A framework for testing Android apps by reusing test cases

Ajay Kumar Jha, Deok Yeop Kim, Woo Jin Lee
School of Computer Science and Engineering
Kyungpook National University
Daegu, Republic of Korea
ajaykjha123@yahoo.com, ejrduq77@naver.com, woojin@knu.ac.kr

***Abstract*—Android apps are generally developed by an individual developer or a small team of developers, and the developers may not have experience of testing Android apps or they may not have experience of testing any software systems. Furthermore, even an individual that does not have a basic knowledge of Android can build Android apps using various app generator tools available in the market. In this environment, apps may not get tested at all or developers may not prioritize the testing activities, which may result in low-quality or error-prone apps. Eventually, users may give negative reviews to the apps or they may abandon the apps due to bugs. Therefore, instead of designing and writing new test cases, developers need tools and techniques that can automatically test their apps by utilizing the test cases of existing apps. It will not only help novice app developers in testing their apps but also help experience app developers in reducing time and effort to test their apps. In this paper, we propose a framework for testing Android apps by reusing test cases. The framework leverages test cases and domain knowledge of existing open-source Android apps to test new Android apps.**

Keywords — Android apps, test case reuse, unit testing, app testing framework

I. Introduction

With the improvement in tools and techniques for the Android platform and their ease of availability, it has never been easier to develop Android apps. Even Android apps can be automatically built using various app generator tools available in the market [1, 2]. Furthermore, Android apps can be easily disseminated to a large number of users through various app stores. Therefore, there is a constant increase in the number of Android apps. As of January 2019, there are over 2.5 million apps available for download in the Google Play store [3], including the apps that perform sensitive and critical tasks such as financial and medical apps. However, low thresholds to app development do not ensure that apps are of high quality or error-free. The low-quality or error-prone apps can significantly impact the user experience, and users may give negative reviews to the apps or they may abandon the apps [4]. Furthermore, low-quality apps may not appear in the Google Play search results [5], which is a primary method for users to find their apps [4]. Therefore, it is important to adequately test the apps before releasing them to the market.

Unfortunately, the trend is not the same for testing Android apps. The advancement in tools and techniques for testing Android apps has yet to catch up, in part because of the rapid evolution of the Android platform. Although recent advancement in tools and techniques for testing Android apps [6, 7, 8, 9], a large number of app developers prefer to test their apps manually due to several challenges [10, 11] in using the available testing tools and techniques. Some of the major challenges faced by app developers in testing their apps are time constraints, compatibility issues, lack of exposure, complexity of the tools, and lack of experience [10]. Therefore, rather than developing yet another tool, it is important to address these challenges when developing new tools and techniques for testing Android apps.

Some of the major tasks in testing are designing and writing effective test cases. A test case is a specification of the inputs, execution conditions, and expected results. It takes a substantial amount of time and effort to design and write effective test cases that can find defects in software under test. However, most of the app developers want to release their apps as soon as possible before someone else develops a similar app [10]. Therefore, app developers may not have sufficient time or they may not want to invest time in designing and writing effecting test cases. Furthermore, app developers may not have exposure to the available testing tools and frameworks such as Junit and Robotium or they may not have experience of using the available testing tools and frameworks. Learning new tools and techniques takes a considerable amount of time and effort. Therefore, app developers may be reluctant in using the available testing tools and frameworks. Test case reuse is a technique that greatly reduces time and effort in testing a software system. Therefore, the aforementioned problems can be addressed or mitigated by reusing test cases written by expert app developers.

Most of the Android apps are small-sized [12], and they are developed to perform a very specific task. For example, most of the alarm clock apps display time and alert an individual or a group of individuals at a specified time. There are hundreds of alarm clock apps available in the Google Play store. The trend is similar for the apps that perform other tasks. Users can find a large number of apps that perform identical tasks. Therefore, it is highly probable that a large number of Android apps have identical implementations at the code level. Code reuse can be used as a metric to assess the identical code in a software system. A study [13] shows that code reuse is prevalent in Android apps. For example, the study found that 217 apps have the exact same set of classes as another app in the same category of the Google

Play store. Therefore, test case reuse in Android apps can be feasible.

Asaithambi and Jarzabek [14] presented a study on test case reuse in Android platform libraries. They analyzed the Android platform test case libraries to assess the degree of redundancies and identified patterns of repetitions among test cases that are potential candidates for reuse. Finally, they outlined generic representations for the repetition patterns as a practical way to realize the concept of test case reuse. Later, in another work [15], they proposed a Generic Adaptive Test Templates (GATT) approach to address the problem of test redundancies in the Android platform libraries. However, the studies [14, 15] do not apply to Android apps. To the best of our knowledge, there is no existing study that investigates test case reuse in Android apps. However, test case reuse is not a new topic in other software systems. Several researchers [16, 17] have studied test case reuse in different software systems.

In this paper, we propose a framework for testing Android apps by reusing test cases. The framework leverages test cases and domain knowledge of existing open-source Android apps to test new Android apps. The framework uses the test cases that are extracted by mining existing open-source Android apps and analyzes the apps that contain test cases to get the domain knowledge. Based on the domain knowledge, the framework categorizes the extracted test cases, which are then generalized for reuse. The framework then analyzes the app under test to get the domain knowledge. Finally, the framework tests the app by reusing the test cases of the existing apps that match the domain of the app under test.

II. THE VISION

Due to the popularity of the Android platform, a large number of developers are attracted to develop and market their apps. However, the developers may not have prior experience of testing Android apps or they may not have experience of testing any software systems. Therefore, the vision is to create a framework for testing Android apps that can be used by the app developers that do not have prior experience of testing. We envision to achieve the objective by reusing the test cases that are designed and written by expert Android app developers. In addition to the lack of experience of app developers, the proposed framework aim to address several other major challenges, such as time constraints, faced by app developers in testing their Android apps.

An architectural view of the proposed testing framework is shown in Fig. 1. The framework has three main components. The domain analysis component analyzes the domain of the app under test and the domain of the existing apps that contain test cases. The test case generalization component generalizes the test cases extracted from the existing apps. Finally, the test case reuse component reuses the generalized test cases in the app under test. In addition to the aforementioned components, the framework has a test execution environment. The framework uses the JUnit and Robolectric unit testing frameworks on top of the Android Studio to execute the reused test cases.

We have outlined a detailed approach for the proposed framework, which is shown in Fig 2. The approach is mainly based on mining test cases, domain analysis, generalizing test cases, and test case reuse. In this approach, existing open-source Android apps are mined and analyzed to get test cases and domain knowledge, respectively. Furthermore, the app under test is analyzed to get the domain knowledge. Then, the test cases of the existing open-source apps that have the same domain as the app under test are reused in the app under test. Finally, the app under test is executed to get the test results. We provide further details on the working process of the approach in the remainder of this section.

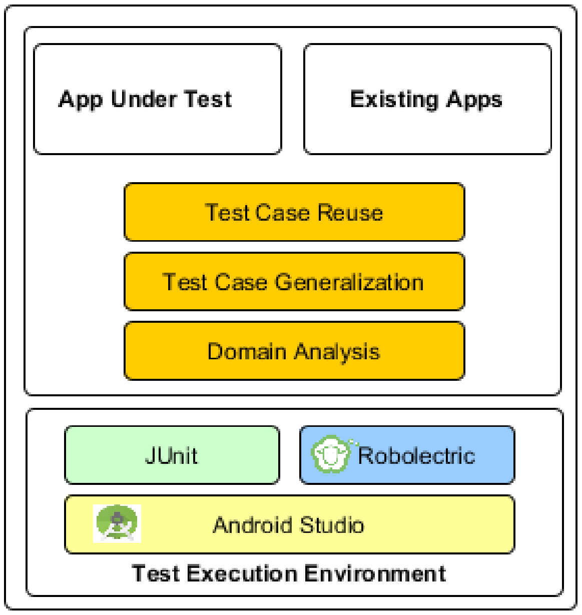

Fig. 1. An architectural view of the test case reuse framework.

A. *Mining Test Cases*

A common way of testing a software system is to design and write effective test cases that can detect bugs in the system. Among different levels of testing such as unit, integration, system, and acceptance testing, the unit testing is used to test individual software components or a group of components, which is widely used for testing a software system. Most of the Android app developers use unit testing frameworks such as JUnit and Robolectric to write test cases for their apps [10, 18]. Therefore, the proposed framework reuses unit test cases of the existing open-source apps for testing new Android apps.

In the proposed approach, the first task is to mine unit test cases from the existing open-source Android apps. To accomplish the task, we will collect a large number of Android apps from various open-source Android app repositories such as F-Droid and GitHub. We will then analyze the collected apps for test cases. The apps that contain unit test cases will be then separated. Finally, we will manually extract test cases from the apps. In this paper, we will explain our approach with a running example. Fig. 3 shows a Robolectric test extracted from a real app named Amaze File Manager [19]. We have excluded some of the code from the test for brevity. The code shown in the figure tests the file open function implemented in the *TextEditorActivity* class of the app. The activity opens the file content in a *TextView*. A user can then edit and save the content in the file.

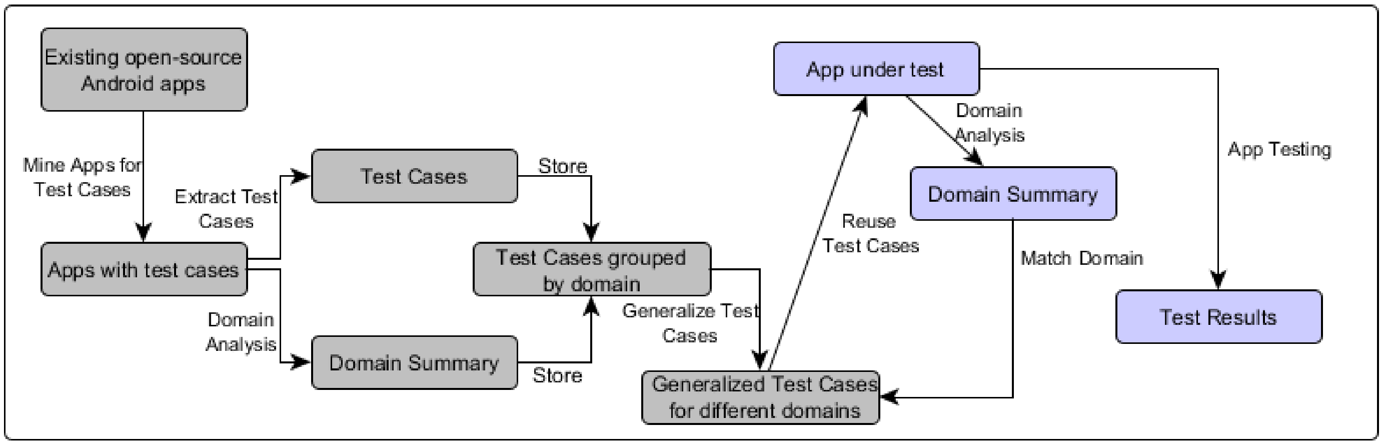

Fig. 2. An approach for test case reuse in Android apps.

```
@RunWith(RobolectricTestRunner.class)
public class TextEditorActivityTest {
  private final String fileContents = "fsdfsdfs";
  private TextView text;

  @Test
  public void testOpenFileUri() throws IOException {
   File file = simulateFile();
   Intent intent = new Intent(Intent.ACTION_VIEW);
   intent.setData(Uri.fromFile(file));
   generateActivity(intent);
   assertThat(text.getText().toString(),
     is(fileContents + "\n"));
  }

  private void generateActivity(Intent intent) {
    ...
    TextEditorActivity activity = controller.get();
    text = activity.findViewById(R.id.fname);
    activity.onDestroy();
  }

  private File simulateFile() throws IOException {
    ...
    PrintWriter out = new PrintWriter(file);
    out.write(fileContents);
    ...
 }
}
```

Fig. 3. A Robolectric test for a file open function.

B. *Domain Analysis*

The systematic discovery and exploitation of commonality across related software systems is a fundamental technical requirement for achieving successful software reuse [20]. Domain analysis is one technique that can be applied to meet this requirement [21]. Therefore, we will use the domain analysis technique to categorize Android apps and reuse their test cases in the app under test.

Although most of the Android apps are designed to perform a very specific task, there is a large variation in the tasks they perform. However, it is not hard to find similar apps in the market. Google Play store organizes similar apps in a category. It has currently 35 top-level categories, and each category has a large number of similar apps. However, we cannot use apps from the Google Play store because they are closed-source. We need open-source apps to reuse test cases. Therefore, we have to categorize the apps collected from various open-source repositories based on their similarity.

The domain analysis will be performed on the existing open-source apps that contain test cases. Based on the result, test cases of the apps that have the same domain will be grouped together along with their domain summary. Furthermore, the domain analysis will be also performed on the app under test. Finally, the domain of the app under test and the domain of the existing apps that contain test cases will be matched to reuse test cases in the app under test.

The Robolectric test shown in Fig. 3 has been extracted from a file manager app [19] available in F-Droid. Through a simple app description analysis, we found three other file manager apps in the F-Droid: Simple File Manager [22], Android File Manager [23], and OI File Manager [24]. Among these four file manager apps, two apps contain test cases. Therefore, the test cases of the two apps, including the test shown in Fig. 3, can be reused in the other two apps. In addition to the file manager apps, the test can also be reused in various other apps that perform the file open function.

C. *Generalize Test Cases*

In the proposed approach, test cases are extracted from the open-source Android apps. The extracted test cases of the apps that belong to the same domain are then grouped together. The test cases are then reused in the app under test if the domain of the app under test matches with the domain of the apps from which the test cases were extracted. However, we need to generalize the test cases that belong to the same domain before reusing them in the app under test.

A test case may contain various parameters and other elements such as resource files that are specific to an app. Therefore, the extracted test cases cannot be directly reused in the app under test. The test cases need to be generalized before they are reused. We will generalize the test cases by identifying and then removing the elements of the test cases that are specific to an app. While testing an app, the removed elements of the test cases will be reintroduced or replaced in the generic test cases using variability techniques adopted from the Software Product Line [25].

Let us take an example of the Robolectric test shown in Fig. 3. It has two components that are specific to an app: the name of the activity class (lines 2 and 18) that performs the file open function and the resource ID (line 19) of the *TextView* that displays the file content. Therefore, the test can be generalized by modeling these two components as variables. Eventually, the test can be reused in another app by assigning the app-specific components to the variables.

As illustrated through the example shown in Fig. 3, the first task that needs to be accomplished to generalize the test is to identify app-specific components in the test file. To accomplish the task, we will manually analyze the extracted test files. The identified app-specific components will be then declared as input variables for the test file. When reusing the test, the developer of the app under test has to provide her app-specific components in place of the declared variables. Manually identifying app-specific components takes substantial efforts, but the proposed system will extract common components that can be reused without any modifications and then transform app-specific components in an automated way.

III. WHY IS IT NEW?

Researchers and practitioners have proposed various automated testing tools and techniques for Android apps [6, 7, 8, 9] to address some of the major challenges [10, 11] faced by Android app developers in testing their apps. However, none of the existing tools and techniques reuses test cases for testing Android apps. Test case reuse is known to reduce effort and improve productivity [16].

The idea of testing a software system by reusing test cases is not new and it has been previously implemented in different software systems using various approaches such as domain-based testing, model-based testing, test patterns, and test frameworks [16, 17]. Furthermore, researchers [14, 15] have also reused test cases for reducing redundancies in testing Android platform libraries. However, the work proposed in this paper has two new components. First, we proposed a framework for testing Android apps by reusing test cases. To the best of our knowledge, none of the existing works that reuse test cases target Android apps, which can be more suitable subjects for test case reuse due to the existence of a large number of open-source apps that provide identical functionalities. Second, our approach is a mining-based test case reuse, which is different than the existing techniques that mostly reuse test cases for regression testing. We mine test cases of the existing open-source Android apps to reuse the test cases in new Android apps.

IV. THE RISKS

There are various research challenges and risks in implementing the proposed framework for testing Android apps. We discuss some of the major research challenges and risks in the remainder of this section.

The proposed framework reuses test cases of the existing open-source apps that have the same domain as the app under test. Therefore, the first major challenge is to collect a large number of open-source apps that cover a wide range of domains, if not all. Open-source app repositories such as F-Droid and GitHub host a large number of Android apps. However, there is a risk that the apps may not belong to a wide range of domains. For example, financial apps are generally closed-source and they may not be available as open-source apps. Therefore, if we cannot find the open-source apps of a particular domain, the framework will not be able to test the apps of that domain.

To reuse test cases, we need a large collection of test cases. However, there may not be adequate open-source apps that have test cases. For example, Kochhar et al. [10] found that only 14.19% of the apps in the F-Droid repository have at least one test cases. Although we plan to include apps from GitHub repository too, we may not still find enough test cases that represent a wide range of domains. Therefore, we may not have test cases of a particular domain that can be reused.

The proposed approach generalizes test cases extracted from existing open-source apps for reuse in the app under test. The approach will use variability techniques adopted from the Software Product Line. However, the current Software Product Line research lacks techniques to address generic test cases. The contributions in this field are either ideas or partial implementations [26]. Therefore, there is a risk that we may not be able to address the problem completely.

V. NEXT STEPS

In this paper, we proposed a framework for testing Android apps by reusing test cases of the existing open-source apps. Furthermore, the proposed framework is aimed at providing a testing platform for Android app developers that do not have prior experience of testing. We are currently working on the implementation of the framework. We have identified a list of key tasks to deliver the proposed framework.

The first major task is to collect open-source apps that contain test cases. We have identified F-Droid and GitHub as our target repositories for collecting apps. We will manually extract test cases from the collected apps. The second major task is domain analysis of the existing apps and the app under test. We could not identify any existing tools that could perform domain analysis on Android apps. Therefore, we plan to use GUI analysis techniques to infer domains of the apps. The third major task is to generalize test cases. We will use variability techniques adopted from the Software Product Line. However, there are hardly any tools available that can be used directly. Therefore, we will develop a tool that can generalize test cases. Overall, the implementation of the proposed framework is a challenging task, which we intend to achieve either by using the available tools and techniques or developing them.

ACKNOWLEDGMENT

This research was supported by the BK21 Plus project (SW Human Resource Development Program for Supporting Smart Life – 21A20131600005), Kyungpook National University, Korea and the Basic Science Research Program through the National Research Foundation of Korea (NRF-2017R1D1A3B04035880 and NRF-2018R1A6A1A03025109) funded by the Ministry of Education.

REFERENCES

[1] Android app builder - https://www.appypie.com/android-app-builder.

[2] Free app creator - https://www.appsgeyser.com.

[3] Number of Android apps in the Google Play store - https://www.appbrain.com/stats/number-of-android-apps

[4] S. L. Lim, P. Bentley, N. Kanakam, F. Ishikawa, and S. Honiden. "Investigating country differences in mobile app user behavior and challenges for software engineering." IEEE Transactions on Software Engineering 1 (2015): 1-1.

[5] Google Play search and discovery algorithm to reflect app quality - https://android-developers.googleblog.com/2017/08/how-were-helping-people-find-quality.html

[6] S. R. Choudhary, A. Gorla, and A. Orso. Automated test input generation for android: Are we there yet? In 2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE), pp. 429-440. IEEE, 2015.

[7] S. Zein, N. Salleh, and J. Grundy. A systematic mapping study of mobile application testing techniques. Journal of Systems and Software 117 (2016): 334-356.

[8] P. Kong, L. Li, J. Gao, K. Liu, T. F. Bissyandé, and J. Klein. "Automated testing of android apps: A systematic literature review." IEEE Transactions on Reliability 99 (2018): 1-22.

[9] P. Tramontana, D. Amalfitano, N. Amatucci, and A. R. Fasolino. "Automated functional testing of mobile applications: a systematic mapping study." Software Quality Journal (2018): 1-53.

[10] P. S. Kochhar, F. Thung, N. Nagappan, T. Zimmermann, and D. Lo. "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, 2015.

[11] M. E. Joorabchi, A. Mesbah, and P. Kruchten. Real challenges in mobile app development. In 2013 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement, pp. 15-24. IEEE, 2013.

[12] A. K. Jha and W. J. Lee. "An empirical study of collaborative model and its security risk in Android." Journal of Systems and Software 137 (2018): 550-562.

[13] I. J. M. Ruiz, M. Nagappan, B. Adams, and A. E. Hassan. "Understanding reuse in the android market." In 2012 IEEE 20th International Conference on Program Comprehension (ICPC), pp. 113-122. IEEE, 2012.

[14] S. P. R. Asaithambi and S. Jarzabek. "Towards test case reuse: a study of redundancies in android platform test libraries." In International Conference on Software Reuse, pp. 49-64. Springer, Berlin, Heidelberg, 2013.

[15] S. P. R. Asaithambi and S. Jarzabek. "Pragmatic Approach to Test Case Reuse-A Case Study in Android OS BiDiTests Library." In International Conference on Software Reuse, pp. 122-138. Springer, Cham, 2015.

[16] R. Tiwari and N. Goel. "Reuse: reducing test effort." ACM SIGSOFT Software Engineering Notes 38, no. 2 (2013): 1-11.

[17] D. Flemström, D. Sundmark, and W. Afzal. "Vertical test reuse for embedded systems: A systematic mapping study." In 2015 41st Euromicro Conference on Software Engineering and Advanced Applications (SEAA), pp. 317-324. IEEE, 2015.

[18] D. B. Silva, M. M. Eler, V. HS Durelli, and A. T. Endo. Characterizing mobile apps from a source and test code viewpoint. Information and Software Technology 101 (2018): 32-50.

[19] Amaze File Manager. https://github.com/TeamAmaze/AmazeFileManager

[20] R. Prieto-Díaz, "Domain analysis: An introduction." ACM SIGSOFT Software Engineering Notes 15, no. 2 (1990): 47-54.

[21] K. C. Kang, S. G. Cohen, J. A. Hess, W. E. Novak, and A. S. Peterson. Feature-oriented domain analysis (FODA) feasibility study. No. CMU/SEI-90-TR-21. Carnegie-Mellon Univ Pittsburgh Pa Software Engineering Inst, 1990.

[22] Simple File Manager - https://github.com/SimpleMobileTools/Simple-File-Manager

[23] Android File Manager - https://github.com/nexes/Android-File-Manager

[24] OI File Manager - https://github.com/openintents/filemanager

[25] E. Engström and P. Runeson. "Software product line testing–a systematic mapping study." Information and Software Technology 53, no. 1 (2011): 2-13.

[26] S. P. R. Asaithambi and S. Jarzabek. "Generic adaptable test cases for software product line testing: software product line." In Proceedings of the 3rd annual conference on Systems, programming, and applications: software for humanity, pp. 33-36. ACM, 2012.