Information and Software Technology xxx (xxxx) xxx–xxx

Contents lists available at ScienceDirect

Information and Software Technology

journal homepage: www.elsevier.com/locate/infsof

Combinatorial-based event sequence testing of Android applications

David Adamo*, Dmitry Nurmuradov, Shraddha Piparia, Renée Bryce

University of North Texas, 1155 Union Circle #311366, Denton, TX 76203, USA

ARTICLE INFO

Keywords:
GUI testing
Automated testing
Combinatorial testing
Mobile apps
Android
Mobile application testing

ABSTRACT

Context: Mobile applications are Event Driven Systems (EDS) that take Graphical User Interface (GUI) event sequences as input and respond by changing their state. EDS are often tested with event sequences that exercise system functionality. Much of prior work focuses on testing random event sequences. Combinatorial-based techniques are often used to systematically generate event combinations and may be extended to test behavior that occurs only when events are executed in a particular order. We expand upon the state-of-the-art by using combinatorial-based techniques to systematically test Android applications with automatically generated GUI event sequences.

Objective: This paper describes a combinatorial-based technique for automatic construction of Android application test suites. The goal is to minimize redundant execution of events, maximize coverage of event combinations, and increase the likelihood of testing behavior that occurs when GUI events are executed in a particular order.

Method: A greedy online algorithm selects and executes GUI events that maximize coverage of *n*-way event combinations, where *n* is a specified event combination strength. We compare our combinatorial-based technique to random and frequency-based techniques. We use a two-hour time budget to generate test suites for ten Android applications and empirically evaluate the test suites in terms of code and event coverage.

Results: Our 2-way and 3-way combinatorial-based test suites achieve better code and event coverage compared to random and frequency-based test suites in the majority of our subject applications. The results show that there is no significant difference in code or event coverage between 2-way and 3-way combinatorial-based test suites.

Conclusion: Given the time budget, the combinatorial-based technique is more effective than random and frequency-based techniques, but its effectiveness may vary depending on specific characteristics of the application under test.

1. Introduction

Mobile devices have become an important part of society as they provide a portable means of communication and access to computing services. Smart mobile devices use Operating Systems (OS) that provide a platform for applications in many critical domains such as e-commerce and mobile banking. It is important to develop techniques to cost-effectively test these applications. Google's Android holds the largest share of the mobile OS market worldwide [1]. Android applications are Event Driven Systems (EDSs) that take Graphical User Interface (GUI) event sequences as input and respond by changing their state. Examples of GUI events include clicking a button or entering data in a text field. EDSs are often tested with manually or automatically generated event sequences that exercise system functionality while covering as much source code as possible [2–6]. EDS often include functionality that can be tested only when events in a sequence occur in a particular order. Interactions among these events may cause a System Under Test (SUT) to enter a failure state. Combinatorial-based methods are able to test EDSs where the order of events is important [4]. These methods often require adaptation to the specific constraints imposed by GUI-based software such as mobile applications. As with other types of EDS, it is important to test a mobile application's response to specific events executed in a particular order. The number of possible event combinations in GUI-based software increases exponentially with the number of events. Combinatorial-based methods for event sequence testing manage this complexity by systematically examining combinations for only a subset of events [4,7,8]. Empirical studies in combinatorial testing show that testing interactions among a small number of inputs may be an effective way to detect software faults [9–12].

In this work, we develop a combinatorial-based technique for

* Corresponding author.
E-mail addresses: davidadamo@my.unt.edu (D. Adamo), dmitrynurmuradov@my.unt.edu (D. Nurmuradov), shraddhapiparia@my.unt.edu (S. Piparia), renee.bryce@unt.edu (R. Bryce).

https://doi.org/10.1016/j.infsof.2018.03.007
Received 18 May 2017; Received in revised form 15 January 2018; Accepted 6 March 2018

automatic construction of Android application test suites. Our online algorithm iteratively selects and executes GUI events to construct event sequences. It does not require static analysis of source code or static abstract models of the Application Under Test (AUT). Our combinatorial-based algorithm maintains a history of event combinations as part of the test suite construction process. It uses the historical information to greedily select and execute events that maximize coverage of n-event-tuples (i.e. n-way event combinations), where n is a specified event combination strength. This technique enables online construction of test suites with an increased likelihood of testing behavior that occurs only when events are executed in a particular order.

Existing techniques for automated GUI testing pay limited attention to combinatorial-based testing of mobile applications and often require static analysis of source code or construction of static behavioral models [3,11,13–19]. It is difficult to construct accurate models of GUI-based software and testers may not have access to the AUT's source code. Prior work in online GUI testing of Android apps uses random and frequency-based algorithms to execute event sequences [19–23]. These random algorithms often use a uniform probability distribution to randomly select GUI events and have a tendency to redundantly execute events without consideration for the order in which events have previously occurred. Since our combinatorial-based technique maximizes coverage of event combinations, it may be effective for testing Android app behavior that occurs only when events are executed in a particular order.

This paper makes the following contributions:

- An online combinatorial-based technique to automatically construct Android application test suites and maximize coverage of n-way event combinations, where n is a specified event combination strength.
- Results of an empirical study that compares 2-way and 3-way combinatorial-based test suites to random and frequency-based test suites across ten Android applications in terms of statement coverage, statement coverage rate, and event coverage.

The results of our experiments show that given a two-hour time budget for Android applications with 1026–7981 lines of code: (i) 2-way and 3-way combinatorial-based test suites often achieve significantly higher statement coverage compared to random and frequency-based test suites (ii) 2-way and 3-way combinatorial-based test suites often achieve significantly higher event coverage than random and frequency-based test suites.

2. Background and related work

This section provides an overview of Android application GUIs, combinatorial testing and automated GUI testing of Android applications.

2.1. Combinatorial testing

Combinatorial-based testing techniques systematically examine combinations of parameter-values for a system. These techniques use sampling mechanisms to cover all parameter-value combinations in as few tests as possible. Empirical studies show that combinatorial-based testing techniques often detect software faults triggered by interactions among inputs of a system [9–12]. Kuhn et al. [9,10] study the faults in several software projects and show that most faults are triggered by interactions among six or fewer parameters.

A significant body of prior research proposes algorithms and techniques to automatically generate covering arrays for combinatorial testing [10,24–30]. Many of these algorithms focus on testing interactions in systems where inputs are not sequence-based and the order of inputs is not important. Kuhn et al. [4] apply combinatorial methods to event sequence testing where the order of events is important. They use Sequence Covering Arrays (SCA) to generate event sequences that test t events in different possible t-way orders. Their technique is limited to situations where events cannot be repeated, sequences are of fixed length and there are no constraints on the validity of event sequences. We extend their technique to the mobile application domain where we must consider constraints on the order of events, sequences are not of fixed length, and events may be repeated. We use an online algorithm to adapt combinatorial event sequence testing to Android applications.

There are several applications of combinatorial-based methods to GUI testing. Yuan et al. [11] use covering arrays to construct GUI event sequences that cover all t-way sequences of events. Their technique is based on Event Interaction Graphs (EIG) [11] and handles situations where events can be repeated. The covering array is constructed from an EIG model that has no ordering relationships between GUI events. The next step of the process generates executable test cases by re-inserting ordering relationships between events in each row of the covering array. Wang et al. [3,13] use combinatorial techniques to automatically construct navigational graphs for web applications. They also describe a technique to test all pairwise (2-way) interactions between any two pages of a web application. Their work is specific to web applications and is based on a navigational graph model of the AUT. Di Lucca and Di Penta [14] present a technique that uses state-chart models to test interactions between web applications and browsers. They use graph-based coverage criteria to generate test cases that cover all sequences of k transitions in the state-chart model. This requires that the covered events must occur consecutively in sequence. Carino [31] introduces Event Pair Graphs (EPGs) and uses them to automatically execute event sequences of predefined length. The EPGs guide event execution toward coverage of events and event pairs.

In this paper, we extend combinatorial methods to automatic construction of Android application test suites. Our combinatorial-based algorithm is adapted specifically to Android apps and probabilistically regulates the length of event sequences to vary the number of events in each test case. It allows specification of a desired event combination strength and maintains a set of covered event combinations during test suite construction to maximize coverage of n-event-tuples. The algorithm considers the order in which events occur and does not rely on static analysis of source code, EPGs or static abstract models of the AUT.

2.2. Android GUI overview

An Android app consists of several Java components that are instantiated at runtime. *Activities* are the primary GUI components of Android apps. Each *activity* in an Android app has a unique name and is composed of GUI widgets that users may interact with (e.g. buttons and text fields). Fig. 1 shows an instance of an Android activity with several

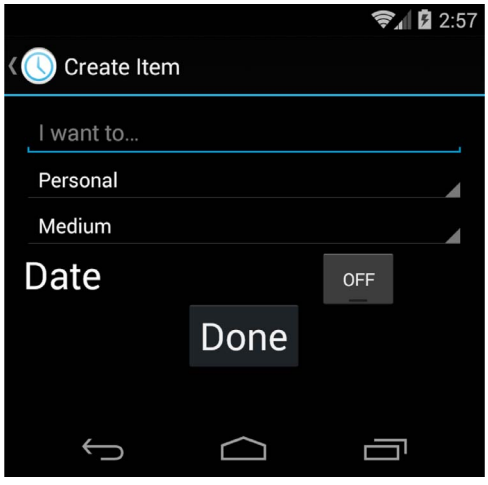

Fig. 1. Example of an Android application GUI.