# TESTING OF MOBILE APPLICATION IN THE WILD: A LARGE OF EMPIRICAL STUDY ON ANDRIOD APPS.

### Authors:

### Fabiano Pecorelli, Gemma Catolino, Filomena Ferrucci, Andrea De Lucia, and Fabio Palomba.

**Conference:** Mon 13 Jul 2020 15:00 - 15:12 at ICPC - Session 1: Tests Chair(s): Dario Di Nucci

[**Paper link**](https://conf.researchr.org/details/icpc-2020/icpc-2020-research/11/Testing-of-Mobile-Applications-in-the-Wild-A-Large-Scale-Empirical-Study-on-Android-)

**Media link**: <https://youtu.be/vKBweIraEyw>

# ****Introduction and motivation****:

This passage is the combined work of Fabiano Pecorelli, Gemma Catolino, Filomena Ferrucci, Andrea De Lucia, and Fabio Palomba of Sesa lab of the University of Salerno, Italy. In the present era many mobile phone and devices have been created to ease the daily tasks of people. However, this had also lead to the growth of the app industry. Several mobile apps are being sold in the market for over 205billion downloads on a day to day basis. Therefore, it is important for app industries of develop and deliver high quality products in order for them to be sold and gain more users for a high commercial success. One of the major factors that play an important role in the growth of app industry is the software development as it enables understanding of the functionalities of the system. However, if not given attention to it could create high risk if defects and lower software reliability. In order to improve the software, research community has tested several techniques that could better support mobile developers. These include testing papers on graphical user interface (GUI) and a framework to ease testing activities. Although despite of all the efforts made there is still more room for improvement in the software development to ensure a more effective and comprehensive automated testing approach. Hence, many software developed conduct tests manually. In addition, very little attention is given to software a testing detail which further lowers the quality of the produced software. For this reason, this paper focuses on a large scale empirical study on prominence, quality and effectiveness of the tests manually written by mobile developers. A dataset of 1780 open source Android applications and extract tests written by developers was considered with the goal of understanding the quantity and quality of tests. Furthermore, in the second step the design of the tests is considered and lastly the effectiveness of mobile test cases is measured. An empirical study suggests that there is insufficient amount of mobile app testing. In addition, the tests are mostly unit level and concerned with the logic while GUI and storage of the apps remains untested. Moreover, the levels of metrics observed are fairly suggesting a poor fault detection in the tests.

The study stressed on 3 most important research questions:

* RQ1. To what extent are test suites developed in the context of mobile apps?
* RQ2. What is the design quality of test cases developed in mobile apps?
* RQ3. What is the effectiveness of test cases developed in mobile apps?

The context of the empirical study consisted of 1,780 open-source Android apps gathered by mining F-Droid, 1 a repository of free and open-source mobile applications that has been widely employed in the past and that contains a set of applications that enables a good generalizability of the findings with respect to the overall population of free and open-source mobile apps. It is important to note that, while F-Droid contains over 3,000 apps, only repositories on Github were considered. Furthermore, duplicated apps and forks of those already existing in the repository were excluded. Based on these filters, we ended up with the final 1,780 open-source mobile apps

# ****Methodology****:

The methodology of

* RQ1 is classified into two phases. The first phase is the tuning phase which specifies how inspectors separated classified a set containing 500 test suites and annotated their granularity and types in a spreadsheet. Whenever possible, the inspectors relied on the documentation available

**Example**:

code comments) to understand the properties of a certain test. Other times, the inspectors relied on the name of the class as well as analyzed its content to check if only a production class was exercised, i.e., it was a unit test, more classes were involved to verify the interactions among components, i.e., it was an integration test, or otherwise, it was a system test. A similar strategy was employed when classifying the type: whenever possible, the inspectors relied on the documentation, while in other cases they manually went over the code to understand which functional or non-functional requirement was exercised. Through the classification of the same test suites, the inspectors could tune their judgments, find a common way to classify. The second phase is classification in which production and test classes are linked. In order to do so the string “test” is removed from the name and search and search the production class that matches the remaining part of the name. later, a number of production classes having a relational test suite were conducted. For the type of production classes tested, first automatic classification was carried out then double-checked manually. Specifically, a set of keywords that can distinguish GUI, application logic, and storage components of an Android app was defined. For the sake of space limitations, a complete list of keywords was included. Since this automatic classification may be erroneous in some cases, one of the authors of the paper double-checked it and corrected the labels assigned whenever required.

In the methodology of

* RQ2 Firstly, we computed test code quality metrics, relying on the metric suite and other metrics related to code quality. We computed the Lines of Code (LOC): according to previous achievements having higher size may cause issues for developers with respect to the maintainability of tests as well as to their fault-proneness. from our empirical analyses it was observed that, while the metric profile of tests would not show potential problems affecting their design, the quality of tests is still threatened by the presence of test smells Despite the fact that they capture two different concepts, this contradiction may potentially indicate that currently available metrics are not enough to measure the actual quality of test suites and, as such, new, different test code metrics that better capture the design quality and understandability of test suites should be defined. Lastly
* RQ3 is focused on two complementary aspects that have been shown to influence the ability of tests to catch defects in production code, namely statement coverage and assertion density . The former measures the amount of lines of production code touched by a test suite during its execution. As for the assertion density, this is defined as follow: Assertion. Density (Tc) = #assertions (Tc)/ LOC (Tc) where etc is the test case under consideration, #assertions(tc) is the number of assert statements in tc and LOC(tc) is the number of lines of code of the test.

# ****Result****:

It was observed that most of the tests in the dataset refer to tests exercising the application logic of mobile apps. There were several explanations to the observed results. Firstly, developers may not be properly supported nor aware of the available techniques when it came to the testing of other aspects of their apps. Secondly, it may seem that they prefer to focus more on the testing of the main functionalities of their apps rather than storage or GUI-related classes. Finally, mobile developers are sometimes junior or with less experience than programmers working in other domains and might be less aware of the importance of testing, hence limiting themselves to exercise a limited amount of classes. Furthermore, other results observed show that while the metric profile of tests would not show potential problems affecting their design, the quality of tests is still threatened by the presence of test smells. Despite the fact that they capture two different concepts, this contradiction may potentially indicate that currently available metrics are not enough to measure the actual quality of test suites and, as such, new, different test code metrics that better capture the design quality and understandability of test suites should be defined. Finally it can be concluded that test suites are mostly not effective: the median code coverage and assertion density are 0.23 and 0.17, respectively. Additional analyses revealed that, in cases where developers take care of their tests, they seem to be rewarded with a reduction of production code fault-proneness. Furthermore, we found that in some cases developers perceive code coverage as highly relevant to accept pull requests.