2014 IEEE International Conference on Software Testing, Verification, and Validation Workshops

Using Combinatorial Approaches for Testing Mobile Applications

Sergiy Vilkomir and Brandi Amstutz
Department of Computer Science
East Carolina University
Greenville, NC 27858
{vilkomirs, polkb11}@ecu.edu

Abstract— **Device-specific faults are very common for mobile software applications. To avoid such faults and guarantee the reliability and quality of mobile applications, sufficient testing is required on different mobile devices, which is expensive and time-consuming. This makes the task of the optimal selection of mobile devices for testing important and interesting from both practical and theoretical points of view. The suggested approach in this paper is based on combinatorial methods for coverage of each device characteristics. The initial results of the experimental investigation using comparisons with a random selection of devices are provided and show that the proposed approach is effective and promising.**

Keywords— testing; mobile applications; device-specific failures; combinatorial coverage

I. INTRODUCTION

Practical use of mobile software applications shows that they often work reliably on many smartphones and tablets, but they do not work properly (i.e., fail) on some specific devices. To avoid such device-specific failures and guarantee the reliability and quality of mobile applications, sufficient testing is required on different mobile devices. However, for Android devices alone, several hundred smartphones and tablets are available in the market. A fragmentation matrix created by OpenSignalMaps [1] based on users' statistical information shows that more than 11,000 distinct Android devices downloaded their app in 2013. It is simply impossible to test a mobile application on so many devices. Different empirical approaches are used in practice, varying from using several devices to testing with 400 (!) different phones and tablets for every app (Animoca, publisher of entertainment products for Android [2]).

The choice of devices for testing is usually based on available resources and the experience and/or common sense of the developers and testers. There are currently no good scientific recommendations or solid theoretical methods for selecting an optimal set of mobile devices for practical testing. Therefore, the goal of this paper is to make a step toward in filling this gap.

The paper is structured as follows: Section 2 explains the device-specific failures of mobile applications and describes the problem of selecting mobile devices for testing. Section 3 contains a proposed approach and considers testing on frequently used devices, using combinatorial methods to provide coverage of the device characteristics and the selection of the required number of devices. The example of the utilization of the proposed approach is provided. Section 4 considers results of the experimental evaluation of the effectiveness of the proposed approach by comparing it with the random selection of devices.

The research is partially supported by the National Science Foundation REU Site Award 1262933.

II. DEVICE-SPECIFIC FAILURES OF MOBILE APPLICATIONS

A typical example of a device-specific failure is a situation when the application, which is normally working under some specific operating system (OS), fails under the latest or older OS. Another example is when some graphics created for high-resolution screens are not shown properly on the mobile devices with extra-high or low screen resolutions. Besides the OS and screen resolution, the factors that affect the reliability of mobile applications include device type (e.g., smartphone or tablet), handset manufacturer, RAM memory, and other device characteristics.

To detect the faults, mobile devices with different characteristics should be used for testing. Increasing the number of devices definitely allows for the detection of more faults. However, this requires more time and financial and human resources and significantly increases the cost of testing. Testing on mobile device emulators is useful but cannot completely replace testing on real devices. Special services that provides remote access to hundreds of real (physical) smartphones and tablets, such as Perfecto Mobile [3], Testdroid [4], and AppThwack [5], can facilitate testing; but even in this case, testing on many devices is expensive and time-consuming. So the selection of the optimal number of devices for testing mobile application is an important practical task. In practice, two very simple and reasonable approaches are often used:

- Testing on the most popular (frequently used) devices.
- Testing on devices with different characteristics.

However, when these approaches are applied separately, they contradict each other. Testing on the most popular devices reflects a "user profile." Such devices make up a significant part of all mobile devices, so this approach makes sense. At the same time, the characteristics of popular modern devices are usually similar, so there is a risk of not detecting some device-specific faults. Testing on devices with different characteristics reveals more faults, but there is the risk of testing mainly old or non-popular models, and therefore this is not very practical. In addition, this type of testing is often done haphazardly without the formal definitions or evaluations of the coverage levels.

While the problem of finding an optimal number of devices for testing is important, the number of used devices is often determined by the available resources and this cannot be changed significantly. In such a situation, the main research question is how to select an optimal set of devices for testing when the size of the set is fixed. We consider this question below.

III. Proposed Approach

Our approach to selecting mobile devices for testing has three components:

1. Combining testing on frequently used devices with coverage of the device characteristics.
2. Using well-defined combinatorial methods to provide coverage of the device characteristics.
3. Selecting a required number of devices by restricting the number of options for each device characteristics.

We consider each of these components below in the corresponding sub-section.

A. *Combining testing on frequently used devices with coverage of the device characteristics*

Both the testing on frequently used devices and the testing to cover different values of the device characteristics are very reasonable, so the idea to combine these two approaches is quite natural. To create a set of devices for testing, we select several of the most popular devices and then extend the set by adding more devices to the required size in a such way that the whole set provides a full coverage of all values of the device characteristics.

Information about the most popular devices (in particular, Android models) can be found in many sources. AppBrain [6] and Unity Technologies [7] provide hardware statistical reports based on their users' information. The official site for Android developers also provides data about the frequently used values of the characteristics of Android devices, such as Android version or screen size [8]. This information reflects the devices running the latest Google Play Store app and is constantly updated.

The number of the initially selected popular devices depends on the total number of devices using for testing. For, example, if five devices are used to test an application, then it would be reasonable to select two of the most popular models and then add three more to provide coverage. If the set of all used devices contains about 20 models, then five or six of them could be initially selected based on their popularity. Examples of the sets of devices generated during our investigations are discussed in Section III-D.

B. *Using combinatorial methods to provide coverage of the device characteristics*

The problem of mobile device fragmentation is well-known and is considered as one of the most difficult aspects of the mobile testing [9–11]. In particular, this paper addresses the problem of the selection of the optimal set of devices for testing as often discussed by the software engineering practitioners' community in numerous forums and blogs. However, the justification of using practical approaches to device selection is often that "a common sense filter" was applied [12]. Despite its practical importance, the problem of the selection of devices for testing has not been studied enough in the research literature and no conventional scientific methods have been introduced. At the same time, the situation is ideal for the application of well-known combinatorial testing methods [13].

In particular, we suggest using Each Choice (EC) and pair-wise methods [14, 15]. EC is a simple straightforward approach that requires that every value of every parameter (in our situation, a value of a device characteristic) be used in at least one test case (in our situation, the device). In other words, each value of each device characteristic should be covered by using a corresponding device for testing. The pair-wise method requires that every possible pair of values of any two parameters (device characteristics) are used in at least one test case (the device). Pair-wise embraces EC and so is more effective, but it requires more devices for testing than EC.

The more general approach is t-wise testing [13, 16], which requires using all combinations of the values of t parameters. When t=1, t-wise becomes EC, and when t=2, t-wise becomes pair-wise. The larger the t, the more effective t-wise is and the more test cases (devices) are required. The large number of devices for testing is the main reason why the t-wise approach is impractical for device selection when $t \geqslant 3$.

To understand which approach should be applied for the selection of devices for testing, it is important to evaluate the numbers of devices required according to EC and pair-wise. These numbers depend on the numbers of available options of the device characteristics. Let $n_1, n_2, n_3, \ldots$ be the numbers of options (values) of the device characteristics, sorted in a decreasing sequence, i.e., $n_1 \geqslant n_2 \geqslant n_3 \geqslant \ldots$. Then the bottom limit of the number of required devices is n_1 for EC and $n_1 \cdot n_2$ for pair-wise. The real number of required devices can be slightly bigger than these bottom limits, especially when

- The numbers of options are close to each other for all characteristics. For example, for ($n_1=n_2=n_3=4$, $n_4=2$), the pair-wise approach requires 16 combinations (devices), but ($n_1=n_2=n_3=n_4=4$) requires 20 combinations.
- Not all combinations are feasible, i.e., no devices exist that satisfy some specific combinations of the characteristics' options.

C. *Selecting a required number of devices*

To select a required (predefined) number of devices for testing according to EC or pair-wise approaches, we can artificially restrict the numbers of options for the device characteristics. For this, if we would like to consider only n options for some characteristic, we can choose (n-1) the most popular options and combine all other options in one option, "other." This gives us the ability to cover all possible options during testing.

For example, the decision is made to restrict the number of different options for "Manufacturer" characteristic to four. To do this, we select three of the most popular manufacturers: Samsung, HTC, and Motorola. All other manufacturers are combined and considered as "other."

Combining the approaches suggested above, the algorithm for the selection of mobile devices for testing is as follows:

1. The decision on the approximate number of devices is made based on available resources (funds, time, human resources, etc.).
2. The list of the device characteristics to be covered during testing is created.
3. The formal method of coverage (EC or pair-wise) is chosen based on the numbers of devices and their characteristics.
4. The maximal numbers of options for each characteristic are fixed based on the total number of devices and the method of coverage.
5. The options of device characteristics are restricted using the "other" option to satisfy step 4.
6. Several the most popular devices are selected and combinations of their characteristics are considered for the coverage purposes.
7. The set of combinations of the device characteristic options is extended to cover all options according to the methods of coverage (EC or pair-wise).
8. For each combination from step 7, a device with these values for the characteristics is included into the set of devices for testing.

D. An example

In this section, we consider an example of the proposed approach for the selection of devices for testing. Two different sets of devices ("A" and "B") were selected, and each provides a required coverage:

1. The predefined number of devices is five.
2. The total number of device characteristics is five. The list of the chosen device characteristics is shown in Table 1.
3. EC is chosen as a method of coverage. The pair-wise approach cannot be used for this example as it requires more than five devices.
4. The maximal number of options for each characteristic is restricted to four.
5. The options for each characteristic are shown in Table 1. For two characteristics (device manufacturer and Android OS Version), the option "other" is used.
6. As the most popular devices, Galaxy S2 i9100 and Galaxy Tab 2 7.0 are selected for set "A" and Galaxy S2 i9100 and Galaxy Tab 10.1 p7100 are chosen for set "B." The values of their characteristics are shown in Table 2.
7. The sets of option combinations are extended. Three combinations are added to each set to make the total number of combinations in each set equal to five and to provide coverage of all options (Table 2).
8. For each combination of options, devices with these options are selected (Table 2). It creates two sets of devices for testing and each set provides EC coverage of device characteristics:
    - Set "A": Galaxy S2 i9100, Galaxy Tab 2, 7.0, Xoom HC, Golf, and Optimus L3 e400.
    - Set "B": Galaxy S2 i9100, Galaxy Tab 10.1 p7100, Legend, Xoom ICS, and Xperia X10 Mini.

TABLE I. DEVICE CHARACTERISTICS AND THEIR VALUES

Type	**Device Manufacturer**	**Android OS Version**	**Screen Resolution**	**RAM Memory**
Smartphone	HTC	Honeycomb (3.0-3.2)	Low: less than 470 dp x 320 dp	< 768 MB
Tablet	Motorola	Ice Cream Sandwich (4.0)	Medium: at least 470 dp x 320 dp	≥ 768 MB
	Samsung	Jelly Bean (4.1-4.3)	High: at least 640 dp x480 dp	
	Other	Other	Extra High: at least 960 dp x 720 dp	

TABLE II. EACH CHOICE SETS OF DEVICES FOR TESTING

Each Choice Set	Device Model	Type	Device Manufacturer	Android OS	Screen Resolution	RAM Memory
A	Galaxy S2 i9100	Smartphone	Samsung	Jelly Bean	High	≥ 768 MB
	Galaxy Tab 2, 7.0	Tablet	Samsung	Jelly Bean	High	≥ 768 MB
	Xoom HC	Tablet	Motorola	Honeycomb	Extra High	≥ 768 MB
	Golf	Smartphone	HTC	Ice Cream Sandwich	Medium	< 768 MB
	Optimus L3 e400	Smartphone	Other	Other	Low	< 768 MB
B	Galaxy S2 i9100	Smartphone	Samsung	Jelly Bean	High	≥ 768 MB
	Galaxy Tab 10.1 p7100	Tablet	Samsung	Jelly Bean	Extra High	≥ 768 MB
	Legend	Smartphone	HTC	Other	Medium	< 768 MB
	Xoom ICS	Tablet	Motorola	Ice cream Sandwich	Extra High	≥ 768 MB
	Xperia X10 Mini	Smartphone	Other	Other	Low	< 768 MB

IV. EXPERIMENTAL EVALUATION OF THE EFFECTIVENESS OF THE PROPOSED APPROACH

To evaluate the effectiveness of the proposed approach, we compared the results of testing the same Android applications using sets of mobile devices created according to our approach and sets of randomly selected devices. The size of the sets of devices is the same in both cases. The effectiveness of any testing approach is considered as the ability to detect faults using this approach. In other words, we compared the number of faults detected during testing on different sets of devices.

This section presents initial results of the experimental evaluation of the effectiveness. Our plans are to continue and significantly extend this evaluation. Three mobile applications for booking hotels and air tickets have been tested: Hotwire [17], Travelocity [18], and BookIt [19]. Four sets of mobile devices were used: two sets created according the EC approach (Table 2) and two sets with randomly selected devices (Table 3).

TABLE III. RANDOM SETS OF DEVICES FOR TESTING

Random Set	Device Model	Type	Device Manufacturer	Android OS	Screen Resolution	RAM Memory
A	Galaxy Note 10.1	Tablet	Samsung	Jelly Bean	Extra High	≥ 768 MB
	HTC First	Smartphone	HTC	Jelly Bean	Extra High	≥ 768 MB
	Galaxy Nexus	Smartphone	Samsung	Ice Cream Sandwich	Extra High	≥ 768 MB
	Droid Razr Mini	Smartphone	Motorola	Jelly Bean	High	≥ 768 MB
	Experia Miro	Smartphone	Other	Ice Cream Sandwich	Medium	< 768 MB
B	Droid X2	Smartphone	Motorola	Other	High	≥ 768 MB
	Nexus 7	Tablet	Other	Jelly Bean	High	≥ 768 MB
	Rezound	Smartphone	HTC	Ice Cream Sandwich	Extra High	≥ 768 MB
	Kindle Fire HD	Tablet	Other	Ice Cream Sandwich	Extra High	≥ 768 MB
	Moto X	Smartphone	Motorola	Jelly Bean	Extra High	≥ 768 MB

TABLE IV. NUMBERS OF FAULTS DETECTED DURING TESTING

Application Name	Device-independent faults	Device-specific faults			
	All sets	*Each Choice Set A*	*Each Choice Set B*	*Random Set A*	*Random Set B*
Hotwire	2 Faults: H1, H6	5 Faults:H2, H3, H4, H5, H7	2 Faults: H2, H7	2 Faults:H2, H7	2 Faults: H2, H7
Travelocity	1 Fault: T5	6 Faults: T1, T2, T3, T4, T6, T7	4 Faults: T1, T2, T3, T7	2 Faults: T1, T3	2 Faults: T1, T3
Bookit.com	1 Fault: B1	3 Faults: B2, B3, B4	2 Faults: B3, B4	3 Faults: B2, B3, B4	2 Faults: B2, B4

Each set contains five devices. Testing was done remotely on real mobile devices (i.e., not on simulators) using the Perfecto Mobile service [3]. The results of our testing are presented in Table 4.

During testing, seven faults were detected for the Hotwire application (H1-H7 in Table 4), seven faults for the Travelocity application (T1-T7 in Table 4), and four faults for the BookIt application (B1-B4 in Table 4). Fig. 1 and Fig. 2 illustrate two of these faults. Fig. 1 shows fault H5 for the Hotwire application. The expected result is that the number of rooms should be updated on the main page when a user selects a new number of desired hotel rooms from the drop-down menu. The fault occurs when the room quantity is not updated appropriately. Fig. 2 shows fault T1 for the Travelocity application. The expectation is that the Travelocity menu bar with its gnome icon is displayed at the top of the home screen. The fault occurs when the menu bar is missing from view.

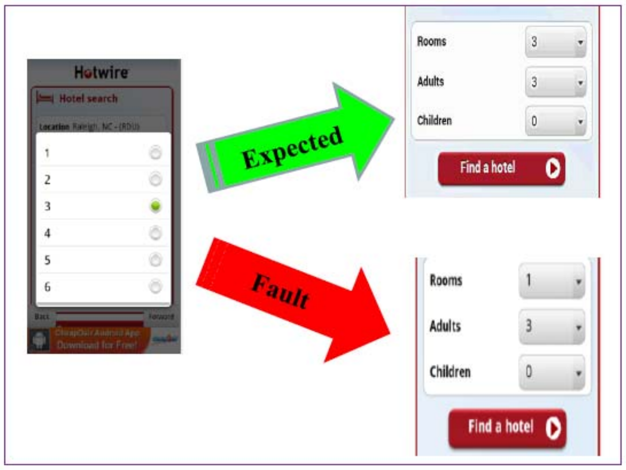

Fig. 1. Fault H5 for the Hotwire Application

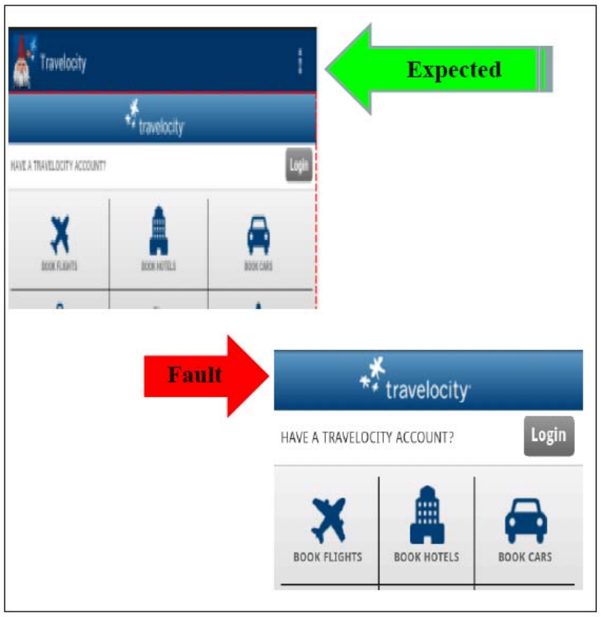

Fig. 2. Fault T1 for the Travelocity Application

Some faults occurred during the testing of the applications on each device and are considered as device-independent. These faults are reflected in Table 4, but they do not affect the comparison of effectiveness. Other faults occurred only during the testing on some specific devices. The numbers of detected faults of this type are presented in Table 4 for the four sets of devices. This gives us 12 cases for comparison of effectiveness: each EC set is compared with two random sets for each of three applications. Table 4 shows that in seven cases (58.5%) our approach is more effective than the random selection of devices; in four cases (33.5%) they provided equal effectiveness; and only in one case (8%) the random set was more effective. Though these experimental results demonstrate that the suggested approach is effective and promising, they cannot be considered as ultimate results and are presented here to show the direction for further more detailed investigations.

V. Conclusions and Future Work

Device-specific faults are very common among mobile software applications. To detect such faults, testing on many different devices is required, which is expensive and time-consuming. This makes the task of optimal selection of mobile devices for testing important and interesting from both practical and theoretical points of view. The suggested approach in this paper for this task includes:

- Combining testing on the most popular devices with coverage testing
- Using Each Choice and pair-wise combinatorial methods for coverage of each device characteristic
- Using restrictions on the maximum numbers of options of the device characteristics to provide required coverage when the number of devices is predefined.

The initial results of the experimental investigation of the proposed approach by comparing with a random selection of devices show that it is effective and promising.

The direction of the future work should include:

- Development of tools to automate the selection of devices and the testing mobile applications. An example of such a tool is the Cloud Testing of Mobile Systems (CTOMS) platform [20, 21]. The CTOMS user can choose a set of models for testing, including EC and pair-wise combinatorial approaches. To select specific devices for testing, CTOMS incorporates the ACTS tool [22] for combinatorial test generation provided by the National Institute of Standards and Technology (NIST).
- Further extended experimental investigations of the combinatorial approaches for device selection. Using more applications, different methods of selection (in particular, pair-wise method), and more devices for testing will increase the trustworthiness of the effectiveness evaluation.
- Analysis of sources of device-specific faults of mobile applications with connection with different device characteristics.

References

[1] OpenSignal, Android Fragmentation Visualized (July 2013), http://opensignal.com/reports/fragmentation-2013/

[2] Kim-Mai Cutler, "This Is What Developing For Android Looks Like," TechCrunch, May 2012, http://techcrunch.com/2012/05/11/this-is-what-developing-for-android-looks-like/

[3] Perfecto Mobile, http://www.perfectomobile.com

[4] Testdroid, http://testdroid.com/

[5] AppThwack, https://appthwack.com/

[6] AppBrain, http://www.appbrain.com/stats/top-android-phones

[7] Unity Technologies, http://stats.unity3d.com/mobile/device.html

[8] Android developers, Dashboards, http://developer.android.com/about/dashboards/index.html

[9] Dan Han, Chenlei Zhang, Xiaochao Fan, Abram Hindle, Kenny Wong, and Eleni Stroulia, "Understanding Android Fragmentation with Topic Analysis of Vendor-Specific Bugs," Proceedings of the 19th Working Conference on Reverse Engineering (WCRE 2012), Kingston, ON, Canada, October 15–18, 2012, pp. 83–92.

[10] Je-Ho Park, Young Bom Park, and Hyung Kil Ham, "Fragmentation Problem in Android," Proceedings of the International Conference on Information Science and Applications (ICISA 2013), 24–26 June 2013, Pattaya, Thailand.

[11] uTest, Inc. "The Essential Guide to Mobile App Testing," (eBook), http://www.utest.com/landing-blog/essential-guide-mobile-app-testing

[12] Kim-Mai Cutler, "How Do Top Android Developers QA Test Their Apps?," TechCrunch, June 2012, http://techcrunch.com/2012/06/02/android-qa-testing-quality-assurance/

[13] D. Richard Kuhn, Raghu N. Kacker, and Yu Lei, Introduction to Combinatorial Testing, Chapman and Hall/CRC, 2013, 341 pages.

[14] M. Grindal, J. Offutt, and S.F. Andler, "Combination testing strategies: a survey," Software Testing, Verification and Reliability, vol. 15, no. 3, Mar. 2005, pp. 167–199, doi: 10.1002/stvr.319

[15] D.R. Kuhn, Y.Lei, and R. Kacker, "Practical Combinatorial Testing - Beyond Pairwise," IEEE IT Professional, Jun. 2008, pp. 19–23.

[16] S. Vilkomir, O. Starov, and R. Bhambroo, "Evaluation of t-wise Approach for Testing Logical Expressions in Software," Proceedings of the IEEE Sixth International Conference on Software Testing, Verification and Validation Workshops (ICSTW 2013), 18–20 March 2013, Luxembourg, Luxembourg, pp. 249–256.

[17] Hotwire - Android Apps on Google Play, https://play.google.com/store/apps/details?id=com.hotwire.hotels

[18] Travelocity.com - Android Apps on Google Play, https://play.google.com/store/apps/developer?id=Travelocity.com&hl=en

[19] BookIt.com - Android Apps on Google Play, https://play.google.com/store/apps/details?id=com.bookit

[20] O. Starov and S. Vilkomir, "Integrated TaaS Platform for Mobile Development: Architecture Solutions," Proceedings of the Eighth International Workshop on Automation of Software Test (AST'13), San Francisco, USA, May 18–19, 2013, in conjunction with the 35th International Conference on Software Engineering (ICSE'13).

[21] O. Starov, S. Vilkomir, and V. Kharchenko, "Cloud Testing for Mobile Software Systems: Concept and Prototyping," Proceedings of the 8th International Conference on Software Engineering and Applications (ICSOFT-EA 2013), July 29–31, 2013, Reykjavik, Iceland, pp. 124–131.

[22] NIST, 2014, ACTS tool, http://csrc.nist.gov/groups/SNS/acts/