Journal of Computer Science

Original Research Paper

SCUTMA: Selecting Components for Unit Testing in Mobile Applications

[1]Josias Gomes, [1]Isabel K. Villanes, [1]Silvia M. Ascate, [3]Awdren Fontão, [2]Eduardo Noronha de Andrade Freitas and [1]Arilo Claudio Dias-Neto

[1]Institute of Computing, Federal University of Amazonas, Manaus – Brazil
[2]Department of Informatics, Federal Institute of Goias, Goiania – Brazil
[3]Faculty of Computing, Federal University of Mato Grosso do Sul, Campo Grande – Brazil

Article history
Received: 15-07-2020
Revised: 28-09-2020
Accepted: 14-10-2020

Corresponding Author:
Josias Gomes
Institute of Computing, Federal University of Amazonas, Manaus – Brazil
Email: josias@icomp.ufam.edu.br

Abstract: Context The source code of a mobile application has several components (i.e., code units) and they have a cost (e.g., test creation and execution) and benefit (e.g., find bug) value for performing unit testing. Problem Choosing in which components unit tests should be done, in order to increase the benefit (effectiveness) and decrease the cost of this activity. Proposal This study presents an approach that supports the selection of components for unit test creation in Android mobile applications. The SCUTMA approach allows the selection of components with respect to the following criteria, which can be combined: Cost of future maintenance, propensity to defects, frequency of call, risk of failure, market vulnerability and business value. Results Two empirical studies were performed to evaluate SCUTMA. The first study aimed at analyzing the correlation between the metrics that represent the criteria described above, where the result showed the possibility of using the metrics Cost of Future Maintenance (CFM), Code Smell (CS), Frequency of Calls (FC), Fault Risk (FR), Market Vulnerability (MV) and Business Value (BV) combined in a solution for component selection. The second study aimed to evaluate the effectiveness of using the SCUTMA approach. The result indicates that SCUTMA selected more components with error (58.33%) than the selection performed by specialists (40%). Conclusions The SCUTMA approach is effective in selecting components for creating unit tests in Android mobile applications.

Keywords: Mobile Application Testing, Automated Testing, Unit Test, Component Selection, Metric Correlation

Introduction

Mobile applications (or simply, mobile apps) are increasingly present in our daily lives. They are perceived in many areas, such as social networking, entertainment and e-commerce (Rubinov and Baresi, 2018). Therefore, guaranteeing mobile app quality is essential and one way to evaluate their quality is designing, conducting and executing software testing. Software testing can be applied in several levels. One of these levels is unit testing, in which individual components[1] of the source code are tested (Jorgensen, 2018). Testing at the unit level brings some benefits, such as reducing flaws in existing resources, improving code structure, decreasing side effects and reducing fear of code change (Burke and Coyner, 2017).

Despite the importance of testing a mobile app, testing all its components is often impractical due to its high effort in a software project. Thus, selecting which components to test is a challenge, seeking an ideal subset of components is most likely to contain defects and that brings less testing effort to the project (de Andrade Freitas *et al*., 2016). The existing technical literature has a limited set of studies that help developers in choosing components for performing unit testing in mobile apps (de Andrade Freitas *et al*., 2016). Thus, developers need to choose, based on their experience, which mobile app components have the best value (i.e.,

[1]In this study, a component refers to a method of a class in an object oriented language.

cost versus benefit) for unit testing. However, developers with little experience in this task will face barriers when selecting components.

The selection of components for unit testing can be modeled as a multiobjective optimization problem within the Search-Based Software Testing (SBST) area, which has attracted much attention in recent years (He *et al*., 2017). Within this research line is the pre-test effort, in which the test code is selected in order to guide the test construction to attain the maximum coverage based on several metrics. Some metrics that can be used are: Long components in code (Liu *et al*., 2015), components with fault risk based on failure history (de Andrade Freitas *et al*., 2016), components that centralize the accountability of a class (Vidal *et al*., 2016), components that call many other components of other classes (Hecht *et al*., 2016), among other possibilities.

In this context, this work proposes an approach based on Genetic Algorithm (GA) to help developers in selecting components that have a greater value in relation to the cost-benefit of unit testing of Android mobile applications (because Android is the most popular and open mobile platform). We implemented the approach as a plugin called Selecting Components for Unit Testing in Mobile Applications (SCUTMA) that was added as an extension at Android Studio IDE. To measure the cost and benefit value of the components against the selection criteria, the following metrics were chosen based on a technical literature review performed by the authors: Halstead Effort (HE), Cost of Future Maintenance (CFM), Code Smell (CS), Frequency of Calls (FC), Fault Risk (FR), Market Vulnerability (MV) and Business Value (BV). The developed plugin extracts these metrics and then runs the NSGA-II multiobjective genetic algorithm to select the components to be tested.

In order to evaluate the proposed approach, two empirical studies were carried out. In the first study, we present a controlled experiment aiming to analyze whether there is a correlation among the selected metrics (CFM, CS, FC, FR, MV and BV), because a strong correlation of a pair of metrics indicates that they should not be used together. For the experiment, two Android open source mobile apps belonging to two different categories were selected. These mobile apps have test scripts[2] of system (a pre-condition of the proposed approach to use one of the criteria for the selection of unit tests). We selected 13 devices for the execution of these test scripts. Results indicate that the CFM, CS, FC, FR, MV and BV metrics can be used together. However, in case the test scripts are incompatible with different devices, we recommend the use of FR or MV.

In the second study, the goal was to analyze the effectiveness of the SCUTMA approach in selecting components with error[3] compared to manual selection performed by unit test specialists[4] in Android mobile apps. For the experiment, we used two Android mobile apps that had test scripts, which were also run on 13 mobile devices. For the comparison between the specialists and SCUTMA, 63 usage scenarios were constructed. The result showed the feasibility of the proposed approach to assist developers in the selection of components for unit testing, since the SCUTMA plugin selected more components with error (58.33%) than the specialists (40%). The main contributions of this work are:

- The correlation analysis between the metrics CFM, FC, FR, MV, CS and BV
- Definition of an approach for component selection in mobile apps on the Android platform using an evolutionary algorithm based on static, dynamic, market and business metrics
- Analysis of an automated component selection using the SCUTMA plugin in relation to a manual selection performed by unit testing specialists in Android mobile apps

This paper is organized as follows: Section two presents the theoretical foundation, section three discusses related works, section four presents the approach and SCUTMA plugin, sections five and six present the planning, execution and analysis of the results and discussions of the studies. Finally, Section seven presents the conclusion and future work.

Background

This work is focused on the selection of components for unit testing: This activity can be modeled as a Multiobjective Optimization Problem (MOP). A MOP is a problem that has two or more objectives that can be optimized simultaneously. It is important to mention that it is common that the MOP objectives are in conflict with each other (Coello, 2006). For example, the tester simultaneously seeks to reduce the cost and maximize the benefit in the test execution, by selecting a subset of components (Harman *et al*., 2006; de Andrade Freitas *et al*., 2016).

A GA can be used to solve a MOP, the GA works by generating an initial population and, according to evaluation criteria, selects the best individuals from that population, which will serve as a solution to the problem or, otherwise, will be combined to get a new generation. This process is repeated until a solution is found or until it is realized that better solutions will not be achieved in the new generations (Coello, 2006). To automate the

[2]Step-by-step instructions that allow one to perform a test (Dwarakanath *et al*., 2018).

[3] The components identified through mobile app commits analysis.

[4] Professionals with experience in performing unit testing on Android mobile apps.

process of selecting a subset of components using a GA, it is necessary to measure some metrics. The metrics are functions, while the measurements are numbers obtained by the application of the metrics.

This paper is based on metrics identified from a technical literature review. We searched for papers that use metrics for source code selection. Code Smell (CS) metric was chosen because it identifies components that indicate weaknesses in the project that may be delaying development or increasing the risk of errors or faults in the future (Martin, 2009). The Business Value (BV) metric was chosen because it allows the identification of components that are of greater importance to the mobile app business. The CFM, FC, FR and MV metrics were based on the SCOUT method for the selection of components for unit testing in mobile apps presented by (de Andrade Freitas *et al*., 2016). For this paper, mobile app business is considered as the component that implements the main mobile app functionalities.

The basic premise is that if critical areas of the mobile apps source code are identified, the effort and cost of testing activities can be reduced. Thus, we define four categories (static, dynamic, market and business analysis) for the set of metrics used in this study, as follows.

Static analysis, which are collected by measurements made up of representations of the system (Somerville, 2011): CFM and CS; Dynamic analysis, which are collected by measurements made up of a program in execution (Somerville, 2011): FC and FR; Market analysis, which are collected by measurements that consider market information (de Andrade Freitas *et al*., 2016): MV; Business analysis, which are collected by measurements that consider the main functionalities (Zakaria *et al*., 2015): BV. The next subsections will detail each metric.

Cost of Future Maintenance (CFM)

Software change due to corrective and non-corrective actions (Bourque and Fairley, 2014). Each component has a propensity for the associated defect and in case of failure, software maintainers spend time to understand the system and make the appropriate changes.

Table 1: Six Hasltead's metrics

Metric	Symbol	Equation
Halstead Length	N	$N = N1 + N2$
Halstead Vocabulary	n	$n = n1 + n2$
Halstead Volume	V	$V = N * (\text{LOG}_2 n)$
Halstead Difficulty	D	$D = \frac{n1}{2} * \frac{N2}{n2}$
Halstead Effort	E	$E = D * V$
Halstead Bugs	B	$B = \frac{V}{3000}$

Legend: $N1$ = total number of operators; $N2$ = total number of operands; $n1$ = different number of operators; $n2$ = different number of operands.

The CFM metric is derived from two metrics: Halstead effort and halstead bugs (Halstead, 1979). The composition of these two metrics is presented in Table 1, they are based on the number of operators and operands of the source code. The equation to calculate this metric is presented in Equation 1. It is based on the paper of (de Andrade Freitas *et al*., 2016):

$$c\,fm_i = E_i * B_i \quad (1)$$

where, E_i is the amount of work in seconds (Halstead Effort metric) to understand and recode component i; and B_i is the estimated number of errors (Halstead Bugs metric) for component i.

Code Smell (CS)

It is a hint that something or part of the mobile app source code may cause some problem (Martin, 2009). Some code smells presented in the existing technical literature for the component level are:

- Long Method (LM): When the component contains more than 100 lines, not counting blank lines or comments (Palomba, 2015)
- Long Parameter List (LPL): When the component has seven or more parameters (Haque *et al*., 2018);
- Switch Statements (SW): When one has a complex switch operator, a sequence of nested if statements, or a nested repetition loop (for, while, do-while) (Haque *et al*., 2018)
- Brain Method (BM): Tends to centralize the functionality of a class in the same way that a God Class centralizes the functionality of an entire subsystem or sometimes even an entire system (Fontana *et al*., 2015)
- Cyclomatic Complexity (CC): When the number of possible paths through the source code is greater than or equal to five (Fontana *et al*., 2015)

Equation 2 is used to calculate the code smell value for each component:

$$cs_i = \frac{S_i}{5} \quad (2)$$

where, S_i is the number of code smell detected for component i.

Fault Risk (FR)

A good source of information is the fault history discovered in previous tests as well as the experience of the software engineer. To uncover the fault risk, test cases are designed specifically by software engineers

who try to anticipate the most plausible faults in a given mobile app (Bourque and Fairley, 2014).

The data needed to calculate this metric comes from the record of the execution of a component in test cases that have passed or failed. Equation 3 was extracted from (Jones *et al*., 2002; de Andrade Freitas *et al*., 2016):

$$rf_i = \frac{P_i}{\left(P_i + F_i\right)} \quad (3)$$

where, P_i is the ratio of the number of past test cases that executed component i to the total number of test cases passed in the test set; and F_i is the ratio of the number of test cases that failed and executed component i to the total number of failed test cases in the test set.

Frequency of Calls (FC)

During mobile app execution, the number of times a component is called is counted, because in static analysis the component may have been assigned a high priority. However, the impact of these static metrics must be associated in some way with a metric that reflects the level of request of a running component (de Andrade Freitas *et al*., 2016). This metric will compute the number of times a component was called while running the test cases.

Market Vulnerability (MV)

Expresses the vulnerability of a component among mobile devices according to market distribution (Android, 2020), such as API market and screen size. One way to calculate market vulnerability for a component is presented in (de Andrade Freitas *et al*., 2016) as follows:

- For each device, a list of failed test cases is created;
- For each component that was run by a failed test case, its minimum and maximum associated market is computed
- The Api Market (AM) of the component is calculated as the sum of the API market percentage of the devices that failed
- The component Screen Market (SM) is calculated as the sum of the screen size and the market density of failed devices
- The minimum vulnerability of the market can be expressed as the maximum value between AM and SM
- The maximum vulnerability of the market can be expressed as the sum of AM and SM
- The market vulnerable component is the average of the minimum and maximum market value Vulnerability

Business Value (BV)

Value Based Software Engineering (VBSE) aims to shift conventional software engineering practices to become business value-centric for system users (Zakaria *et al*., 2015). In this concept, each requirement, use case, object and defect of the mobile app are not treated as equally important. Therefore, use cases with a higher business value should have more test cases. This metric was adapted from (Ray and Mohapatra, 2012). The user sets a value for each test case and when a test case executes a component, the value of that test case is added to the component.

Table 2 shows a comparison between the metrics presented above.

This section presents the components selection as a multiobjective problem, also some metrics that can be used for this problem. The next section presents the related papers found in technical literature on component selection and prioritization, some of these papers use the metrics mentioned in this section.

Related Work

Related works were classified into two categories, according to the nature of the problem: (1) Component prioritization and (2) component selection, although the focus of this work will be the selection of components. First, we discuss the prioritization of components. Next, we discuss the component selection works. Finally, we compare some attributes of the works described with the proposed study.

Table 2: Comparison of metrics

Metric	Analysis	Requires execution of app UI tests	Based
Cost of Future Maintenance (CFM)	Static	Not	It is based on error risk and the required effort required to correct the error.
Code Smell (CS)	Static	Not	It is the propensity to defect based on code smells.
Fault Risk (FR)	Dynamic	Yes	It is based on the failures of the UI tests.
Frequency of Calls (FC)	Dynamic	Yes	It is based on the number of times the component is called during the execution of the app.
Market Vulnerability (MV)	Dynamic (Market)	Yes	It is based on the percentage of the device's market where the UI tests that run the component have failed.
Business Value (BV)	Dynamic (Business)	Yes	It is based on the weight assigned to the UI tests that run the component.

Component Prioritization

Component prioritization helps to create an order (priority) associated with components from defined criteria, such as code line coverage, risk exposure factor, among others, with the aim of revealing defects earlier (Ray and Mohapatra, 2012). Several studies have been conducted on known techniques and strategies for components prioritization. For example, (Shihab *et al.*, 2010) proposed an approach that leverages the project's development history to generate a prioritized list of components for unit tests writing. The list of components is updated dynamically as the development of the legacy system progresses. To evaluate this approach, a case study was conducted on a large legacy commercial software system. The results suggest that heuristics based on component size, frequency of modification and frequency of bug fixes should be used to prioritize the unit test writing of legacy systems that employ Test-Driven Maintenance (TDM).

Ray *et al*. (2011) proposed a program metric called influence metric to find the influence of a program element on the source code. The influence metric for a component m in a program shows the program statements number that directly or indirectly use the output produced by component *m*. The authors calculate the influence metric for a class c based on the influence metric of all its components. Experiments were conducted for two well-known case studies - Library Management System and Commercial Automation System - and prioritized critical elements in the source code of each case study. The experimental studies justify that this approach is more precise than those existing in the prioritization of critical elements at the implementation level.

Ray and Mohapatra (2012) proposed an approach that considers five factors of a component, such as influence value, average execution time, structural complexity, severity and market value as inputs and produces the component priority value as output. Experiments were conducted to compare this approach with a related approach. The results show that the approach that prioritizes the test effort within the source code is able to minimize the highly discriminated types of faults as well as the number of faults in the post-release time of a software system.

Mensah *et al*. (2018) proposed a prioritization scheme that mainly comprises the identification, examination and estimation of the rework effort of the prioritized tasks, for that they use the comments of the source code. The proposed prioritization scheme is a technique that helps in making decisions before launching the software, in an attempt to minimize overhead maintenance costs.

Component Selection

The first generic formulation for the Component Selection Problem (CSP) in the area of Search-Based Software Engineering (SBSE) was presented by (Harman *et al*., 2006), suggesting as future work the use of automated approaches employing search-based software engineering. After that, many papers have been proposed in different fields of software engineering, for example, software reuse (Ismail *et al*., 2008; Mahmood and Noman, 2014; Amjad and Khan, 2015) and the Next Release Problem (NRP) (Durillo *et al*., 2011; Zhang *et al*., 2013). In the field of unit testing for mobile apps, some works were found in this context: de Andrade Freitas *et al*. (2014; 2016).

de Andrade Freitas *et al*. (2014) presented a multi-objective evolutionary approach that looks for a subset of components for unit testing that minimizes cost while maximizing its strategic importance. To define the strategic importance of the component, the authors used static and dynamic metrics, such as cyclomatic complexity, operational coverage and frequency of modification. The experiments carried out in a real context on an industry web system, confirm the proposed benefits in selecting components by reducing cost and maximizing strategic importance.

de Andrade Freitas *et al*. (2016) presented an automated process for collecting static, dynamic and market value metrics and a multi-objective method for selecting components for unit testing called SCOUT (Selector of Software Components for Unit Testing). SCOUT can assist testers in different domains; however, experiments were performed on the Android platform and results show that SCOUT reduces market vulnerability compared to other methods. The study also compared the efficacy of seven algorithms in solving the component selection problem for unit testing, where the NSGAII genetic algorithm was the most effective. However, the SCOUT method does not consider some important metrics, for example, the Code Smell (CS) metric, which identifies components that are at risk of errors or faults (Fontana *et al*., 2015; Palomba, 2015; Haque *et al*., 2018); and the Business Value (BV) metric, which identifies components of greater importance to business mobile apps (Hosseingholizadeh, 2010; Ray and Mohapatra, 2012).

Our work was inspired by (de Andrade Freitas *et al*., 2016). We propose an approach that performs the selection of components based on the SCOUT method, including the metrics CS and BV that have been shown to be important for the component selection problem.

Table 3 presents a comparison with related works; it can be observed that most of the works perform experiment with source code written in the Java language. It is also highlighted that of the three levels, the level that most appeared in the works was that of component.

Related works presented in this section help to notice the importance of automating the selection of components for unit testing in mobile applications. The next section presents the proposed approach for component selection and the technologies used for implementation.

Table 3: Comparison of related works

Work	Category	Level	Language	Platform
Shihab *et al*. (2010)	Prioritization	Component	C and C++	Uninformed
Ray *et al*. (2011)	Prioritization	Class and Component	Java	Desktop
Ray and Mohapatra (2012)	Prioritization	Class and Component	Java	Uninformed
Mensah *et al*. (2018)	Prioritization	Class	Java and C	Web and Desktop
de Andrade Freitas *et al*. (2014)	Selection	Component	Java	Web
de Andrade Freitas *et al*. (2016)	Selection	Component	Java	Mobile
This work	Selection	Component	Java	Mobile

Legend: Work = reference to work; Category = informs if the work is component selection or prioritization; Level = specifies the part of the code that the selection or prioritization will focus on; Language = informs the language of the source code used in the experiment; Platform = refers to the platform on which the experiments were run.

SCUTMA Approach

Problem Modeling

The Component Selection Problem (CSP) for unit testing was mapped from an analogy of a theory extracted from Biology: The structure of chromosomes. As a chromosome has several genes and a population contains several chromosomes, in the work in focus, each gene represents a component and a chromosome represents a possible solution to the problem. From then on, we have the genetic representation for NSGA-II (Fig. 1).

For the present work, the NSGA-II multi-objective algorithm was used, based on results from (de Andrade Freitas *et al*., 2016) which evaluated seven algorithms (random approach, heuristic construction, gurobi, GA, SPEA-II, NSGA-II and NSGA-III) for CSP applied in unit testing. The result obtained showed that the NSGA-II algorithm proved to be the most effective.

NSGA-II uses the tournament selection technique. In this technique, a certain number of individuals of the population are randomly chosen. Then, there is direct competition from these individuals for the right to be a father, according to the objective functions of each one. The best individual in this tournament will be selected to be the father of the next generation. In conjunction with turner selection, elitism is used, in which NSGA-II ensures the maintenance of the best individuals of one generation in the next generation. The crossover operator that will be used will be the single point and the mutation operator will be the bitwise for binary-coded.

In this study there are two objectives: (1) Maximizing the benefit and, (2) reducing the cost of unit testing. The objective function for maximizing the benefit is represented in Equation 4:

$$\sum_{i=1}^{N} = \left(pCfm * cfmi + pFr * fri + pFc * fci + pMv * mvi + pCs * csi + pBv * bvi \right) / \left(pCfm + pFr + pFc + pMv + pCs + pBv \right) * xi \quad (4)$$

Where:

pCfm = Percentage of the cfm metric
cfmi = The cost of future maintenance
pFr = The percentage of *fr* metric
fri = Fault risk
pFc = The percentage of *fc* metric
fci = The frequency of calls
pMv = The percentage of the mv metric
mvi = The market vulnerability
pCs = The percentage of the *cs* metric
csi = Code smell
pBv = The percentage of the *bv* metric
bvi = The business value
xi = The value 1 (one) if component *i* is selected, otherwise it is 0 (zero).

Values for *pCfm*, *pFr*, *pFc*, *pMv*, *pCs* and *pBv* may be one of the following percentages (0; 0.2; 0.4; 0.6; 0.6 and 1.0), where 0 means 0% of the metric, i.e., the metric will not be used, 0.2 means 20% of the metric value and so on. The objective function to minimize cost was based on (de Andrade Freitas *et al*., 2016) and is represented in Equation 5:

$$\sum_{i=1}^{N} = ci * xi \quad (5)$$

where, *ci* represents the cost to develop unit test (Halstead Effort metric) for component *i*; and *xi* is the value 1 (one) if component *i* is selected, otherwise it is 0 (zero).

Overview of the SCUTMA Plugin

The SCUTMA approach has three main processes: (1) Run static analysis, where it extracts the CFM and CS metrics; (2) run dynamic, market and business analysis, where it extracts the metrics FC, FR, MV and BV; and (3) components selection, where the genetic algorithm is performed to select the components to be tested. These and other processes are shown in Fig. 2 and explained in the next subsections. The SCUTMA plugin was implemented to be used on the Android Studio tool, which is an integrated development environment to develop mobile apps for Android platform.

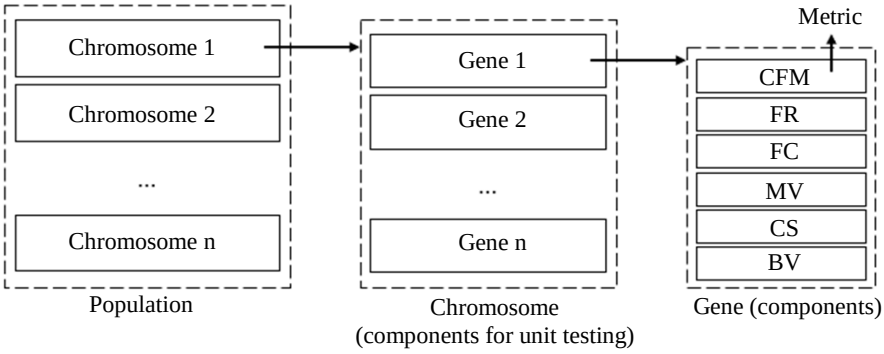

Fig. 1: Genetic representation

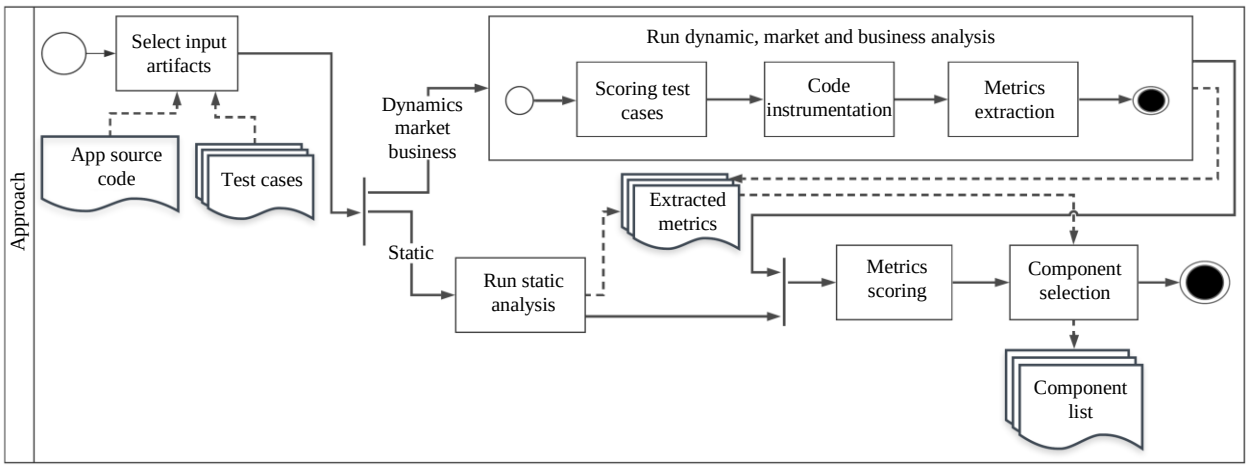

Fig. 2: Overview Approach SCUTMA

Android Studio was chosen because it is the official development environment of Android platform. Technologies used for development of the SCUTMA plugin are the following:

- Intellij IDEA[5]: A development interface that allows one to create plugins fully compatible with Android Studio
- Java Language: The Java language was chosen for the SCUTMA plugin development, this is an interpreted and object-oriented programming language
- JHawk[6]: This tool collects metrics on four different levels, including method, class, package and system. It was used to aid in the extraction of static metrics
- Comment Remover[7]: It is a source code commenting library for Java TM 7 and above. It also supports JavaScript, HTML, CSS, Properties, JSP and XML comments. This library was used to remove comments from Java and XML files
- Android Debug Bridge (ADB)[8]: It is a versatile command line tool that allows one to communicate with an emulator instance or with a connected Android device. It was used to get the ID of the connected devices and to copy the file in the ".trace" format of the device to the computer
- Spoon[9]: This tool runs a test script on multiple devices simultaneously and displays the results in a meaningful way. It was used to run the system test scripts
- TraceView[10]: It is a tool that provides graphical representations of trace logs. It was used to convert the log file from ".trace" format to ".csv"
- Framework jMetal[11]: It was chosen to assist in the implementation of the genetic algorithm, the Pareto curve is generated by the jMetal framework. It has

[5]https://www.jetbrains.com/pt-br/idea/
[6]http://www.virtualmachinery.com/jhawkprod.htm
[7]https://github.com/ertugrulcetin/CommentRemover/blob/master/Readme.md
[8]https://developer.android.com/studio/command-line/adb.html
[9]http://square.github.io/spoon/
[10]https://developer.android.com/studio/profile/traceview
[11]http://jmetal.sourceforge.net/

open source, is based on Java for multiobjective optimization with metaheuristics (Nebro *et al*., 2015)

Each step that composes the processes presented in Fig. 2 will be detailed below, showing some screens of the SCUTMA plugin. The SCUTMA plugin is available at "https://bit.ly/2TKxsPG".

Select Input Artifacts

As input artifacts, SCUTMA has two components: (1) Source code consisting of a mobile app source code for the Android platform, already available in the Android Studio environment; and (2) test cases that are system tests designed for the mobile app and written in the Java language and that use the Espresso[12] framework. In Fig. 3 we can see the Budget Watch mobile app opened in Android Studio and the access menu for the SCUTMA plugin on the right side of "Help".

After clicking the SCUTMA option, the main screen is displayed, shown in Fig. 4. The initial screen has 3 fields. The Source Code Folder field is populated automatically with the source code path of the application opened in Android Studio and cannot be changed. The SDK Folder field is also filled out automatically with the SDK folder's default path and can be changed if it is incorrect. The Choose Type Of Metrics field allows one to choose whether to use static and/or dynamic metrics in component selection.

Run Static Analysis

This process allows the extraction of two metrics: CFM and CS:

- Cost of Future Maintenance (CFM): The extraction of this metric was done with the aid of the JHawk tool, that is used for the extraction of Halstead metrics
- Code Smell (CS): The code smells were also extracted using the JHawk tool. For each code smell detected in a component, 1 point will be added to the component. Then, the number of code smells detected for each component will be added and, after that, Equation 2 will be applied

Run Dynamic, Market and Business Analysis

This process is divided in three steps: test case scoring, code instrumentation and metrics extraction. These steps are described in the following subsections.

Scoring Test Cases

The user will report a weight on an ordinal scale from 0 to 5 for the importance of each test case, where 0 means the test case will not run, 1 is least and 5 is very important. Test cases represent the system's use cases. For example, on the Android project (Fig. 3), the system test cases are in the directory /app/src/androidTest/. These test cases are shown in Fig. 5, where the user would score the test cases of the Budget Watch mobile app.

Code Instrumentation

In the instrumentation step, the plugin modifies the main activity of the application. First, it identifies such activity from the application's AndroidManifest.xml file. Then, the Comment Remover tool removes all comments from this file and the main activity to avoid errors in instrumentation. Finally, the plugin modifies the application to insert the startMethodTracing commands into the OnCreate method and stopMethodTracing into the OnDestroy method of the mobile app's main Activity. These two methods entered as commands are contained in the android.os.Debug class and are responsible for starting and stopping the profile collection respectively. This is to know what is happening while running the mobile app.

Metrics Extraction

For the extraction of the dynamic, market and business metrics, it is necessary to execute the test cases, which is performed with the aid of the Spoon tool, because it manages the simultaneous execution of a test case across multiple devices, as well as logs the runtime and whether the test case has passed or failed for each device.

After each test case run, a tracking file in the ".trace" format will be generated on each device. It provides detailed metrics about a component, such as the number of calls, execution time and time spent running the component. Such file will be copied to the computer through the ADB tool. After that, it will be transformed into the ".csv" format by the trace view tool provided with the Android SDK.

The information generated by the Spoon tool and the tracking file in the ".csv" format will be used to extract the metrics: Call frequency, risk of failure, market vulnerability and business value:

- Call Frequency (FC): The number of calls will be needed to compute the call frequency metric
- Risk of Failure (RF): The data needed to calculate this metric comes from recording the execution of a component in test cases that passed or failed. Thus, the plugin analyzes the result of the Spoon tool and the tracking file. As a result of this analysis, a set of data will be generated from which test cases failed and passed for each component
- Market Vulnerability (MV): The market vulnerability metric is used to represent the percentage of the market in which a component is vulnerable. For instance, some devices with the following API configurations 19, 21, 22 and 23 have, respectively, 20.8, 9.4, 23.15 and 31.3% average market vulnerability and the edit component

[12]https://developer.android.com/training/testing/espresso

failed on devices with APIs 22 and 23 this component had a 54.4% market vulnerability

- Business Value (BV): The plugin analyzes the result of the Spoon tool and the tracking file and generates a data set of which test cases performed each component. When a test case runs a component, the weight of that test case is added to the component

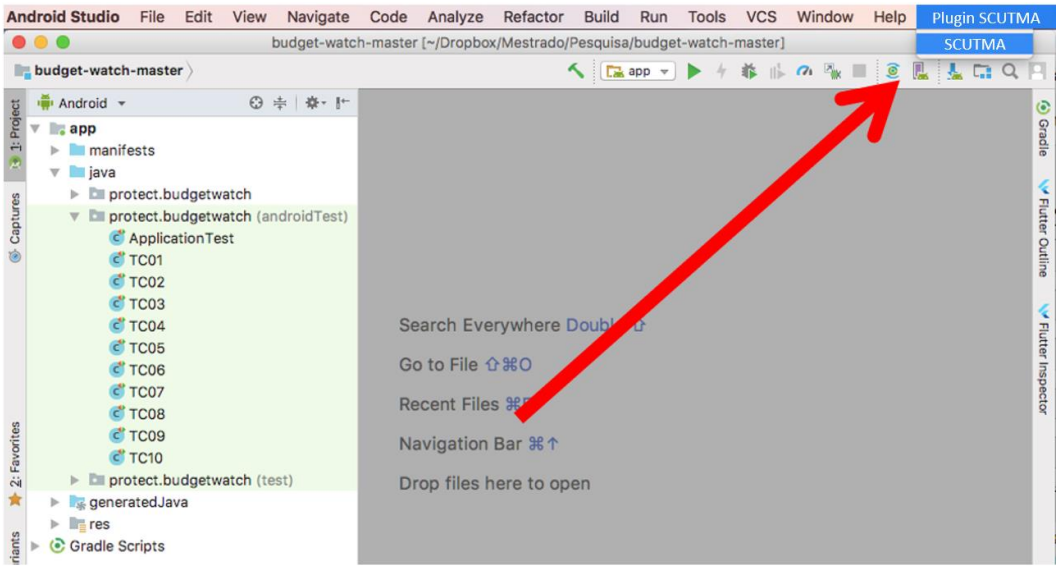

Fig. 3: SCUTMA plugin menu

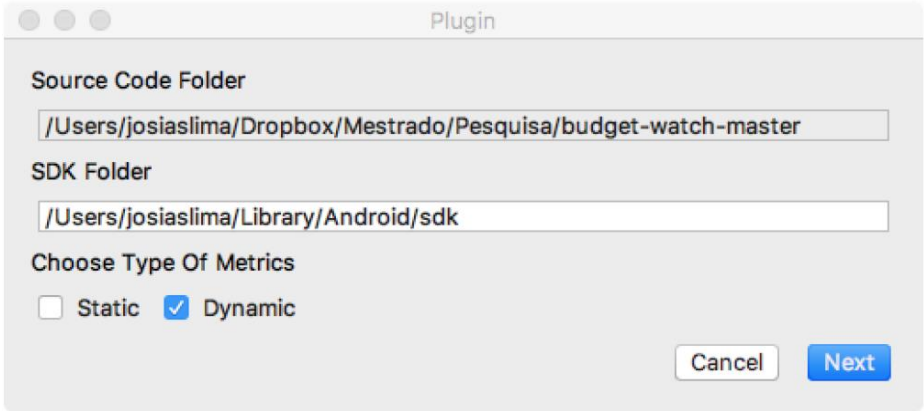

Fig. 4: SCUTMA plugin home screen

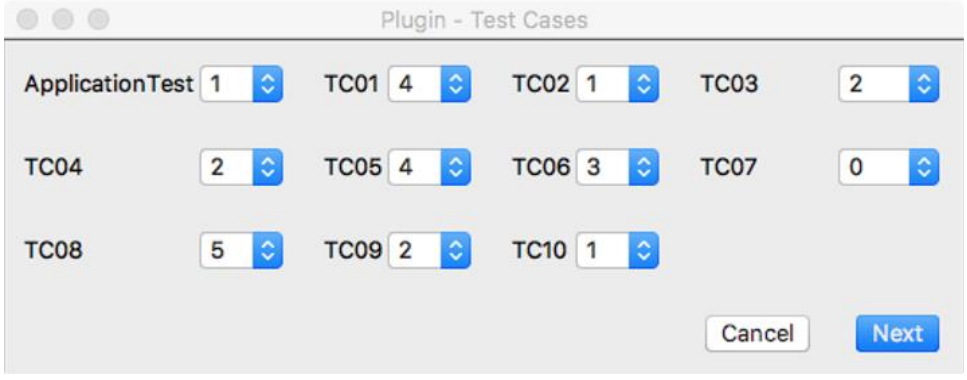

Fig. 5: Screen mobile app test cases

Metrics Scoring

This step allows the user to select a usage percentage for each metric with one of the following percentages 0; 0.2; 0.4; 0.6; 0.8 and 1.0, where 0 means that 0% of the metric will be used, 0.2 will use 20% of the metric value and so on. In Fig. 6 the SCUTMA screen is shown where the user chooses the percentage of use of each metric.

Component Selection

The metrics used correspond to the objective functions of the NSGA-II; because in the design of these metrics it was planned that they would be part of the objective functions. For this reason, all metrics are normalized in the interval [0;1] to facilitate the combination of the objectives in the NSGA-II. An algorithm combines the results of the extraction of metrics into a single file that serves as input to the genetic algorithm. The NSGA-II implementation was developed in the Java language with the aid of the jMetal framework.

The NSGA-II algorithm implements the concept of dominance, that is, the Total Population is classified into borders according to the degree of dominance. Individuals at the first border are considered the best, while individuals at the last border are the worst. Then one can find better solutions (points closer to the Pareto region) (Zitzler *et al*., 2004). Figure 6 shows the screen where the user informs the parameters necessary to execute the genetic algorithm. After execution, the list of components of the optimal Pareto solution will be presented to the user in the format: Id, package name, class name, method name (component), parameters and return type, as shown in Fig. 7.

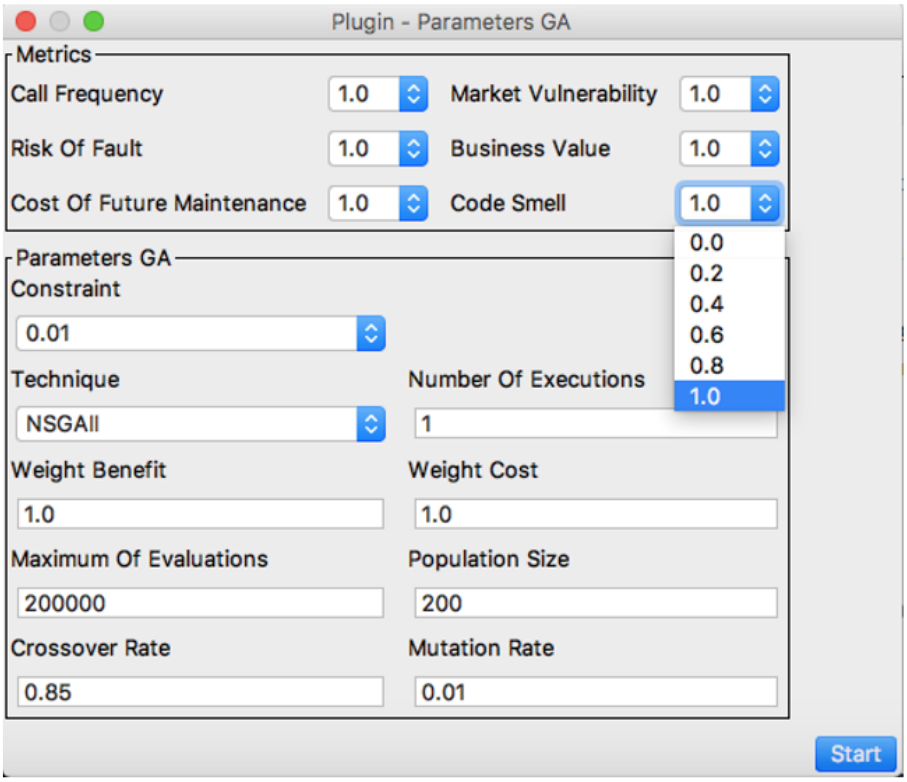

Fig. 6: Parameters of genetic algorithm

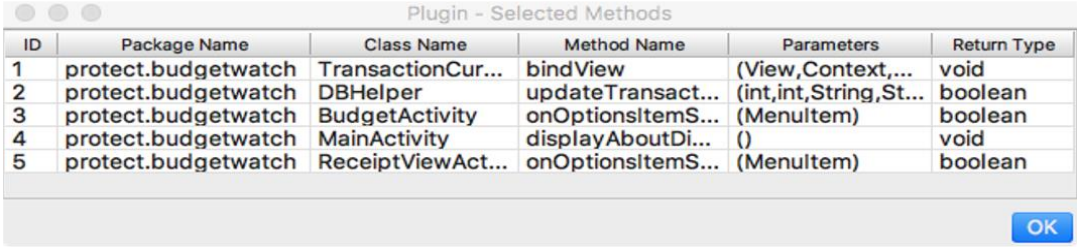

Fig. 7: Screen showing selected components

Limitations of the SCUTMA Plugin

The SCUTMA plugin works on Windows 7, Windows 10 and macOS Catalina 10.15.4 operating systems. It is compatible with Android Studio version 2.2.3 up to version 3.5 (current). Another limitation is the Android application source code to be in the Java language, due to the Jhawk tool, only analyzing code in that language. To use dynamic metrics, market and business, the application needs to have system test scripts, being a test case per Java file.

This section has presented the details of the SCUTMA approach, the design process and the technologies used for the implementation. The next section presents the first study carried out with the purpose of analyzing the correlation among the metrics that are part of the multiobjective solution.

First study: Metrics Correlation

This study was designed and performed as a controlled experiment based on (Runeson and Höst, 2009) on how to design, plan and conduct case studies, surveys, experiments or controlled experiments and action research.

RQ1: Is there a Correlation Among the Metrics?

The answer to this question will help us decide which set of metrics can be used in combination.

Considering the metrics: CFM, CS, FC, FR, MV and BV it is necessary to investigate if there is a correlation among them, aiming to identify a subset of metrics to support the selection of components for unit testing in mobile apps. Table 4 shows a summary of these metrics, including symbol, description and equation.

Subjects Selection

The subjects selection had two phases; mobile apps selection and mobile devices selection. Selection of mobile apps was based on four parameters: Popularity (number of downloads in the Google Play store), diversity (different categories of mobile apps), self-containment (no more configurations to be implemented) and UI test scripts. According to (Kochhar *et al*., 2015) finding automated test script is not a trivial task since the authors conducted an empirical investigation regarding techniques used to test mobile apps, frameworks used and types of testing used into open source mobile apps and the concluded Android mobile apps are not properly tested since around 86% of the investigated mobile apps did not contain any test cases.

The first three parameters were based on (Fazzini *et al*., 2017); and the last one was based on the need to run the UI scripts to extract dynamic metrics. Table 5 shows the pre-selected mobile apps. In order to identify which mobile app could be used for our study we use two mobile apps with more UI tests. Our objective was to run mobile apps for identifying the correlation among metrics.

Selection of mobile devices was based on the screen size, resolution and the Android SDK versions most used by mobile apps published in the Google Play Store (data updated in October 2018). In this study, we used mobile devices with different versions of Android OS, models, screen sizes and densities with the purpose of identifying possible failures in mobile devices with great representation in the market. The information on which devices certain tests have failed is important to obtain the MV metric. Table 6 shows the market share of the most commonly used API versions and the model and number of devices that would be used for each API version. Based on this information and according to (Vilkomir *et al*., 2015), the use of 13 devices is sufficient for fault-finding. We defined 13 devices that would be used in the study, listed in Table 6.

Table 4: Summary of metrics

Symbol	Description	Equation
CFM	Cost of Future Maintenance	$cfmi = Ei * Bi$
CS	Code Smell	$csi = \frac{Si}{5}$
FC	Frequency of Call	-
FR	Fault Risk	$rfi = \frac{Pi}{(Pi + Fi)}$
MV	Market Vulnerability	-
BV	Business Value	-

Table 5: Mobile apps used in this study

ID	App	Version	Classification	Downloads	Code (LOC)	Components	UI Tests	Category
1	Bee Count (BC)	-	4.5/5	10-50 k	3.8k	13/230	15	Productivity
2	Budget Watch (BW)	0.7	5.0/5	1-2 k	946	15/141	15	Finance
3	Counter (C)		4.5/5	100-500 k	3 k	37/96	15	Tool
4	Money Balance (MB)	-	-	-	2.9 k	14/257	15	Finance
5	Pocket Code (PC)	1.1.15	3.8/5	100-500 k	83 k	496/7610	420	Education
6	Pocket Paint (PP)	0.9.28	3.9/5	100-500 k	11 k	239/916	33	Tool
7	Recurrence (R)	1.5	4.5/5	10-50 k	2.4 k	27/194	15	Productivity
8	Simple Draw (SD)	-	4.5/5	50-100 k	569	14/55	15	Tool
9	Simple Flashlight (SF)	-	4.4/5	10-50 k	1 k	33/110	15	Tool
10	Simple Notes (SN)	-	4.5/5	10-50 k	875	27/103	15	Tool

Legend: ID = a number to identify the mobile app; Version = mobile app version; App = mobile app name; Classification = based on users rating on the Google Play Store; Downloads = number of downloads in the Google Play Store; Code (LOC) = number of lines of mobile app code; Components = number of components the tests called/number of all components in the mobile app; UI Tests = number of User Interface (UI) tests; Category = category to which the mobile app belongs

Table 6: Mobile devices used in this study

ID	Version	Version name	API	Percentage	Model	Size	Density
D1	4.4.2	KitKat	19	7.6	BLU	480×800	230
D2					LG E977	768×1280	320
D3	5.0		21	3.5	Galaxy Note 3 LTE	1080×1920	480
D4					Alcatel PIXI4 (4.0)	320×480	160
D5	5.1	Lollipop	22	14.4	Alcatel PIXI4 (6.0)	1280×720	320
D6					Galaxy J1 Mini	480×800	240
D7					ASUS Zenfone Go LTE	720×1280	320
D8					Galaxy A9	1080×1920	420
D9	6.0	Marshmallow	23	21.3	LG X Power	720×1280	320
D10					Moto E	960×540	240
D11					Moto Z power edition	1440×2560	640
D12	7.0	Nougat	24	18.1	Galaxy A5	720×1280	294
D13					Moto G 5	1080×1920	480

Study Execution

SCUTMA plugin was used for each mobile app. A value between 1 and 5 was randomly selected for each test script. Therefore, the test scripts were run at the same time on all the mobile devices used in this study. Subsequently, the extraction of the value of the metrics for each mobile app was carried out.

Results and Discussion

The static, dynamic, market and business metrics of two mobile apps were extracted and the test scripts of each mobile app were executed on 13 devices. The value of these metrics for each component per mobile app is available at ”https://goo.gl/NwM3je”.

The SPSS[13] program was used to support the verification of the existence or not of correlation among the metrics, in order to generate the Pearson correlation coefficient. This coefficient is adequate when the data are quantitative and have normal distribution. Table 7 presents the value generated by the Pearson correlation coefficient; it was extracted from (Rumsey, 2009).

For this paper the correlations will be interpreted as follows: The metrics could be used together when the correlation is no linear relationship, weak or moderate; and the metrics cannot be used together when the correlation is strong or perfect.

Pocket Code Mobile App

Dynamic, market and business metrics depend on the outcome of system test scripts. In the result of the execution, of the 420 test scripts of the Pocket Code mobile app by the SCUTMA plugin, it was observed that 251 (60%) of the test scripts failed on at least one device and 169 (40%) of the scripts passed on all the devices.

Table 8 shows the number of test scripts that failed per device. Analyzing this data, we can see that most test scripts have failed on almost every device since the median percentage of faults per device is 72.51% of all test scripts that failed.

[13]https://www.ibm.com/analytics/us/en/technology/spss/

Table 7: Interpreting the correlation value

Value	Correlation
0	No linear relationship
+0.3 or -0.3	Weak
+0.5 or -0.5	Moderate
+0.7 or -0.7	Strong
+1 or -1	Perfect

Table 8: Quantity of faults by device pocket code.

ID device	Total faults	Percentage faults
D1	184.00	73.31
D2	192.00	76.49
D3	180.00	71.71
D4	196.00	78.09
D5	177.00	70.52
D6	184.00	73.31
D7	182.00	72.51
D8	173.00	68.92
D9	179.00	71.31
D10	185.00	73.71
D11	176.00	70.12
D12	184.00	73.31
D13	178.00	70.92
Average	182.31	72.63
Median	182.00	72.51
Standard deviation	6.37	2.54

Table 9 shows the value of the Pearson correlation coefficient for the Pocket Code mobile app. The correlation between the Fault Risk (FR) and Market Vulnerability (MV) metrics is likely strong due to the 161 (64.14%) test scripts that failed on all devices and the median failure of the test scripts per device to be 72.51% making the Market Vulnerability (MV) have a value close to 1 (one) for most components, since the value of this metric depends on the devices in which the failure occurred. The Fault Risk (FR) metric was also close to 1 (one) for each component, making the correlation between the two metrics strong.

However, if the tests fail with greater variability among the devices, the value of the Market Vulnerability (MV) metric will have a greater variation, causing it to change the correlation value between the two metrics.

Table 9: Pearson correlation pocket code

	CFM	CS	BV	FC	MV F	R
CFM	1					
CS	0.19a	1				
BV	0.009	-0.006	1			
FC	0	0.024b	0.238a	1		
MV	-0.001	0	0.528a	0.122a	1	
FR	-0.001	0.002	0.473a	0.119a	0.987a	1

Legend: a. The correlation is significant at the 0.01 level (2 extremities); b. The correlation is significant at the 0.05 level (2 extremities)

Table 10: Quantity of faults by device pocket paint

ID device	Total faults	Percentage faults
D1	22.00	81.48
D2	20.00	74.07
D3	22.00	81.48
D4	22.00	81.48
D5	20.00	74.07
D6	19.00	70.37
D7	20.00	74.07
D8	19.00	70.37
D9	19.00	70.37
D10	19.00	70.37
D11	24.00	88.89
D12	19.00	70.37
D13	20.00	74.07
Average	20.38	75.50
Median	20.00	74.07
Standard deviation	1.61	5.96

Table 11: Pearson correlation pocket paint.

	CFM	CS	BV	FC	MV	FR
CFM	1					
CS	0.324a	1				
BV	-0.024	-0.032	1			
FC	-0.009	0.006	0.362a	1		
MV	-0.031	-0.041	0.732a	0.266a	1	
FR	-0.028	-0.053	0.413a	0.159a	0.882a	1

Legend: a. The correlation is significant at the 0.01 level (2 extremities).

Pocket Paint Mobile App

In the result of the execution of the 33 system test scripts of the Pocket Paint mobile app by the SCUTMA plugin, it turns out that 27 (82%) test scripts failed on at least one device and 6 (18%) test scripts passed on all devices.

The number of test scripts that failed per device for the Pocket Paint mobile app is shown in Table 10. In this mobile app, the median percentage of device faults was 74.07%. This means that most test scripts failed on almost every device. Since 17 (62.96%) test scripts failed on all devices.

We consider the large number of faults in both the Pocket Code mobile app and Pocket Paint due to the following reasons. First of all, it is difficult to make a script that works on different devices because even if all the required user interface elements are on screen, layouts can still differ based on OS version, screen size and orientation (Samuel and Pfahl, 2016). Second, we did not have the information on which devices were used to create test scripts. Finally, we need to measure MV and BV metrics, therefore we could not run the test scripts previously.

Table 11 shows the value of the Pearson correlation coefficient for the Pocket Paint mobile app. The results show that there is a strong correlation between the Market Vulnerability (MV) and Business Value (BV) metrics. However, the Business Value (BV) of each component depends on the weight given to each test script. Therefore, the correlation may change according to the weight given to each test script.

The explanation for the strong correlation between the Market Vulnerability (MV) and Fault Risk (FR) metrics is the same as that reported in the Pocket Code mobile app, because here as well, there is a slight variation in the devices where the test scripts failed.

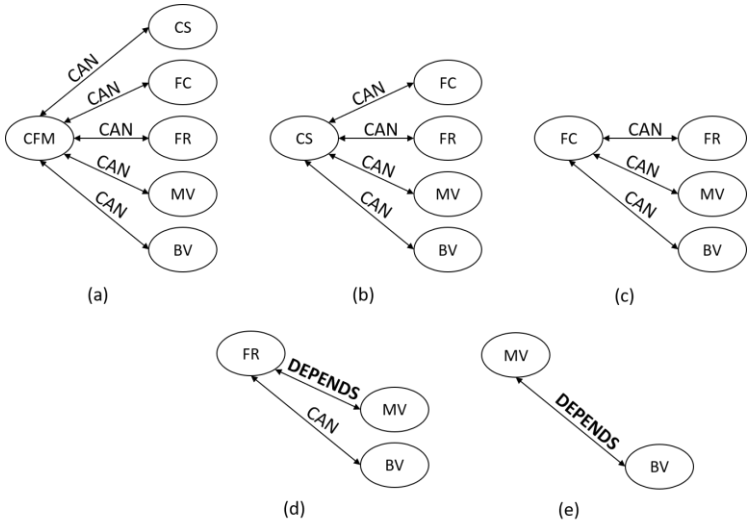

Fig. 8: Metrics that can be used together

Discussion

Figure 8 shows which metrics can be used together. We can observe that the CFM and CS static metrics can be used together, since the result of the correlation between these two metrics was weak (0.19) and moderate (0.324). The CFM and CS metrics can be used together with the metrics FC, FR, MV and BV, as there is a weak correlation among them. Probably this weak correlation is due to the CFM and CS metrics being extracted statically and the metrics FC, FR, MV and BV require execution of the mobile app system test cases.

The CFM, CS, FC, FR, MV and BV metrics can be used together in the same objective function, as observed in Fig. 8. However, as a suggestion, we could recommend that the tester chooses to use only one of the FR or MV metric for component selection, in which the test cases still depend on the compatibility problem among different devices, because in the results of this study, the correlation between these two metrics was strong (0.987 and 0.882) and the test cases still depended on the compatibility problem among different devices.

The BV metric can be used together with the MV metric. However, in some cases depending on the user's weight for each test cases, the correlation between these two metrics should vary between weak and strong. Because in Pocket Code mobile app was moderate (0.528) and in Pocket Paint mobile app was strong (0.732).

Threats to Validity

This section discusses different threats to the validity of our experiment. The instrumentation could be a threat, since a change in the mobile apps' source code was made to include support for profile collection of each mobile app, because it was not possible to collect this data without the instrumentation. However, this instrumentation does not influence the operation of the mobile app. In order to reduce this threat, the choice of mobile apps was not done randomly, since they were selected based on the availability of their source codes and test scripts.

To reduce the threat of results generalization, two mobile apps of different categories were used. However, the study may not be representative for other mobile app categories. The experimental environment can be a threat. Our environment was academic, but the computational infrastructure used (devices) represents the same as in industry by testers.

Taking into account the fact that the results of the previous tests run could affect the next test executions, after executing each test script, the SCUTMA plugin performs a test procedure to uninstall and clear the App Under Test (AUT). Also to reduce the threat of confidence in measurements, the weight of each use case was assigned randomly in order to simulate the options that the user would have.

The elements outside the experimental environment could influence the results, for example, a message or an incoming call during the test case execution. Therefore, the mobile devices were placed in airplane mode, since the AUTs do not need an internet connection to work. Another configuration on mobile devices is to configure the keep awake option so that the screen does not enter the standby mode while the test is running. In addition to the first study, we execute a feasibility study of the SCUTMA approach; this study will be present at the next section.

Second Study: Feasibility

This study aims to analyze the effectiveness of the plugin in selecting components for unit testing in mobile apps on the Android platform. We followed the guidelines of (Runeson and Höst, 2009). For this study we measured

the efficiency with the number of components with error selected. We define a research question as follow:

RQ2: What is the effectiveness of the SCUTMA plugin in component selection in relation to the selection made manually by a specialist with regard to the number of components with selected errors?

Answering this question will help us to understand which usage scenarios are recommended to use the plugin.

In the next subsections of this study, it will be explained: (1) How the components with error were identified, (2) which scenarios will be considered in the assessment, (3) how the selection of participants was carried out, (4) which materials were used, (5) how the study was carried out and (6) what results were found. In addition, threats to the validity of this study will be discussed.

Identification of Components with Error

The identification of the components with error will occur through the analysis of the error correction commits. To identify these commits, a manual search of commits messages will be performed using the keywords fix, problem, incorrect, correct, conform (Mockus and Votta, 2000). To mitigate the threats to the validity of this study, only the commits of the mobile app version used in the experiments will be analyzed.

We identify some specialists to select components manually, following the next steps: (1) Each specialist will receive a document explaining the main features and features of the mobile app. Next, (2) the specialist will select the components for unit testing and will describe which criteria were used for the selection. Then, (3) once the lists of manually selected components are obtained for each mobile app, the researcher will evaluate the effectiveness of the selection of each specialist. Finally, (4) we will compare the selection of specialists of each mobile app with the selection of the SCUTMA plugin.

Evaluated Scenarios

In order to compare the selection of specialists with the selection of the SCUTMA plugin, 63 different prioritization scenarios were constructed to simulate the broad spectrum of diverse realities present in the software industry. The scenarios are based on six criteria: Cost of Future Maintenance (CFM), Defect Propensity (PD), Frequency of Call (FR), Fault Risk (FR), Market Vulnerability (MV) and Business Value (BV).

In the first scenario (C01), the priority is to select components with high cost rate of future maintenance for unit tests. In the second (C02), components with a high degree of readiness to defect (code smells) are selected. This rule is followed up to the sixth scenario (C06), according to the other criteria. From the seventh scenario (C07) onwards, an arrangement is generated that incorporates all the combinations of criteria among them, as presented in Table 12.

For example, in the thirteenth scenario (C13), the components prioritized for selection are defined as those with a high defect propensity rate and risk of failure. The normalized value of the considered criteria was used to construct the scenarios.

Due to the random nature of the evolutionary approaches, each scenario was executed 30 times with the NSGA-II evolution algorithm and the mean among the best results was used for comparison according to the parameters usually cited in the technical literature for evolutionary algorithms.

The data that were used to execute the algorithm were:

- Population size: 200
- Maximum number of evaluations: 200.000
- Crossover rate: 0.85
- Mutation rate: 0.01

Table 12: Component selection prioritization scenarios

ID	Criterions	ID	Criterions	ID	Criterions	ID	Criterions
C01	CFM	C17	FC x MV	C33	PD x FC x MV	C49	CFM x FC x FR x BV
C02	PD	C18	FC x BV	C34	PD x FC x BV	C50	CFM x FC x MV x BV
C03	FC	C19	FR x MV	C35	PD x FR x MV	C51	CFM x FR x MV x BV
C04	FR	C20	FR x BV	C36	PD x FR x BV	C52	PD x FC x FR x MV
C05	MV	C21	MV x BV	C37	PD x MV x BV	C53	PD x FC x FR x BV
C06	BV	C22	CFM x PD x FC	C38	FC x FR x MV	C54	PD x FC x MV x BV
C07	CFM x PD	C23	CFM x PD x FR	C39	FC x FR x BV	C55	PD x FR x MV x BV
C08	CFM x FC	C24	CFM x PD x MV	C40	FC x MV x BV	C56	FC x FR x MV x BV
C09	CFM x FR	C25	CFM x PD x BV	C41	FR x MV x BV	C57	CFM x PD x FC x FR x MV
C10	CFM x MV	C26	CFM x FC x FR	C42	CFM x PD x FC x FR	C58	CFM x PD x FC x FR x BV
C11	CFM x BV	C27	CFM x FC x MV	C43	CFM x PD x FC x MV	C59	CFM x PD x FC x MV x BV
C12	PD x FC	C28	CFM x FC x BV	C44	CFM x PD x FC x BV	C60	CFM x PD x FR x MV x BV
C13	PD x FR	C29	CFM x FR x MV	C45	CFM x PD x FR x MV	C61	CFM x FC x FR x MV x BV
C14	PD x MV	C30	CFM x FR x BV	C46	CFM x PD x FR x BV	C62	PD x FC x FR x MV x BV
C15	PD x BV	C31	CFM x MV x BV	C47	CFM x PD x MV x BV	C63	CFM x PD x FC x FR x MV x BV
C16	FC x FR	C32	PD x FC x FR	C48	CFM x FC x FR x MV		

Equation 6 was used in the results analysis, where it is the relation between the components with error and the total number of components:

$$f(x) = \frac{TCF}{TC} \quad (6)$$

where, x is the solution found in the Pareto curve; *TCE* is the total of components with error that has in solution x; and *TC* is the total of components you have in the solution x.

Participant Selection

To participate in this study, industry and academia professionals with in-app unit testing experience on the Android platform were invited. The developer/researcher base was obtained from a Linkedin social network search, as well as the Android Dev BR community. To participate in the study, the professionals had to express interest in participating in the study, agreeing to the Informed Consent Form (TCLE) and filling out a characterization form. This was done with the objective of having the knowledge of the degree of experience of each professional and thus directing the selection to the next phase of the experiment. The tools are available at "https://goo.gl/71TvEQ".

The characterization form was sent to more than 100 professionals. Of this total, 31 participants moved on to the next phase of group selection. The 31 participants were divided into two groups, Group A and Group B, taking into account the information that was filled out on the characterization form, on development experience and unit test experience time on Android mobile apps.

Of the total number of participants, only 7 completed the study. Of these, 2 of them have 2 years of experience (29%); 2 of them have 3 years of experience (29%); 3 of them have 1, 4 and 5 years of experience, respectively and each represents (14%).

Materials

Because the study involved the manual and automated selection of components, it was necessary to use mobile apps with UI tests. Previously we analyzed and executed UI tests of mobile apps in Table 5. We selected two mobile apps with UI tests more compatible with different mobile devices. Table 13 shows the description of some features of mobile apps used for this study. More detailed information about these mobile apps is available at "https://goo.gl/71TvEQ". For this study, we have used the same devices as used in our first study (Table 6).

Study Execution

The feasibility study was conducted online[14] and executed in two parts. In the first part, participants were divided into groups A and B, then emailed a description of the characteristics of the mobile app for each participant. Group A got the Recurrence mobile app and Group B with the Budget Watch mobile app. For each participant they were asked to perform the selection of components for unit test writing and to send the list of selected components by e-mail.

In the second part, for the study execution process, each mobile app was opened by the SCUTMA plugin in Android Studio, a value between 1 and 5 was randomly selected for each test script, so the test scripts were run at the same time in all the mobile devices used in the experiment. Subsequently, the value of the metrics for the mobile app was started. Finally, in the SCUTMA plugin, components for the unit test writing were selected and 63 different selections were made using the 63 scenarios to select the components. The list of components selected from each scenario was compared to the list assembled from the selection made by the specialists.

The following will present the results of this study for each of the mobile apps, starting with the Budget Watch mobile app and then the result for the Recurrence mobile app. First, the list of components with error will be shown, followed by the evaluation of the effectiveness of the manual selection performed by the specialists, continuing with the evaluation of the effectiveness of the automated selection performed by the SCUTMA plugin, finally the comparison of the manual selection with the automated selection.

Mobile App Budget Watch Results

Table 14 shows the list of Components that were identified with Error (CE) for the Budget Watch mobile app. This list of components will be used to evaluate the selection of specialists and the SCUTMA plugin.

Specialists Evaluation

The first Specialist (S1) selected eight components using the criterion "Public methods, which are possible to be tested, methods that are used within functionalities", which represents 5.67% of the total components of the mobile app. Of these, one of them (ID CE9) is in the list of twelve components with known errors, presented in Table 14, representing 8.33%.

The second Specialist (S2) selected sixteen components using the criterion "Classes dealing with lifecycle methods", which represent 11.35% of the total components of the mobile app. Of these, eight (Ids CE1, CE2, CE3, CE4, CE6, CE8, CE11 and CE12) are in the list of twelve components with known errors Table 14, representing 66.67%. Figure 9 presents the graph that shows the coverage percentage of each of the two specialists for the components that were identified with errors in the Budget Watch mobile app.

[14]The study was conducted with specialists from the industry and academia (target audience).

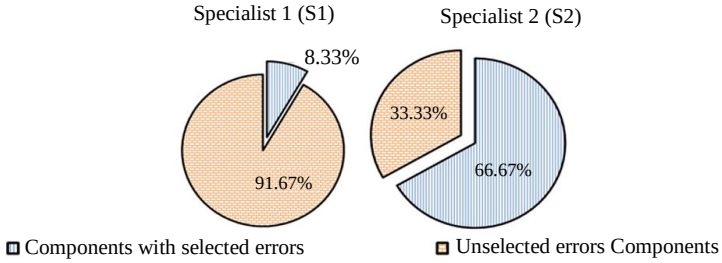

Fig. 9: Coverage of components with errors by specialists for the Budget Watch mobile app

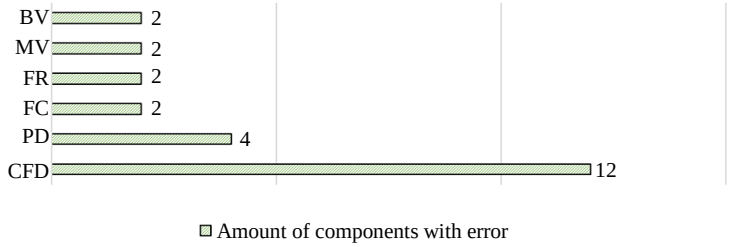

Fig. 10: Number of components with error that could be selected by each metric for the Budget Watch mobile app

Table 13: Mobile app descriptions

ID	App	Version	Code (LOC)	Components	Tests	Category
1	Budget Watch (BW)	0.7	946	36/141	15	Finances
2	Recurrence (R)	1.5	2.4 k	27/194	15	Productivity

Legend: ID = a number to identify the mobile app; App = mobile app name; Version = mobile app version; Code (LOC) = number of lines of code in the mobile app; Components = number of components the tests called/total number of mobile app components; Tests = number of mobile app UI tests; Category = category to which the mobile app belongs

Table 14: List of components with errors in the Budget Watch mobile app

ID	Class	Component signature
CE1	BudgetActivity	protected void onCreate(Bundle savedInstanceState)
CE2	BudgetActivity	public boolean onContextItemSelected(MenuItem item))
CE3	BudgetActivity	public void onResume()
CE4	BudgetActivity	protected void onDestroy()
CE5	BudgetViewActivity (R)	public boolean onOptionsItemSelected(MenuItem item)
CE6	BudgetViewActivity (R)	public void onResume()
CE7	BudgetViewActivity (R)	protected void onDestroy()
CE8	BudgetViewActivity (R)	protected void onCreate(Bundle savedInstanceState)
CE9	CsvDatabaseExporter (R)	public void exportData(DBHelper db, OutputStreamWriter output)
CE10	DatabaseCleanupTask (R)	protected void doInBackground(Void... nothing)
CE11	MainActivity (R)	protected void onCreate(Bundle savedInstanceState)
CE12	MainActivity (R)	private void displayAboutDialog()

SCUTMA Plugin Evaluation

For a better understanding of the coverage of components with errors by the SCUTMA plugin, Fig. 10 shows the number of components with error (presented in Table 14) that could be selected for each metric used in the 63 scenarios.

The two components (IDs CE1 and CE3 from Table 14) were executed during the execution of the test cases. Therefore, they are more likely to be selected in scenarios using the metrics BV, MV, FR and FC. The four components with CE2, CE6, CE9 and CE12 IDs of Table 14 are more likely to be selected in scenarios using the PD metric, since code fragments have been identified in those components. Finally, all components of the mobile app, including the twelve faulty components in Table 14, can be selected in scenarios that use the CFM metric, as all components have this metric.

Components with Error

Table 15 presents the coverage of components with error of the best and worst of the 63 scenarios for the Budget Watch mobile app. It is observed that the solutions of each of the scenarios managed to find at most 10 components (83.33%) with error. It is worth noting that each of the scenarios obtained a different

number of solutions at the Pareto frontier. The Pareto frontier was generated from the 30 runs of the genetic algorithm for each of the scenarios and only the Pareto frontier solutions were analyzed. The complete table with the result of the 63 scenarios can be accessed at "https://goo.gl/92kQEf".

Still analyzing Table 15, it can be seen that the scenario 1 (C01) obtained solution with the highest value for the relative number of components in error divided by the total number of components (Equation 6) when return from 1 to 8 parts with error, this is due to the fact that all components have a Cost of Future Maintenance (CFM).

The C02 scenario using the default Propensity (PD) metric selected four components with error and these components had identified code smells (Fig. 10). The four scenarios C03, C04, C05 and C06 using the Frequency of Call (FC), Fault Risk (FR), Market Vulnerability (MV) and Business Value (BV) metrics respectively selected two components with error, which were performed by the test cases (Fig. 10).

With regard to the best case, scenarios C51 and C50 can be cited, as these were the best and second best scenario for Equation 6, respectively. The best solution for the C50 scenario obtained 21.27% of the total components of the mobile app and with this number of components this solution covered 75% of the components with errors identified and got 70% more components. For scenario C51, the best solution was 26.95% of the total components of the mobile app, covering 83.33% of the components with errors identified and over 73.68% of components. This indicates that in the best case SCUTMA is achieving good coverage of the components with error, however, it needs a refinement to decrease the number of additional components that make up the solution.

As far as the worst case is concerned, scenario C42 can be cited because it had the lowest value for Equation 6. The worst solution for scenario C42 obtained 21.98% of the total components of the mobile app, with that number of components, this solution covered 8.33% of the components with errors identified and was with 96.77% of components.

Components with General Error

Figure 11 shows the percentage of solutions for each number of error components identified in the Pareto frontier solutions. It is observed that most Pareto frontier solutions identified three components with error and only 1.43% of the solutions identified zero components with error. This indicates that SCUTMA has good error coverage. However, Fig. 12 shows that 84.61% of the components in the solutions did not contain errors. For this reason, it is necessary to refine the plugin to reduce the number of components that do not contain errors (false positives), directing to components that have real errors, in real mobile apps.

Table 15: List of components with errors in the Budget Watch mobile app

		Components with error												
		0	1	2	3	4	5	6	7	8	9	10	11	12
C1	Quantity	489	619	500	578	282	198	570	264	69	7	0	0	0
	Higher f(x)%	0.00	100.00	100.00	75.00	66.67	62.50	46.15	53.85	34.78	25.71	0.00	0.00	0.00
	Lower f(x)%	0.00	12.50	20.00	20.00	25.00	25.00	13.95	14.58	15.69	17.65	0.00	0.00	0.00
C2	Quantity	28	18	2	5	6	0	0	0	0	0	0	0	0
	Higher f(x)%	0.00	20.00	10.53	14.29	17.39	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
	Lower f(x)%	0.00	5.26	10.00	12.50	15.38	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
C3	Quantity	6	31	19	0	0	0	0	0	0	0	0	0	0
	Higher f(x)%	0.00	100.00	25.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
	Lower f(x)%	0.00	7.14	11.76	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
C4	Quantity	8	41	4	0	0	0	0	0	0	0	0	0	0
	Higher f(x)%	0.00	50.00	14.29	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
	Lower f(x)%	0.00	7.14	12.50	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
C5	Quantity	7	19	2	0	0	0	0	0	0	0	0	0	0
	Higher f(x)%	0.00	100.00	13.33	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
	Lower f(x) %	0.00	7.14	12.50	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
C6	Quantity	3	34	28	0	0	0	0	0	0	0	0	0	0
	Higher f(x)%	0.00	100.00	40.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
	Lower f(x)%	0.00	7.14	11.76	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
C42	Quantity	20	440	310	196	273	306	495	215	261	89	0	0	0
	Higher f(x)%	0.00	50.00	40.00	13.04	12.90	15.15	17.14	17.95	19.05	20.45	0.00	0.00	0.00
	Lower f(x)%	0.00	3.23	5.88	7.89	10.00	11.90	12.50	14.00	14.29	15.52	0.00	0.00	0.00
C50	Quantity	21	242	383	550	652	364	305	575	134	12	0	0	0
	Higher f(x)%	0.00	50.00	28.57	20.00	22.22	25.00	27.27	26.92	27.59	30.00	0.00	0.00	0.00
	Lower f(x)%	0.00	5.26	8.70	12.00	14.29	17.24	18.18	12.96	14.81	17.65	0.00	0.00	0.00
C51	Quantity	14	201	460	690	749	444	303	533	121	12	8	0	0
	Higher f(x)%	0.00	100.00	40.00	18.75	22.22	25.00	27.27	29.17	28.57	21.43	26.32	0.00	0.00
	Lower f(x)%	0.00	5.00	8.70	12.00	13.33	16.67	18.18 1	3.73	16.33	18.37	17.54	0.00	0.00

Legend: C01 to C63 = are the scenarios; 0 to 12 = the number of components with error; Quantity = is the total of solutions that found a specific number of components with error (0 - 12); Higher f(x) = refers to the solution that had the highest value for Equation 6; Lower f(x) = refers to the solution that had the lowest value for Equation 6

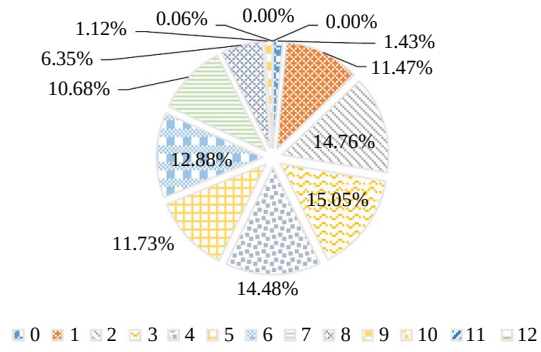

Fig. 11: Percentage of SCUTMA solutions for the budget watch mobile app

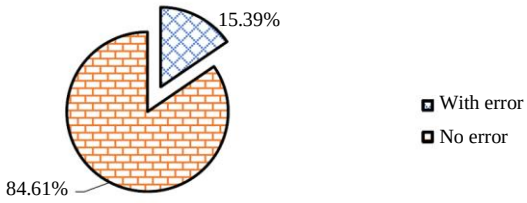

Fig. 12: Percentage of error-free components of SCUTMA solutions for the Budget Watch mobile app

Table 16: Metrics and components with error for the budget watch mobile app

Metric	CE	NE	CE/NE
Cost of future maintenance	422511	2296622	0.18
Market vulnerability	230023	1263927	0.18
Business value	224797	1265740	0.17
Code smell	220903	1398426	0.15
Frequency of call	220808	1252222	0.17
Fault risk	216164	1206150	0.18

Metrics and Components with Errors

With regard to the metrics that compose the component selection prioritization scenarios, Table 16 presents the number of Components with Error (CE) and No Error (NE) for each of the metrics. The Future Maintenance Cost (CFM) metric has the highest number of components with error and no error. This is due to the fact that all the components have a value for this metric, making the scenarios that use this metric have more solutions and consequently more components. It is also observed that the relation between components with error and without error was similar to all the metrics, being between 0.15 and 0.18.

Comparison between the Specialists and the SCUTMA Plugin

For the comparison of the automated selection using the SCUTMA plugin with the manual selection performed by specialists, the result of the specialist S2 was used, since between the two specialists he had a greater coverage (8; 66.67%) of the components with errors. For SCUTMA, scenarios C01, C50 and C51 were selected because they had the best result among the 63 scenarios. Figure 13 presents a comparison between the percentage of components with error selected by specialist S2 and the three best scenarios (C01, C50 and C51) of the SCUTMA plugin.

It is observed that the specialist S2 obtained a lower coverage of the components with error compared to the scenarios C50 and C51 of SCUTMA. Although the number of error components selected by the SCUTMA is greater than that of the S2 specialist, it was observed that the SCUTMA solutions brought a larger number of components that were not identified with error, as shown in Table 17. This indicates that the SCUTMA plugin needs to improve on this aspect, based on the result of this mobile app.

Mobile App Recurrence Results

Table 18 lists the 15 components that were identified with error (CE) for the Recurrence mobile app. This list of components will be used to evaluate the selection of specialists and the SCUTMA plugin.

Specialists Evaluation

The first Specialist (S1) selected 24 components using the criterion “representative components for the business, without dependencies of the Android framework”, which represent 12.37% of the total components of the mobile app. Of these, two of them (components with ids CE5 and CE6) are in the list of fifteen components with known errors presented in Table 18, representing 13.33%.

The second Specialist (S2) selected 37 components using the criterion “components of more complex classes, classes where silly mistakes usually happen”, which represent 19.07% of the total components of the mobile app. Of these, one (component with id CE6) is in the list of components with known errors of Table 18, representing 6.66%.

The third specialist (S3) selected 21 components using the criterion “main features: 1 - Create reminder; 2 – view reminder; 3 - CRUD Database”, which represent 10.82% of the total components of the mobile app. Of these, one (component with CE15 id) is in the list of components with error, representing 6.66%.

The fourth Specialist (S4) selected 33 components using the criterion “public components of business classes and utilities”, which represent 17.01% of the total components. Of these, two (components with ids CE5 and CE6) are in the list of components with errors, representing 13.33%.

The fifth Specialist (S5) selected 34 components using the criteria “components that are independent of

the Android SDK", which represent 17.52% of the total components. Of these, no component is in the list of failed components. In Fig. 14 the graph showing the coverage percentage of each of the five specialists is shown for the components that were identified with errors in the Recurrence mobile app.

Table 17: Result of Equation 6 for the Budget Watch mobile app

	Equation 6 (fx) (%)
Specialist 2 (S2)	50.00
SCUTMA (C01)	34.78
SCUTMA (C50)	30.00
SCUTMA (C51)	26.32

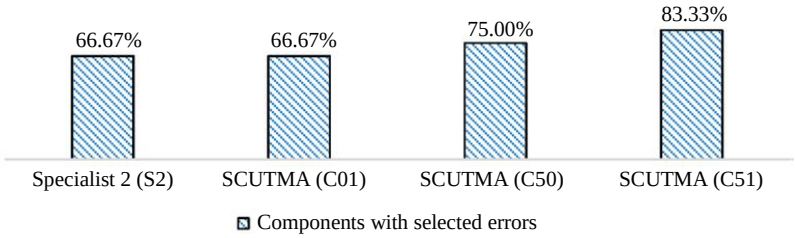

Fig. 13: Comparison between specialists and SCUTMA for the Budget Watch mobile app

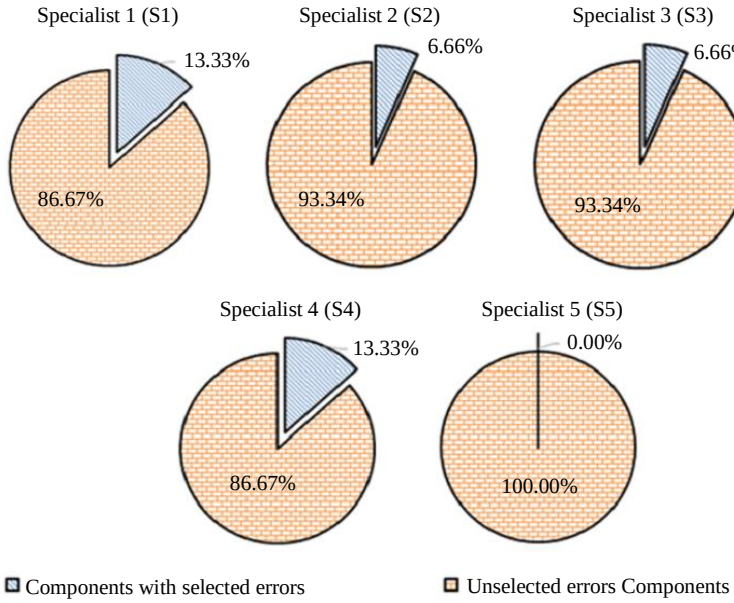

Fig. 14: Coverage of components with errors by specialists for the Recurrence mobile app

Table 18: List of failing components of the recurrence mobile app

ID	Class	Component signature
CE1	AboutActivity	void launchAppURL()
CE2	AboutActivity	void launchEmail()
CE3	AboutActivity	void showContributorsDialog(View view)
CE4	AboutActivity	void showLibrariesDialog()
CE5	AlarmReceiver	public void onReceive(Context context, Intent intent)
CE6	AlarmUtil	public static void setNextAlarm(Context context, Reminder reminder, DatabaseHelper database)
CE7	CreateEditActivity	void colourSelector()
CE8	CreateEditActivity	void datePicker(View view)
CE9	Create Edit Activity	void iconSelector()
CE10	CreateEditActivity	void repeatSelector()
CE11	CreateEditActivity	void switchClicked()
CE12	CreateEditActivity	void timePicker()
CE13	CreateEditActivity	void toggleSwitch()
CE14	MainActivity	void fabClicked()
CE15	ViewActivity	public void actionMarkAsDone()

SCUTMA Plugin Evaluation

Figure 15 shows the number of components with error (Table 18) that could be selected for each metric used in the 63 scenarios. The component with id CE14 of Table 18 was run during the execution of the test cases. Therefore, it has a greater probability of being selected in the scenarios that use the metrics BV, MV, FR and FC. The component with id CE6 is more likely to be selected in scenarios using the PD metric, since code smells have been identified in that component. Finally, all components of the mobile app, including the fifteen components with errors in Table 18, can be selected in scenarios that use the CFM metric, since all components have a cost of future maintenance.

Components with Error

Table 19 shows the coverage of components with error of the best and worst of the 63 scenarios for the Recurrence mobile app. It is observed that the solutions of each of the scenarios managed to find 5 at most components (33.33%) with error. It is worth noting that each of the scenarios obtained a different number of solutions at the Pareto frontier. The Pareto frontier was generated from the 30 runs of the genetic algorithm for each of the scenarios and only the Pareto frontier solutions were analyzed. The complete table with the result of the 63 scenarios can be accessed at "https://goo.gl/92kQEf".

In Table 19 it can be verified that scenario 1 (C01) obtained the solution with the highest value for the ratio of components with error divided by the total number of components (Equation 6), when it found 2 to 5 components with error. This is due to the fact that all components have a Cost of Future Maintenance (CFM).

Scenario 2 (C02) using the default propensity metric selected a component with error and this component has code smells identified (Fig. 15). The four scenarios C03, C04, C05 and C06 using the Call Frequency (FC), Risk of Failure (FR), Market Vulnerability (MV) and Business Value (BV) metrics respectively did not select any components with error, because to reduce the search space of the genetic algorithm, components with less than two cyclomatic complexity were removed and this includes the CE14 component that has cyclomatic complexity equal to one.

Table 19: Component coverage with the best and worst of the 63 scenarios for the Recurrence mobile app

		Components with error															
		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
C01	Quantity	1031	799	346	398	37	4	0	0	0	0	0	0	0	0	0	0
	Higher f(x)%	0.00	50.00	66.67	27.27	15.38	13.16	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
	Lower f(x)%	0.00	3.70	5.26	6.25	8.70	11.11	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
C02	Quantity	7	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	Higher f(x)%	0.00	14.29	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
	Lower f(x)%	0.00	12.50	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
C03	Quantity	29	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	Higher f(x)%	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
	Lower f(x)%	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
C04	Quantity	18	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	Higher f(x)%	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
	Lower f(x)%	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
C05	Quantity	13	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	Higher f(x)%	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
	Lower f(x)%	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
C06	Quantity	29	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	Higher f(x)%	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
	Lower f(x)%	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
C07	Quantity	530	1370	342	384	39	1	0	0	0	0	0	0	0	0	0	0
	Higher f(x)%	0.00	33.33	28.57	17.65	13.79	13.16	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
	Lower f(x)%	0.00	3.45	5.13	6.52	8.33	13.16	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
C23	Quantity	418	1277	345	136	15	1	0	0	0	0	0	0	0	0	0	0
	Higher f(x)%	0.00	50.00	11.76	10.00	11.76	12.82	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
	Lower f(x)%	0.00	2.22	4.55	6.12	7.84	12.82	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
C28	Quantity	816	582	316	292	25	3	0	0	0	0	0	0	0	0	0	0
	Higher f(x)%	0.00	16.67	14.29	10.71	12.50	13.16	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
	Lower f(x)%	0.00	2.56	4.00	6.12	8.16	11.36	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
C29	Quantity	848	603	332	321	41	4	0	0	0	0	0	0	0	0	0	0
	Higher f(x)%	0.00	50.00	14.29	10.71	12.50	13.16	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
	Lower f(x)%	0.00	2.78	4.44	5.77	7.84	11.11	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
C58	Quantity	440	1249	423	178	28	2	0	0	0	0	0	0	0	0	0	0
	Higher f(x)%	0.00	20.00	11.76	15.00	12.50	13.16	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
	Lower f(x)%	0.00	2.56	4.08	6.00	7.84	11.11	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00

Legend: C01 to C63 = are the scenarios; 0 to 15 = the number of components with error; Quantity = is the total of solutions that found a specific number of components with error (0 - 15); Higher f(x) = refers to the solution that had the highest value for Equation 6; Lower f(x) = refers to the solution that had the lowest value for Equation 6.

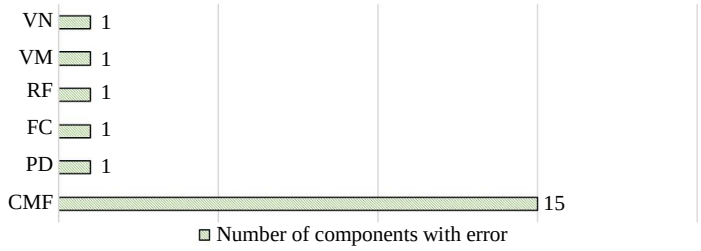

Fig. 15: Number of components with error that could be selected by each metric for the Recurrence mobile app

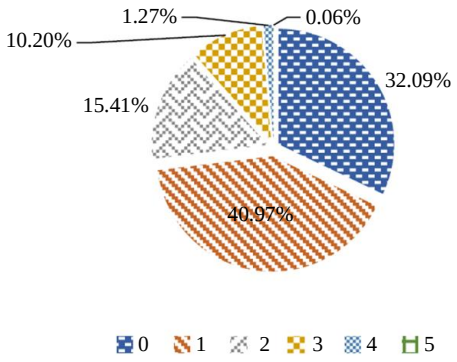

Fig. 16: Percentage of SCUTMA solutions for the Recurrence mobile app

In relation to the best case we will analyze scenario C0, as this was the best scenario for Equation 6. The best solution for scenario C01 with 5 components with error obtained 19.58% of the total components of the mobile app and with this number of components this solution covered 33.33% of the components with errors identified and got 86.84% of additional components. This indicates that in the best case SCUTMA is reaching a low coverage of the components with error, being that the majority (10; 66.67%) of the components with error for this mobile app has a cyclomatic complexity equal to one, making it impossible for its selection by the plugin.

As for the worst case we will analyze scenario C23, because it had the lowest value for Equation 6. The worst solution for scenario C23 obtained 23.19% of the total components of the mobile app and with that number of components this solution covered 6.66% of the components with errors identified and was with 97.77% of components.

Components with General Error

Figure 16 shows the percentage of solutions for each number of error components identified in the Pareto frontier solutions is shown. It is observed that most Pareto frontier solutions identified a component with error and 32.09% of the solutions identified zero components with error. This indicates that SCUTMA has a low error coverage when errors are concentrated in components with complexity equal to one.

For a better understanding, Fig. 17 shows that 95.31% of the components in the solutions did not contain errors. For this reason, it is noticed that the SCUTMA plugin does not have a good error coverage when the errors are concentrated in components with a cyclomatic complexity equal to one.

Metrics and Components with Errors

In terms of the metrics that compose the component selection prioritization scenarios, Table 20 presents the number of Components with Error (CE) and No Error (NE) for each of the metrics. The Cost of Future Maintenance (CFM) metric has the largest number of components with error and no error, because all components have a value for this metric, making the scenarios using this metric have more solutions and consequently more components. It is also noticed that the relation between components with error and without error was equal in all the metrics, with the value of 0,04.

Comparison between the Specialists and the SCUTMA Plugin

For the comparison of the automated selection using the SCUTMA plugin with manual selection performed by specialists, the results of the specialists S1 and S4 were used, since the specialists were among the five participants who obtained a greater coverage (2; 13.33%) of the components with errors. For SCUTMA, scenarios C01, C07, C28, C29 and C58 were selected because they had the best result among the 63 scenarios.

Figure 18 presents a comparison between the percentage of components with error selected by the specialists (S1 and S4) and the five best scenarios (C01, C07, C28, C29 and C58) of the SCUTMA plugin. It is observed that specialists S1 and S4 obtained a lower coverage for the components with error than the SCUTMA.

It can also be observed that specialists S1 and S4 presented a greater number of components that were not identified with error than the SCUTMA, as shown in Table 21. This shows that the SCUTMA plugin obtained a better result than the experts for this mobile app.

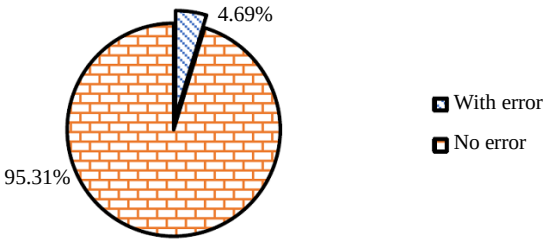

Fig. 17: Percent of components with and without error of SCUTMA solutions for the Recurrence mobile app

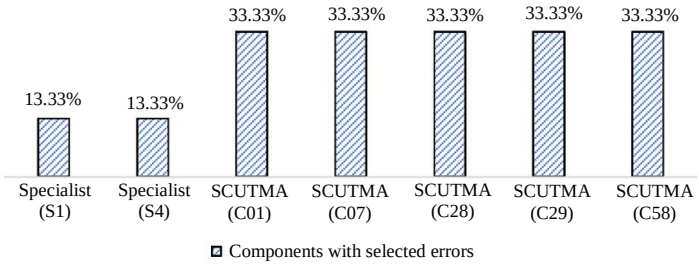

Fig. 18: Comparison between experts and SCUTMA for the Recurrence mobile app

Table 20: Metrics and components with error for the Recurrence mobile app

Metric	CE	NE	CE/NE
Cost of future maintenance	79050	1596576	0.04
Market vulnerability	40194	850033	0.04
Business value	39193	835852	0.04
Code smell	41456	842526	0.04
Frequency of call	39145	823446	0.04
Fault risk	39389	832566	0.04

Table 21: Result of Equation 6 for the recurrence mobile app

	Equation 6 (fx) (%)
Specialist 1 (S1)	8.33
Specialist 4 (S4)	6.06
SCUTMA (C01)	13.16
SCUTMA (C07)	13.16
SCUTMA (C28)	13.16
SCUTMA (C29)	13.16
SCUTMA (C58)	13.16

Threats to Validity

In this section, we discuss the threats to the validity of our results and the measures applied. As part of the study was conducted with people, there is the threat of abandonment of the experiment. To reduce this threat, the time of submission of the second part of the study with specialists was no more than 48 h after the response of the characterization form.

The selection of mobile apps can be a threat, so the sample of the projects was not random, since it came from the availability of f-droid[15] open-source mobile apps that contained UI test scripts within the research group in which the work was accomplished.

In order to reduce the bias of the mono-operation, the empirical study used two Android platform mobile apps from different categories developed in the Java language. However, the study may not be representative for other categories of mobile apps, the study offers some evidence of feasibility of SCUTMA since in some scenarios it selected more components with error than the experts.

In the future, we intend to increase the number of mobile apps to include different mobile app categories.

The threat of selection and treatment interaction was reduced by applying a questionnaire to the characterization of the profile, to verify if the participant fit the profile expected for the study. To reduce the threat of confidence of the measures, the component selection using the SCUTMA plugin was done performing 30 executions of the genetic algorithm to decrease the effect of randomness.

Conclusion and Future Works

In this study, we explore the automated selection of components for unit testing in Android mobile apps.

[15] https://f-droid.org/en/packages/

Therefore, we present the SCUTMA approach in order to increase the cost-benefit value of the selected components. For the SCUTMA approach a set of metrics (HE, CFM, CS, FC, MV and BV) was selected to measure the cost and benefit value of the components. In order to analyze whether the metrics could be used together, a first study was carried out. The result of this empirical study showed the possibility of using the metrics CFM, CS, FC, FR, MV and BV combined in a solution for the selection of components according to Fig. 8.

A second study was conducted to evaluate the effectiveness of SCUTMA in selecting components with error compared to the list generated through the selection of components performed by specialists. The results confirmed the feasibility of the proposal in assisting the developer in the selection of components for unit testing, because in the two applications SCUTMA scenarios selected more components with error than the experts, for example, in the Budget Watch mobile app the best scenario C51 and better E2 specialist respectively selected 83.33 and 66.67% of the components with error. However, the need for refinement to improve the result in some component selection prioritization scenarios was noted, as some SCUTMA scenarios presented more false positives than the experts, for example, in the Budget Watch mobile app the best scenario C51 and better specialist E2 selected respectively 65.22 and 50% of false positives.

Our findings provide a basic understanding of how the SCUTMA approach can be used in different scenarios. This knowledge will help developers choose which metrics to use in component selection in order to improve the efficiency of unit tests. As future work, it is foreseen to carry out a study using the Technology Acceptance Model (TAM), in order to evaluate the usability and acceptance of the SCUTMA plugin. Another possibility would be a study that evaluates the efficiency of using the SCUTMA plugin on the Android platform.

The current version of the SCUTMA approach component selection for the unit test; it would be interesting to include the automatic generation of unit tests from the components selected by SCUTMA.

Another interesting future research will be to analyze how the selection of components behaves through other multi-objective solutions. Finally, there is the possibility of adapting the approach and conducting a feasibility study for the iOS platform.

Acknowledgement

This research was funded by CAPES (Higher Education Personnel Improvement Coordination), Brazil, Finance Code 001; and Research Support Foundation State of Amazonas (FAPEAM) - PAPAC Project (Edital 005/2019).

Author's Contributions

Josias Gomes: Contributing to the conceptualization of the research, developing the methodology, analyzing the results and to the writing of the manuscript.

Isabel K. Villanes, Silvia M. Ascate and Awdren Fontão: Worked on most of the parts, introduction, background and empirical studies and to the writing of the manuscript.

Eduardo Noronha de Andrade Freitas: Supervising the research.

Arilo Claudio Dias-Neto: Supervising the research.

Ethics

This manuscript is original and has not been published elsewhere and contains no ethical issues.

References

Amjad, S. A., & Khan, S. A. (2015, September). A framework for enhancing readability and opportunistic reuse of enterprise software. In 2015 6th IEEE International Conference on Software Engineering and Service Science (ICSESS) (pp. 48-53). IEEE.

Android (2020). Market. https://goo.gl/8d974X

Bourque, P., & Fairley, R. E. (2014). Guide to the software engineering body of knowledge (SWEBOK (R)): Version 3.0. IEEE Computer Society Press.

Burke, E. M., & Coyner, B. M. (2017). Top 12 reasons to write unit tests. https://web.archive.org/web/20170811233259/http://www.onjava.com/pub/a/onjava/2003/04/02/javaxpckbk.html

Coello, C. C. (2006). Evolutionary multi-objective optimization: a historical view of the field. IEEE computational intelligence magazine, 1(1), 28-36.

de Andrade Freitas, E. N., Camilo-Junior, C. G., & Vincenzi, A. M. R. (2016, October). SCOUT: a multi-objective method to select components in designing unit testing. In 2016 IEEE 27th International Symposium on Software Reliability Engineering (ISSRE) (pp. 36-46). IEEE.

de Andrade Freitas, E. N., Vincenzi, A. M. R., & Júnior, C. G. C. (2014). Prioritization of Artifacts for Unit Testing Using Genetic Algorithm Multiobjective Non Pareto. Institute of Informatic, Universidade Federal de Goiás, Goiânia, Goiás., Brazil2009.

Durillo, J. J., Zhang, Y., Alba, E., Harman, M., & Nebro, A. J. (2011). A study of the bi-objective next release problem. Empirical Software Engineering, 16, 29-60.

Fazzini, M., Freitas, E. N. D. A., Choudhary, S. R., & Orso, A. (2017, March). Barista: A technique for recording, encoding and running platform independent android tests. In 2017 IEEE International Conference on Software Testing, Verification and Validation (ICST) (pp. 149-160). IEEE.

Fontana, F. A., Ferme, V., Zanoni, M., & Roveda, R. (2015, October). Towards a prioritization of code debt: A code smell intensity index. In 2015 IEEE 7th International Workshop on Managing Technical Debt (MTD) (pp. 16-24). IEEE.

Halstead, M. H. (1979). Advances in software science. In Advances in Computers (Vol. 18, pp. 119-172). Elsevier.

Haque, M. S., Carver, J., & Atkison, T. (2018, March). Causes, impacts and detection approaches of code smell: A survey. In Proceedings of the ACMSE 2018 Conference (pp. 1-8).

Harman, M., Skaliotis, A., Steinhöfel, K., & Baker, P. (2006, July). Search--based approaches to the component selection and prioritization problem. In Proceedings of the 8th annual conference on Genetic and evolutionary computation (pp. 1951-1952).

He, Y., Zhu, X., Wang, G., Sun, H., & Wang, Y. (2017, July). Predicting bugs in software code changes using isolation forest. In 2017 IEEE International Conference on Software Quality, Reliability and Security (QRS) (pp. 296-305). IEEE.

Hecht, G., Moha, N., & Rouvoy, R. (2016, May). An empirical study of the performance impacts of android code smells. In Proceedings of the international conference on mobile software engineering and systems (pp. 59-69).

Hosseingholizadeh, A. (2010, April). A source-based risk analysis approach for software test optimization. In 2010 2nd International Conference on Computer Engineering and Technology (Vol. 2, pp. V2-601). IEEE.

Ismail, S., Wan-Kadir, W. M., Saman, Y. M., & Mohd-Hashim, S. Z. (2008, August). A review on the component evaluation approaches to support software reuse. In 2008 international symposium on information technology (Vol. 4, pp. 1-6). IEEE.

Jones, J. A., Harrold, M. J., & Stasko, J. (2002, May). Visualization of test information to assist fault localization. In Proceedings of the 24th International Conference on Software Engineering. ICSE 2002 (pp. 467-477). IEEE.

Jorgensen, P. C. (2018). Software testing: a craftsman's approach. CRC press.

Kochhar, P. S., Thung, F., Nagappan, N., Zimmermann, T., & Lo, D. (2015, April). Understanding the test automation culture of app developers. In 2015 IEEE 8th International Conference on Software Testing, Verification and Validation (ICST) (pp. 1-10). IEEE.

Liu, H., Liu, Q., Niu, Z., & Liu, Y. (2015). Dynamic and automatic feedback-based threshold adaptation for code smell detection. IEEE Transactions on Software Engineering, 42(6), 544-558.

Mahmood, S., & Noman, M. (2014, September). Acceptance test case driven component selection approach. In 2014 8th. Malaysian Software Engineering Conference (MySEC) (pp. 49-54). IEEE.

Martin, R. C. (2009). Clean code: a handbook of agile software craftsmanship. Pearson Education.

Mensah, S., Keung, J., Svajlenko, J., Bennin, K. E., & Mi, Q. (2018). On the value of a prioritization scheme for resolving Self-admitted technical debt. Journal of Systems and Software, 135, 37-54.

Mockus, A., & Votta, L. G. (2000, October). Identifying Reasons for Software Changes using Historic Databases. In icsm (pp. 120-130).

Nebro, A. J., Durillo, J. J., & Vergne, M. (2015, July). Redesigning the jMetal multi-objective optimization framework. In Proceedings of the companion publication of the 2015 annual conference on genetic and evolutionary computation (pp. 1093-1100).

Palomba, F. (2015, May). Textual analysis for code smell detection. In 2015 IEEE/ACM 37th IEEE International Conference on Software Engineering (Vol. 2, pp. 769-771). IEEE.

Ray, M., & Mohapatra, D. P. (2012). Code-based prioritization: a pre-testing effort to minimize post-release failures. Innovations in Systems and Software Engineering, 8(4), 279-292.

Ray, M., Lal Kumawat, K., & Mohapatra, D. P. (2011). Source code prioritization using forward slicing for exposing critical elements in a program. Journal of Computer Science and Technology, 26(2), 314-327.

Rubinov, K., & Baresi, L. (2018). What Are We Missing When Testing Our Android Apps?. Computer, 51(4), 60-68.

Rumsey, D. J. (2009). Statistics II for dummies. John Wiley & Sons.

Runeson, P., & Höst, M. (2009). Guidelines for conducting and reporting case study research in software engineering. Empirical software engineering, 14(2), 131.

Samuel, T., & Pfahl, D. (2016, November). Problems and solutions in mobile application testing. In International Conference on Product-Focused Software Process Improvement (pp. 249-267). Springer, Cham.

Shihab, E., Jiang, Z. M., Adams, B., Hassan, A. E., & Bowerman, R. (2010, July). Prioritizing unit test creation for test-driven maintenance of legacy systems. In 2010 10th International Conference on Quality Software (pp. 132-141). IEEE.

Somerville, I. (2011). Software Engineering. Addison Wesley. https://www.pearson.com/us/higher-education/product/Sommerville-Software-Engineering-9th-Edition/9780137035151.html

Vidal, S., Guimaraes, E., Oizumi, W., Garcia, A., Pace, A. D., & Marcos, C. (2016, April). On the criteria for prioritizing code anomalies to identify architectural problems. In Proceedings of the 31st Annual ACM Symposium on Applied Computing (pp. 1812-1814).

Vilkomir, S., Marszalkowski, K., Perry, C., & Mahendrakar, S. (2015, May). Effectiveness of multi-device testing mobile applications. In 2015 2nd ACM International Conference on Mobile Software Engineering and Systems (pp. 44-47). IEEE.

Zakaria, N. A., Ibrahim, S., & Mahrin, M. N. R. (2015, December). A proposed value-based software process tailoring framework. In 2015 9th Malaysian Software Engineering Conference (MySEC) (pp. 149-153). IEEE.

Zhang, Y., Harman, M., & Lim, S. L. (2013). Empirical evaluation of search based requirements interaction management. Information and Software Technology, 55(1), 126-152.

Zitzler, E., Laumanns, M., & Bleuler, S. (2004). A tutorial on evolutionary multiobjective optimization. In Metaheuristics for multiobjective optimisation (pp. 3-37). Springer, Berlin, Heidelberg.