtually adapted, using Java language. In this section we put a special accent toward ACVTool architecture. This section explains technical considerations and implications about the process of checking manually beta testing of an Android application binary file (apk), with an emulator available into Android SDK, from the standpoint concerning automatically evaluation process of the statement code coverage. The fifth section explains our proposed solution to check and prove with the help of a statement code coverage percentage and how to recognize with a general classification a pattern in doing a repetitive manual beta testing evaluation, with the help of the Android's Studio SDK emulator. In the sixth section we show our conclusion and future work related to other intentions regarding similar projects.

2 Quality Characteristics of m-Learning Applications

2.1 General overview about quality characteristics of mobile applications

A group of related types concerning quality characteristics, extracted and adapted from ISO/IEC 25010 [2], to be allocated for further interpretation against mobile learning applications contains references to functional suitability and maintainability as a root chapter toward other quality subcategories containing detailed intrinsic functional and maintainability areas like **functional correctness** and **testability**. The quality sub-characteristic to deliver the correct results, named functional correctness, meaning accomplishment and covering of specific tasks could be twined with the quality of sub-characteristic, named **testability**, designed to be a sign for effectiveness and efficiency of any test software criteria performed. Related to quality characteristic named **performance efficiency** there are three sub-categories concerning time behavior, resource utilization and capacity. Firstly, there is a direction toward processing times and outcome rates regarding utilization of its source code, and secondly, the importance of what kind of resources used and in which amounts is the objective of the sub-characteristic named resource utilization. Finally, the capacity tends to be an indicator to the maximum limits of the parameters allocated to the product. **Compatibility** is another quality characteristic [2] involved into co-existence and interoperability issues. Co-existence is a marker for sharing a common software component in such a way that no negative impact occurs and interoperability shows a predisposition to exchange and use the information in a duplex manner. **Usability** is a complex quality mobile application characteristic that regards satisfaction of users who desires to achieve efficiency and effectiveness. A user wants an easy way to recognize if a mobile product is appropriate for a satisfactory utilization and, in addition, a less time to acquire knowledge to a proper handling of functions. In first case we are taking about the ability for a suitable recognize process and, supplementary to that, for learning to use, control, and access with minimum effort. A user shows satisfaction if the product protects him against any inherent error that could occurs and user interface aesthetic enables a better interaction. The quality characteristic named **reliability** [2] is a milestone for coagulating mandatory conditions needed to operate under normal parameters, above inherent hardware faults, with the ability for a recover action, if the situation required, or to be fault tolerance. An attribute of quality regarding **security** is always a point where different technical areas converges coming from prevention about unauthorized access to, tampering of, source code or documentation. Non-repudiation is, in this case, the quality attribute for security

DOI: 10.12948/issn14531305/23.4.2019.07

characteristic that offers certain prove related to a past event that cannot be rejecting later. The non-repudiation cannot be done if an automatic procedure, attached to the mobile application, to trace actions or events is not available and this capability, therefore, is named accountability. A resource or user connected during the normal utilization of the mobile application must be recognized as an authentic one if a proved formal guarantee is delivered. The ability to be not difficult for analysis, altering and testing is always related to the quality characteristic named **maintainability** [2]. If effectiveness of a successfully installation of a mobile application product, into different hardware, was acquired then the quality characteristic could be named portability.

2.2 Specific quality characteristics of m-learning applications

A research about discrete quality characteristics related to m-learning applications was conducted by Pocatilu [3] from which we learn that the most important characteristics were aggregated around loading time, path length, level of user required information, continuity of interaction between human and mobile application, the length of the path toward resources required by the application and the complexity of the used components. From this research [3] they conclude that an important quality characteristic, 17%, is connected to the time, expressed in seconds, needed to load logical interfaces located inside the mobile applications and, not a less important one, 15 %, is the level of the information required by the mobile application against user profile. From all of these quality characteristics mentioned above we consider that **functional correctness and testability** of the mobile application is a common denominator if the statement code coverage could be analyzed through an experimental study.

It is known that the quality characteristic named **functional correctness and testability** has a metric named code coverage which is a measure in computer science that embraces a percentage of code that could by calculated when a specific software test case was done. Java code coverage tools require technologies that insert statements to app's Java source code and afterward recompilation plus instrumentation to the byte code. Automatic testing concerning multiple versions of an Android application versus a single version of the same Android Application started a research, finished by Quoan [4], from which we could see that a major component of their framework was a code coverage generator that inserts automatically instrumentation for every block in byte code level tested in such a way that logs generated exposed which block is executed or not. A recent study realized by Pilgun [5] reveals that code coverage of an Android application is an important metric to evaluate efficiency which is a key attribute regarding software quality evaluation. The main distinction between Pilgun's research and the general trend of other software that covers code coverage inspection was that his method involves a black-box approach with no need to access Android app's code source. Pilgun's tool repository was published [6] and could be implemented by anyone using a software resource that implies Python, Android SDK, and Java to investigate the smali code coverage through any Android's apk package regarding the classes, methods and instructions. This research team [7] reports a percentage of 96.9 % apps successfully instrumented with ACVTool with a small or not noticeable time overhead. Our research helps readers to investigate code coverage of their Android applications knowing that a similar goal is a path to a better efficiency regarding time consumption when developers build an Android project.

3. Literature Review

A compelling study about **functional suitability**, regarding completeness and data accuracy, was conducted by Puspaningrum [8] which shows that **performance efficiency** could be measured for an Academic Information System, which delivers support for learning teaching activities, keeping into account the relation between function and

DOI: 10.12948/issn14531305/23.4.2019.07

function suitability. His measurements turn out to be relevant to a quality improvement for the institution than the separate quality characteristics evaluation according with ISO/IEC 25010 only. The framework used in this research was based on Goal-Question-Metrics paradigm that helps them to identify those educational activities or tasks important for quality metric measurement. From researcher Sari [9] we learn about the suitability of mobile application architecture between a native mobile application and a web-based one. She studies for the web-based solution different m-learning scenarios, effective development and the process of mobile application deployment and the limitation in browser functionality. For native mobile application there is a central focus to main benefits and development complexity considering a collateral effect related toward a hybrid solution to simplify the process of developing native mobile application. The final conclusion of researcher Sari [9] was that native mobile application is a better choice for m-learning activities due to the capability of outperforming regarding offline availability, user experience and security options against other mobile application solutions. The quality characteristic named **performance efficiency** was studied by Olajide [10] from which we learn about the favorable parameters required by Android native application concerning processor time utilization, consumption of energy at the level of internal battery, usage of internal memory and the total time required for loading web pages. The author Wikanaso [11] studied how to implement different ways to calculate efficiency of **usability** as a tracking indicator for the performance of the educational organization which is supporting suitable framework for this research. They used [11] a psychometric scale, named Likert Scale, where respondents offer their approval or disapproval regarding o series of statements starting from a total disagree to an end of complete agreement about the user's satisfaction encounter during utilization of an Android mobile application. The quality characteristic named **reliability** was studied by Meskini [12] using four theoretical reliability growth models applicable both to the desktops and mobile devices hardware against two mobile applications, Skype and Vtok. They concluded [12] that the mobile applications area failed to obey with the rules established for desktops / laptops reliability growth models due to the differences traced back to the development life cycle DLC allocated to mobile applications. For example, because of inherent market constraints, as they said, there is a possibility to skip phases like design level which is a very important criterion into production of a mobile application. They [12] calculated a metric named Mean Time Between Failure, MTFB, for mobile applications under different conditions to observe cumulative curves regarding failure time. Another author Kaur [13] was involved into a research about software **reliability** metrics using an agile process against development of a software product. They emphasize different metrics like rate of occurrence of failure, probability of failure on demand, availability and mean time to repair. Linked to the **security** of a mobile application the researcher Khokhlov [14] studied a group of internal and external metrics with certain influences related to the data security evaluation. Amongst them a central focus was placed about possible attacks of sensor data manipulation of a legitimate user or a non-legitimate user of a mobile device. They finished [14] a classification concerning internal and external metrics which contains references toward internal issues like root access, device lock, Android OS version, device model, installed applications but and, on the other hand, device rating and system vulnerabilities when we speak about external metrics.

Related to functional correctness and testability a research to calculate code coverage using black box and white box techniques during dynamic tests over Android applications was completed by Horvath [15] which underpins the idea that a synergetic effect will be obtained if we follow a central position between them. They emphasize that a black box technique answers to the question

DOI: 10.12948/issn14531305/23.4.2019.07

what is the result of an application when an input is assigned to it, and the white box answers to the question how the previous result is obtained. This package of techniques returns a code coverage criterion and it offers an important feedback on the quality of the software. Their results regarding code coverage was studied following a path through test cases, redundant code and traceability calculations. The subject of the user's point of view about the necessity to search quality software metric using code coverage reports, through Android mobile developers, was a key target for scientific research generated by [16] which deploys a survey regarding code coverage issues, automated testing opportunities and test cases. This survey was harbored using the online platform named Qualtrics and the respondents were involved using email. The final outcome from this study was orientated through building of a body of knowledge which will be considered an important asset to any researcher or developer in this field. A study about the process of obtaining code coverage for Android dynamic analysis tools was realized by [17] which disclose their approach as a set of connected processes between decompiling an apk file, inserting instrumentation code, recompiling, repackaging and testing the final product modified. They report a low rate of successfully repackaging Android apk file, situated at the value of only 36%, but the code coverage results were similar to those prefabricated with EMMA tool. EMMA [18] is compatible with all Android devices with version 4.0.3.or higher and it is recognized as a tool for Java code coverage. In no way connected to the static analysis previously mentioned they started a new approach using offline tools like Android emulator and DroidBox to determine code coverage on the same apk packages used in static analysis. DroidBox [19] is a software tool to build dynamic analysis of Android applications which is known by the fact that two graphs could be generated automatically. First graph is related to the temporal sequences of the operations and the second one is presented using a tree map layout that offers parallel similarity between package actions. They reported similar results between offline and online tools but some observations were influenced by the UI operations commonly encountered like login activity, license messages or pop-up ads. Online tools used in this study like Anubis, TraceDroid and CopperDroid are examples of discontinued software products which cannot be used. A study about the quality of industrial applications regarding code coverage and fault detection under examination of Google's tool named Monkey [20] was performed by [21] to identify user interface behavior and consequences related to it, reproduced by human gestures, under an assault of randomly user interface streams events sequences. The Monkey tool from Google returns the highest method coverage on 22 of 41 Android applications whose method coverage was obtained. This result was an answer to the question regarding what is the code coverage obtained by a test generation tool over an industrial Android application. Monkey tool can be used as a stress – test software in a random style and is embedded into emulator or device where it runs. Other test generation tool selected for this study was DroidBot [22] which allows developers to design scripts for a personalized strategy exploration. DroidBot is a test generator input for Android application that generates a transition graph after testing. The main characteristics about DroidBot convey to the fact that it does not require application instrumentation, could produce user interface structures for further analysis, events are not based on a random model but a GUI one and it is programmable. For code coverage measurement they used a tool named Ella [23] to perform collecting statistics of method coverage. Ella tool is designed to instrument Android applications to record which methods were being touched with time-stamped value registered or estimated values of parameters of methods. This tool requires Unix operating system, Android SDK and Java SDK. Another study about measuring code coverage using EMMA was done by a team [24] which shows that

DOI: 10.12948/issn14531305/23.4.2019.07

their technique requires an injection of a method call during the application testing process and the EMMA tool was launched by an instrumentation activity responsible with starting of the main activity of the every Android application studied. They proposed a software solution, Dynodroid, that is an input generation system for Android applications with a technique constructed around the cycle defined by the sequence observe – execute – select. Their algorithm establishes which events are important to the application in a predefined stage and then select and execute one of these events for the purpose of obtaining a new state where the whole cycle is repeated in the same way mentioned through above sequence. They recognized the importance of the human arbitrary actions when we need an activity, for experimenting some of the Android application functionalities like entering passwords or choosing different settings values, that generate a pause into Dynodroid generating mechanism but resume it immediately after the user's choice is realized.

4. Methodology

We have developed our previous Android application [1] and let's say that we are using two teams for our research, one consisting of programming developers, dubbed here with A label, and the other one, named with B label, for beta testing process. Our research question is the following: *How do we know if an improvement of a predefined class from our Java's source code, completed by a user from A-Team, is properly manually tested by B-Team, according with an imposed path of sequences, through a manually beta testing process?* The answer of this question is linked with a proper classification of code coverage recorded during white box development of the Android application.

To deliver a beta-testing plan from A-Team toward B-Team there is a need to envision a flow of sequences that should be followed by every user, from B-Team, implied in those activities required by a properly manually beta-testing process. This flow of sequences would be depicted as a general structure like our proposed model shown in Figure 1.

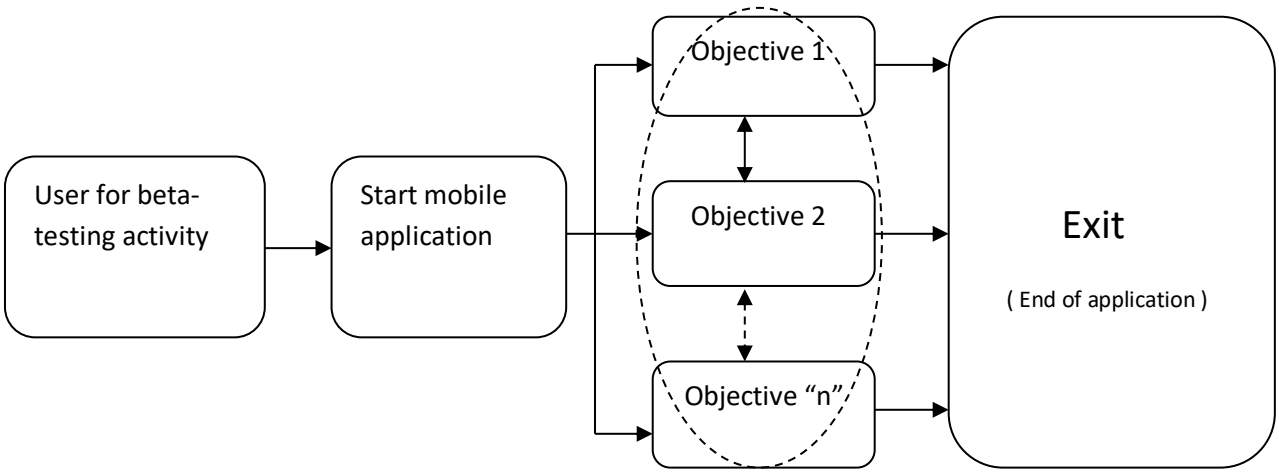

Fig. 1. General model for mobile application beta testing

From Figure 1 we see that the whole beta-testing process requires a complete running of all operational objectives mentioned from number "1" to "n". Those operational objectives could be actions embedded into mobile application options that trigger, for example, generating demo database, inserting a new record, searching a value, deleting a record or generating a report with results. If an user implied in a beta–testing process execute a complete circle from objective "1" to "n" then the final statement code coverage obtained will be according with the classification of statement code coverage documented by the A-Team. If not all objectives from "1" to "n" will be completed then a difference for statement code coverage would be visible. It is necessary to say that repeating any particular objective, for example an objective like inserting a new

DOI: 10.12948/issn14531305/23.4.2019.07

record, and omitting another objective, does not influence the final code coverage to achieve the proper statement code coverage obtained by a correct usage of beta-testing required plan. The software procedure ACVTool, developed by Pilgun [5], to reveal a statement code coverage requires an offline and online phase. The offline phase is defined by all software preparation needed to start emulator for Android application, instrumentation of mobile application's apk and starting a separate shell for working with Android Debug Bridge. The online phase is defined by the process of working with instrumented apk file following the required objectives and preparing final report with statement code coverage. The research team [5] establishes a performance of ACVTool publishing a value of 96.7 % healthy instrumented apps with an average total instrumentation time situated at the value of 36.2 seconds per application.

5. Case Study

5.1 Application description

Using an adequate classification, from white box's point of view, we could establish a framework for a technical explanation about the flow of sequences really tested by an end user, in charge with a beta testing duty, against the primary objective required by the A-Team. For understanding our study, we propose a generic and general model for a beta testing manually process of a predefined class, embedded into an Android application, that could be seen as a set of predefined paths, required to be tested by A-Team, like those depicted into Figure 2.

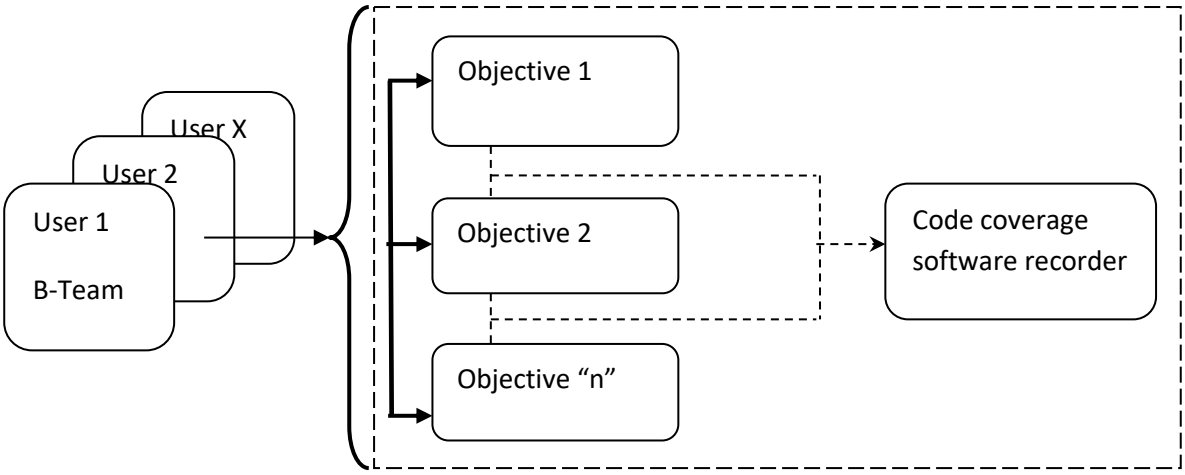

Fig. 2. The flow sequence requested by A-Team for beta testing process required to be used by those users from B-Team

Using Figure 2 we try to explain that a group of different users, from B-Team, will be asked to execute a beta testing process, only following by the path defined by the flow sequence 1-4-6 and, specifically, will not to be tested using the flow 2-3-5. It is obvious that if above requirement will not be accomplished, during the phase of beta testing, then the effort done to any improvement of the Java class 1 from Figure 2 could not be seen by the any code coverage software recorder. On the contrary a high score of statement code coverage recorded during "i" beta testing process should be interpreted as an indicator that the flow sequence 1-4-6 asked from A-Team was completed normally. It is important to say that the A-Team is concerned only to improve the Java's class, shown in Figure 2, named with "Java 1", and this will be, for our above research question, predefined class needed to be improved. The model applied to those users from B-Team will be considered as a black box one, where we know what is the result, and the model applied to the A-Team will be considered as a white box, from which we know what the cause regarding of the result is obtained by a black box model. We know from [15] that merging of these two models should obtain a synergetic effect concerning

DOI: 10.12948/issn14531305/23.4.2019.07

the quality of the software.
In order to evaluate statement code coverage, we are using a software recorder previously developed by a research team [5]. The ACVTool [5] is capable to establish a measure of the level concerning statements, methods and classes executed or touched during a dynamic testing applied of an Android Java application, without knowing its source code, known by the name of the statement code coverage. This method generates detailed description supported by HTML and XML format from which we obtain tree level reports viewing expanded or collapsed packages, through inner classes, methods and instructions, presented by subsequently numerical calculations when instrumentation was completed. The architecture of ACVTool is represented by three levels as depicted in Figure 3.

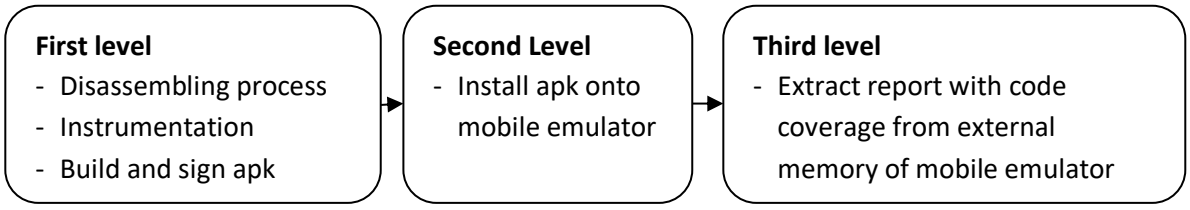

Fig. 3. The ACVTool methodology used to study the statement code coverage

First level shows that original apk suffers a modification generated by some byte code probes injected into a smali code version of the apk original file. This smali code version of the original apk file was automatically obtained during a previous disassembling process supported by *backsmali* disassemble [25] integrated into apk tool. After this process a new one is automatically executed meaning a repackaging of the whole content into a new apk file with the help of the *apksigner*. The last tool, *apksigner*, is located under Android Studio SDK and is used to properly sign the apk file to confirm that apk's signature will be verified successfully on all versions of Android platforms. This level generates an instrumentation report that will be combined with runtime report produced after the third level into a file with *pickle* extension. The second level shows that modified apk is installed into an emulator or a physical device and, afterward, a process of initiating instrumentation is launched.
The third level shows that ACVTool initiate a drag operation of a runtime report from external memory of the device, used in previous phase, where it was recorded during Android application testing. This report is overlapped with the instrumentation report generated by the first level. This report offers a sum of information starting with the name of the smali package followed by a percentage number indicating the ratio of the code coverage and numbers representing missed versus executed or touched instructions along the process of testing the Android application.

5.2 Application implementation

Our research methodology was cyclically implemented using a new architecture illustrated in Figure 4.
For our study we choose to apply ACVTool to our previous Android Java application [1] to evaluate the efficiency of the statement code coverage metric when we want to classify the activity of beta testing accomplished by a group of users. We proceed for collecting values concerning statement code coverage with ACVTool to find a general classification of statement code coverage established during white box's development. Our software resources allocated to this study are represented by Android Studio SDK from which we are using the adb.exe (Android Debug Bridge). In this case, the emulator used in our experiment is nexus_5x_api_27. Our operating system was Windows 10. We used three interfaces to operate with adb tool, ACVTool and the last one used for starting the Android emulator. There is necessary to say that testing for our Android Java apk project was conducted in a manual style and Android Studio was not loaded during this methodology.

DOI: 10.12948/issn14531305/23.4.2019.07

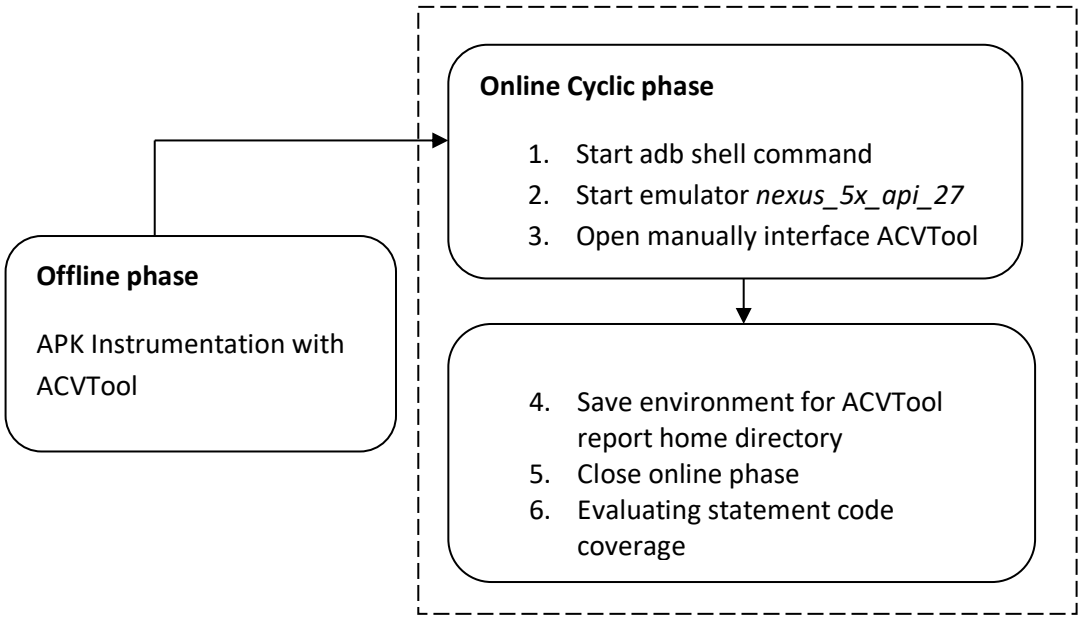

Fig. 4. The methodology used to study in our research the statement code coverage

We choose a manual apk testing style because in our research we want to evaluate any modification with regression analysis regarding the correlation between dates recorded with ACVTool when the team in charge with testing Android apk application executes different task at the level of the emulator. The stage 4 in figure 4 is necessary because any new testing and evaluation for statement code coverage with ACVTool overwrite the local default directory used to save the final HTML and XML runtime report. Our study refers to the general framing of different use cases, regarding statement code coverage, at the level of using our Android application into emulator, presented into the Figure 5.

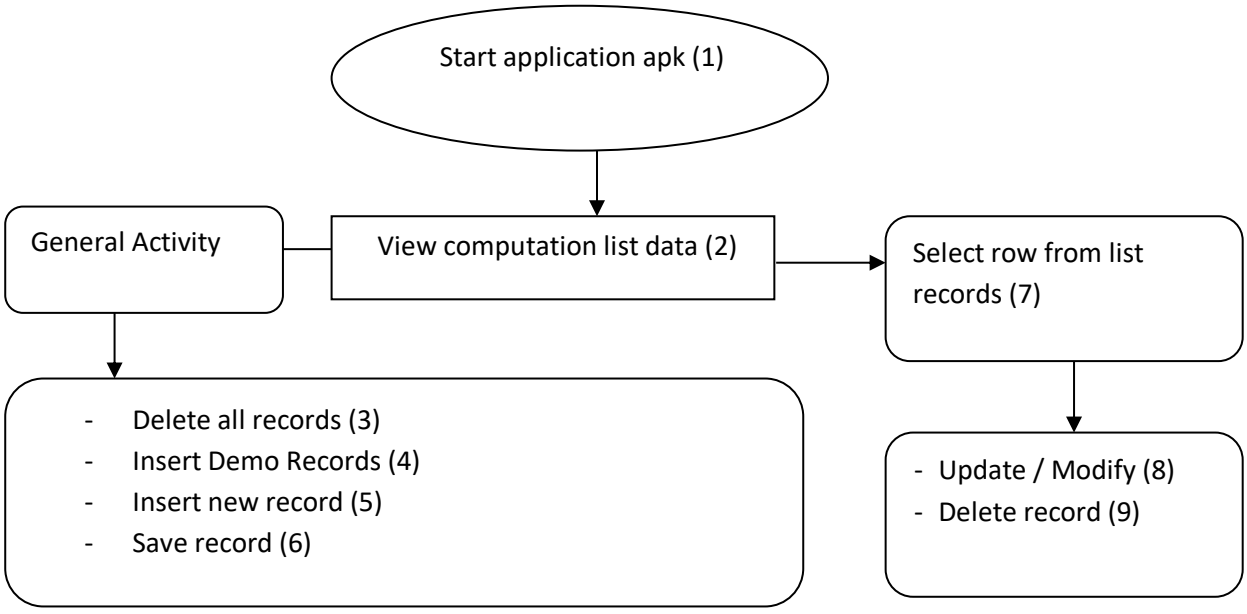

Fig. 5. The cases used for flow sequence applied to beta testing process

The target class *DatabaseProject* and it was monitored using ACVTool. Our experimental data, from white box's point of view, are classified in Table 1.

DOI: 10.12948/issn14531305/23.4.2019.07

Table 1. Data obtained related to statement code coverage white box use cases

Interval	Minimum Value CC	Maximum Value CC	Flow Sequence
1	0	9.69	Activate application's interface (1)
2	11.47	15.67	Insert + Save + Delete all records (1-5-6-3)
3	15.67	17.44	Insert + Save (1-5)
4	17.44	35.7	Insert database Demo (1-4)
5	35.7	37.47	View computational list (1-2)
6	37.47	39.25	View computational list + Select on row (1-2-7)
7	39.25	40.87	Insert database Demo + Insert + Save + Delete all Records (1-4-5-6-3)
8	47.33	67.37	Insert database Demo + Update + Delete (1-4-2-7-8-9)
9	83.36	96.93	Insert database Demo + Update + Delete + Save new records (1-4-2-7-8-9-6)

6 Interpretation

From black box's point of view, a group of user in charge with beta testing duties uses the Android application instrumented with AVCTool and loaded into an emulator and obtained the values recorded into the Table 2.

Table 2. Data obtained related to statement code coverage

User	Code Coverage (CC) (%)	Missed Line	Line	Missed Method	Method	Missed Class	Class	Interval Classification for beta testing
1	39.25	376	619	12	19	0	1	6
2	40.87	366	619	12	19	0	1	7
3	67.37	202	619	6	19	0	1	8
4	95.15	30	619	3	19	0	1	9
5	35.7	398	619	12	19	0	1	5
6	39.25	376	619	12	19	0	1	6
7	67.37	202	619	6	19	0	1	8

Using general classification shown into the table 1 we consider the following outcomes classified after the code coverage obtained. In this way we have a technical disclosure about the activity deployed by any end user in charge with beta testing activity. The formula used to calculate the values, allocated to statement code coverage, indicated by code coverage (CC), in Table 1 is:

$$\text{Statement code coverage (CC)} = 100 - \frac{Missed\ Line}{Line} \times 100 \quad (1)$$

From Table 2 we concluded that only user with number 4, with 95.15 % code coverage (CC), has reached the flow sequence asked for a complete beta testing required. All statement code coverages shown in table 1 are reproducible results. The highest value obtained for the statement code coverage is assigned to the exhaustive way of doing a beta testing process against an Android application. The flow of sequence studied by a beta testing process will have the statement code coverage established by the highest code coverage embedded into the primary sequences. Testing the composite elements of a flow sequence will not have a meaning that all statement code coverage allocated to every element will be added to the final result. If an end user in charge with beta testing repeat execution for a function, then this fact does not mean a highest code coverage result. In this case the code coverage result is determined by the use case specified by the function itself.

DOI: 10.12948/issn14531305/23.4.2019.07

The scenario used to manually test the apk application has a starting point during the second level indicated under the fifth label in figure 2. After installation of instrumented apk into emulator and starting it we proceed to use and test the Android apk application using only a pattern of actions preselected.
We proposed that arithmetic mean formula for overall statement code coverage allocated to the Java sources of the Android application apk should be calculated without taking into consideration values reported for Android support classes, R classes or other classes automatically generated and always embedded into any final apk product according with the latest Android technology production.

7. Conclusions and Future Work

Technical conditions discovered, seen as a software perspective, for a valid implementation of ACVTool, toward a beta testing process of Android applications, using code coverage metric, will be correlated with a proper utilization of Android operating system, Android Debug Bridge to consolidate the path between a mobile emulator and ACVTool, Java and Python. There are real opportunities to track the evolution of different beta testing scenarios, inside of a specialized educational laboratory for mobile programming, when we try to manually testing an Android application. The information recorded into runtime report generated by ACVTool could be automatically extracted and inserted into a database log file to a future interpretation. For this objective there is a need to use SQLite with the help of Android Debug Bridge. The statement code coverage is a software quality metric that could be rapidly evaluated and generates feedback actions to improve general structure of an Android's application and is an excellent topic for a laboratory dedicated to study programming and testing Android applications for mobile devices. The following directions will be considered to improve our work:

Acknowledgment

Parts of this research have been published in the Proceedings of the 18th International Conference on Informatics in Economy, IE 2019 [26].

References

[1] D. Mihail-Vaduva, "Enriching Curricula with Mobile Solutions," Informatica Economică, vol. 22, no. 3, 2018.
[2] IEC25010. [Online]. Available: https://iso25000.com/index.php/en/iso-25000-standards/iso-25010. [Accessed 04 April 2019].
[3] P. Pocatilu and C. Boja, "Quality characteristics and metrics related to m-learning process," Anfiteatru economic, nr. 26, Vol XI., Iunie 2009.
[4] D. Quoan, Y. Guowei, C. Meiru, H. Darren and R. Jefferson, "Redroid: A Regression Test Selection Approach for Android Applications," in Proceedings of the International Conference on Mobile Software Engineering and Systems, Austin, Texas, USA, 2016.
[5] A. Pilgun, O. Gadyatskaya, S. Dashevskyi, Y. Zhauniarovich and A. Kushniarou, "DEMO: An Effective Android Code Coverage Tool," in Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security , Toronto, Canada — October 15 - 19, Pages 2189-2191, 2018.
[6] A. Pilgun, "pilgun/acvtool," Oct 2018. [Online]. Available: https://github.com/pilgun/acvtool. [Accessed 9 March 2019].
[7] A. Pilgun, O. Gadyatskaya, S. Dashevskyi, Y. Zhauniarovich and A. Kushniarou, "Fine-grained Code Coverage Measurement in Automated Black-box Android Testing," eprint arXiv:1812.10729, December 2018.
[8] S. Puspaningrum, S. Rochimah and J. Akbar, "Functional Suitability Measurement using Goal-Oriented Approach based on ISO/IEC 25010 for Academics Information System," Journal of Information Systems Engineering and Business Intelligence, vol. 3, no. 2, pp. 68-

DOI: 10.12948/issn14531305/23.4.2019.07

74, 2017 October.

[9] M. N. Sari and M. Ali, "The Suitability of Native Application for University E-Learning Compared to Web-Based Application," International Journal of Science and Research (IJSR), vol. 4, no. 1, 2015.

[10] A. Olajide and O. Adebola, "Performance Evaluation of Native and Hybrid Android Applications," Communications on Applied Electronics (CAE), vol. 7, no. 16, 2018 May.

[11] S. Wicaksono, D. Firdausy and M. Saputra, "Usability Testing on Android Application of Infrastructure and Facility Reporting Management Information System," Journal of Information Technology and Computer Science, vol. 3, no. 2, pp. 184-193, 2018.

[12] S. Meskini, B. Nassif and L. Capretz, "Reliability Models Applied to Mobile Applications," in IEEE 7th International Conference on Software Security and Reliability-Companion (SERE-C), Gaithersburg, M.D., USA, 2013.

[13] G. Kaur and K. Bahl, "Software Reliability, Metrics, Reliability Improvement Using Agile Process," IJISET - International Journal of Innovative Science, Engineering & Technology, vol. 1, no. 3, May 2014.

[14] I. Khokhlov and L. Reznik, "Data Security Evaluation for Mobile Android Devices," in 20th Conference of Open Innovations Association (FRUCT), St. Petersburg, Russia, 3-7 April 2017.

[15] F. Horváthy, S. Bognáry, T. Gergelyy and R. Ráczy, "Code Coverage Measurement Framework for Android Devices," Acta Cybernetica 21, p. 439–458, 2014.

[16] M. Linares-Vasquez, C. Bernal-Cardenas, K. Moran and D. Poshyvanyk, "How do Developers Test Android Applications?" in International Conference on Software Maintenance and Evolution (ICSME), Shanghai, China, 17-22 Sept., 2017.

[17] C.-Y. Huang, C.-H. Chiu, C.-H. Lin and H.-W. Tzeng, "Code Coverage Measurement for Android Dynamic Analysis Tools," in IEEE International Conference on Mobile Services, New York, NY, USA, 27 June-2 July, 2015.

[18] "EMMA: a free Java code coverage tool," [Online]. Available: http://emma.sourceforge.net/. [Accessed 19 March 2019].

[19] P. Lantz, "pjlantz/droidbox," [Online]. Available: https://github.com/pjlantz/droidbox. [Accessed 19 March 2019].

[20] "UI/Application exerciser Monkey," [Online]. Available: http://developer.android.com/tools/help/monkey.html. [Accessed 19 March 2019].

[21] W. Wang, D. Li, W. Yang, Y. Cao, Z. Zhang, Y. Deng and T. Xie, "An Empirical Study of Android Test Generation Tools in Industrial Cases," in ACM/IEEE International Conference on Automated Software Engineering, Montpellier, France — September 03 - 07,, 2018.

[22] "honeynet/droidbot," [Online]. Available: https://github.com/honeynet/droidbot. [Accessed 19 March 2019].

[23] "ELLA: A Tool for Binary Instrumentation of Android Apps," [Online]. Available: https://github.com/saswatanand/ella. [Accessed 3 March 2019].

[24] A. MacHiry, R. Tahiliani and M. Naik, "Dynodroid: An Input Generation System for Android Apps," in Joint Meeting on Foundations of Software Engineering, Saint Petersburg, Russia — August 18 - 26, 2013.

[25] JesusFreke, "smali/backsmali. 2018.," JesusFreke/smali, 2018. [Online]. Available: https://github.com/JesusFreke/smali. [Accessed 10 March 2019].

[26] D. Mihail-Vaduva, "Android App Code Coverage," in Proceedings of the 18th International Conference on Informatics in Economy (IE 2019), pp. 351-358, , Bucuresti, 30-31 May 2019, ISSN 2284-7472.

DOI: 10.12948/issn14531305/23.4.2019.07

Dinu MIHAIL-VĂDUVA has graduated the Faculty of Economic Cybernetics, Statistics and Informatics of the Bucharest Academy of Economic Studies in 2010. He received an alumnus achievement award in 2012 of the Informatics Economics Master affiliated to the Bucharest University of Economic Studies. Starting with 2017 he was admitted to the PhD student at the Doctoral School of The Bucharest University of Economic Studies, Economic Informatics domain for his PhD thesis proposal involving educational systems based on mobile technologies. Between 2016 and 2017 he finished with a high school diploma postgraduate training programs named Psychological, Pedagogical and Educational skills training, level I and II, connected with a particular branch of the Bucharest University of Economic Studies. Furthermore, he graduated classes between 1984 and 1989 with a bachelor degree of the Faculty of Technology for Chemistry of the Polytechnic Institute of Bucharest towards specialization concerning Inorganic Chemistry Engineering. Currently he is working as a software analyst within the Department of Information Technology at the Regia Autonomă "Monitorul Oficial" from Bucharest and he is using Oracle database with APEX technology correlated with specific software languages for intranet projects development. His main scientific preoccupation is heavily orientated towards domains situated at the interface between education as a primarily economic and social activity and the most recent software technology.

DOI: 10.12948/issn14531305/23.4.2019.07