International Journal of Future Generation Communication and Networking
Vol. 9, No. 11 (2016), pp. 317-326
http://dx.doi.org/10.14257/ijfgcn.2016.9.11.29

An Effective Mobile Applications for Testing Strategies

Haeng-Kon Kim

Department of Computer Engineering, Catholic University of Daegu, Korea
hangkon@cu.ac.kr

Abstract

The effectivivity of test planning in Mobile Application testing improves the quality of Mobile Apps. In the reason of increasing numbers of smart phone applications and their progressive features, mobile phone or smart phone has become a primary resource of communication for worldwide business owners, industrial and office workers.. In fragmented and competitive global market the mobile development cycle is of short period. It is important that one should make sure that the experience every consumer every time they use your application, will be exceptional. Same to functional testing the non-functional testing like Security testing, usability testing also represents an important role effectivivity of test planning.

The emergence of mobile applications gives computing environment to number of companies and platforms, but there are a number of companies and platforms that are not much advanced in mobile computing; for those companies and platforms, it has become a challenge to avail the functionality of the mobile application.

We use UML diagrams in this represented paper in the form of a tree to extract, test cases to verify/validate the behavior of mobile objects. A new model based approach for automated generation of test cases for object oriented systems in this study is presented. Test cases are derived by analyzing the dynamic behavior of the objects due to internal and external stimuli. The scope of the paper has been limited to the object diagrams taken from the Unified Modeling Language model of the system. The genetic Algorithm's tree crossover has been proposed to bring out all possible test cases of a given object diagram. Illustrative case study has been presented to create the effectiveness of our methodology coupled with mutation analysis.

Mobile application testing presents matchless challenges. For each and every challenge that mobile devices face, there should be a proper mobile testing strategic plan.

***Keywords**: Mobile Applications, Mobile Applications Testing Strategies*

1. Introduction

As of today, modern systems are able to generate inputs that drive execution to almost any point in the control flow. Mobile application code-based test generation has made tremendous progress in the recent past: However, automation reaches its limits when it comes to controlling the test's environment: For example, if the tested code accesses an external component such as a service or a database, then this component needs to be controlled by the test as well in order to prevent unwanted side-effects like data loss. A common solution in test generation is to create a stub version of the component that is difficult to control. Such a stub object provides the same interface as the component it represents, but returns predefined values on method calls.

We propose an approach that automatically generates such stub objects, helping to drive test generation towards its goal in cases where automatic generation is difficult or impossible otherwise. The answer applied is to map the stubbed behavior to the input space of the test generation problem: For each method call on the stub object in the test case we need to find an appropriate value that the stub returns during execution. The difficulty of this is that within the scope of a test case the stubbed component should behave similarly to the real component. If this is not the case then that is problematic for two reasons:

First, it can lead to a false sense of confidence in the correctness, because a coverage goal may be satisfied in a way that cannot occur in practice. Second, it may lead to an incorrect test failure or (false positive), *i.e.*, a failure that is not caused by a original bug but by an invalid stubbed behavior. To solve this issue, we examine different methods to infer knowledge about the real behavior of the stubbed instance. Such knowledge can be approximated from existing execution traces or dynamic invariants on the component's behavior derived from these traces, or it can be derived precisely by solving path constraints derived from the stubbed component.

In this paper we use UML diagrams represented in the form of a tree to extract, test cases to verify/validate the behavior of mobile objects concerned. A new model based approach for automated generation of test cases in object oriented systems has been presented. The test cases are derived by analyzing the dynamic behavior of the objects due to internal and external stimuli. The scope of the paper has been limited to the object diagrams taken from the Unified Modeling Language model of the system. The Genetic Algorithm's tree crossover has been proposed to bring out all possible test cases of a given object diagram. Illustrative case study has been presented to establish the effectiveness of our methodology coupled with mutation analysis .

We also provide here a comparison of the two test plans, highlighting their characteristics and respective weaknesses, focusing on different aspects of the envisioned plans, such as, for example, the time and effort necessary to draw up the two test plans or the expressiveness and the degree of detail of test cases attained for the same functionality in the two approaches. It is necessary to specify that it is out of the scope of this paper comparing the final test cases derived using the two approaches either in terms of number of produced tests or time required to execute all them nor in terms of detecting failures.

2. Related Works

2.1. Key Mobile Testing Challenges in Mobile App Test Automation [1]

The main factor that determines an automation tool's success is its ability to work across platforms and technology stacks. Presented list of challenges influence automation success:

For companies the Mobile applications become a "game changing" force in all industries. Let's look at what all challenges introduced this game-changing technologies:

1) Planning of Quick Rollouts

The companies are looking for the golden business opportunities in unique Mobile apps and expecting rapid rollout of quality application or improvements and bug fixes if application is already launched. They push the applications in the market as quickly as possible to avail the benefits of the market boom mobile sector. As a result the QA testing cycle which generally takes two to three weeks depends on the complexity and the size of the application is now reduced to half or one week. Due to clutch in the timelines the QA it is very difficult to problems if mobile applications don't meet the customer expectations.

2) Multi-Platform Compatibility

With the propagation of mobile devices like iPhone, iPad, Smartphones, Tablets, Windows Mobile and wide range of Andriod devices *etc.*, mobile application providers have to provide the multi platform compatibility to reach their audience. In the mobile industries, there is no any industry standards for Operating Systems or device hardware, so testing of apps over a variety of devices is not a simple task. So here we cannot 100 % say that test cases which passed for one device are also passing for other devices, even if the device from the same family.

Lists of combinations while testing mobile apps like Screen resolution, memory sizes, battery, Operating System *etc.*, The creation of separate test case and execution on each device can be the most expensive and time consuming task.

The mobile application market is increasingly growing with a demand of quality product with no any excuses for errors and security holes.

3) Dealing with a variety of connectivity modes

One more important parameter to be considered in the mobile testing is the "Modes of Connection" to access the application. This step can be ignored if the internet connection does not require for application under test, however, almost all applications requires internet so this test case needs to run over different connections like WiFi, 3G, 4G *etc.*, Even you test the application you will face the wide range of applications over different connectivity options. While planning your QA Automation testing strategy you need to consider connectivity modes which are equally important.

4) Creating end-to-end tests

The mobile market demand is to integrate the mobile applications with all platforms and expected to flawlessly access the data on mobile and other platform like Web site. The end to end test cases should be working as expected on mobiles. Consider an example where the order is placed from the mobile device and same can be from the log in into Web site. These mobile apps are expected to work on front end and back-end systems.

2.2. Types of Mobile App Testing [1]

1) Functional Testing

Functional testing performs on the functional behavior of the application to ensure that the application is working as per the requirements. Mostly, testing performs on the user interface and call flows of the application. As like other UI applications, mobile applications also require lots of human consideration. If, functional testing performs on mobile devices manually, not automatic, it is going to be extremely complex, exhaustive and time-consuming task due to various mobile-specific challenges like; various mobile devices, mobile operating systems, and functions & applications involve with mobile devices. The functional testing automation process also requires lots of human resources, money, and time, then too testers are ready to automate the testing process by using many tools due to its strong market value and user demand. Teams can then combine automated tests with selected manual test scenarios to balance the coverage and efficiency of the functional testing. To test some functionalities of the application tester go for manual testing process, later on tester combines manual testing and automation testing for better result.

2) Testing for Performance

The testing process is carried out by tester to test the performance and actions of the applications that pass through various mobile device challenges like; low battery power due to heavy battery uses, network out of coverage area/poor bandwidth/changing internet connection mode (2G, 3G, or WiFi)/changing broadband connection, transferring heavy file, less memory, concurrent approach to the application's server by various users, *etc.*, Application's server and client both strongly affect the performance of the mobile application, so testers perform testing on both side of the application.

3) Testing for Memory Leakage

Memory leakage is one of the worst issues of the mobile application testing that directly affect on the performance of the mobile devices. Due to memory leakage, the process might slow down while transferring the file or in-between accessing any application mobile device might switch off automatically. Thus, Mobile devices come with limited memory as compared with computer system, and by default, most of the mobile OS stop applications those are using extreme memory for processing; memory leakage testing becomes essential to check the performance of the Mobile Applications Testing to ensure that each application of the mobile device is using optimized memory for processing.

4) Testing for Interrupt

Interrupt testing is a process of testing a mobile application that functions may get interrupted while using the application. Those interruptions can be; incoming and outgoing SMS/MMS/calls, incoming notifications, battery/cable insertion and removal for better uses, network outage and recovery, switch off/switch on of the media player and other connecting devices, Low memory warning, and device power cycle(like; low battery notification). An application should be capable to hold these interruptions by going into a suspended state and restarting afterwards.

5) Testing for Usability

Usability testing is used to test the mobile applications in terms of usability, flexibility, and friendliness. The testing process makes sure that the mobile app is now easy to use and offers a suitable user experience to the customers.

6) Testing for Installation

Mobile devices hold two types of applications; the one which automatically comes with mobile OS (while installing OS, it automatically gets installed), and another one you have to install specially from the store to use the particular application.

Installation testing is used to test the particular application is installed, uninstalling, and updating properly without any interruption (user is smoothly and flexibly installing the application).

7) Testing for Operational

Any mobile OS and desktop OS provides in-built back-up and recovery operational functions that save or recover all files or doc of mobile devices or applications that had been lost due to some reason. Operational testing is used to test that the particular back-up and recovery process is working properly and responding as per the requirement

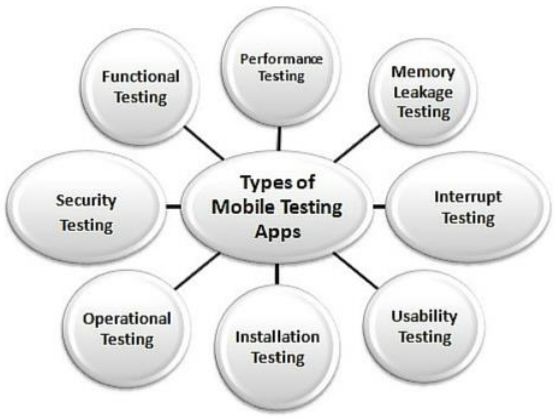


Figure 1. Types of Mobile App Testing

3. Mobile Applications Automated Stubbing

Automated stubbing has been addressed in the past: as taking an existing test case and try to replace some of the used objects with automatically generated stubs that simulate the same behavior, thus aiming to improve performance and making it easier to isolate and detect bugs. Tillmann and Schulte [2] implement stub objects that represent symbolic variables, which the test generator can interpret as inputs. The objective of this approach is to test for robustness, *i.e.,* unlikely (but admissible) test inputs are desired. Automated stub generation is also related to environment generation in software model checking: Tkachuk *et. al.,* [3] simulate a component's environment based on user specified assumptions and additional information derived with different types of static analysis. On the other hands, the object oriented (OO) paradigm has gained widespread use in both industry and academia. The reason for such popularity is mainly the natural correspondence between system components and objects furnished with the distinctive features of OO: encapsulation and inheritance. Moreover, the graphical notation adopted by OO model facilitates system analysis and the representation of various design aspects at different levels of abstraction. UML, the Unified Modeling Language [3], is the de facto standard OO notation and nowadays is being widely adopted in industrial design practice for the modeling and specification of OO systems throughout all phases of the software development process. Although a large body of literature exists for using UML in design, only a relatively small portion is devoted to its application in testing or provides specific assistance for planning and generating tests from UML descriptions. Our research deals with UML-based testing. Testing is clearly an important part of the software development process, which can impact heavily on the cost and reliability of the final product. Hence, it is understandable that the search for practical UML-based methods for improving the effectiveness of software testing has been attracting ever-increasing interest in research circles. The guiding principle of our research is to take the same UML diagrams developed for analysis and design, and apply them, as is, for testing, without the need for any additional formalism or ad hoc mechanisms specifically for testing purposes.

3.1. UML Test Case Generation

Use case development begins at an earlier stage, so real use cases for key product functionality are available fairly early in the project. Use case describes a sequence of actions performed by a system to provide an observable result of value to a person or at the same time another system. Use cases tell the customer what to expect from it, the developer what to basically code, the technical writer on what to document, and the tester what to test.

With this testing approach, creation of test cases is the first step. Then test scripts (collections of test cases) are made for these test cases, and lastly, a test suite/plan is created to implement everything. [13]

Test case (TC) A set of test inputs, executions, and expected results developed for a particular objective.
Test Script/Procedure. A document, providing detailed instructions for the [manual] execution of one or more test cases.
Test suite. A collection of test scripts or test cases that is used for validating bug fixes (or finding new bugs) within a logical or physical area of a product

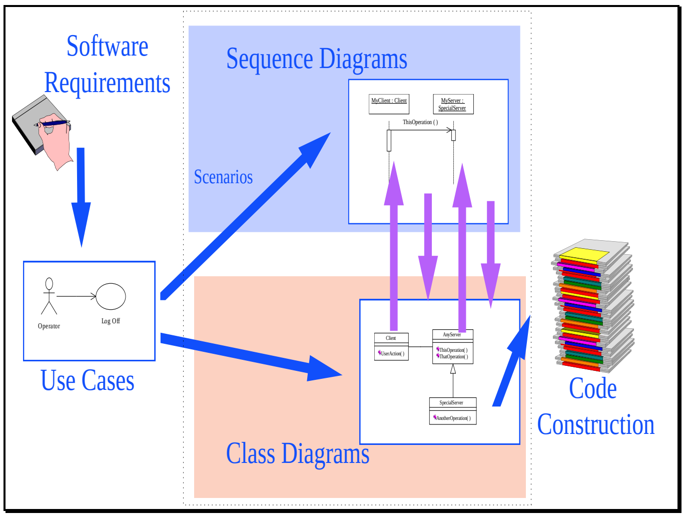


Figure 2. UML Test Case Generation

These test cases are key to the process because they classify and communicate the conditions that will be implemented in test and are essential to verify successful and acceptable implementation of the requirements given.

Although some do it, developers can begin creating test cases as soon as use cases are available, well before any code is written. Based on the Unified Modeling Language (UML) and can be visually represented in use-case diagrams.

The ovals represent use cases, and the stick figures represent "actors," which can be either humans or other systems. In addition, the lines represent communication between an actor and a use case. In the presented figure, this use case diagram provides the big picture: Each use case represents a big chunk of functionality that will be implemented, and each actor represents something outside our system that interacts with it. Each use case also requires a significant amount of text to describe it. This is usually formatted in sections.

Using Use Cases to discover the objects (in the class diagram) that will construct a system to satisfy all functional requirements, and to construct the scenarios to ensure that the functionality can be supported. The functionality can be viewed as a set of processes that run horizontally through the system, and the objects as sub-system components that stand vertically. Not every functionality uses every object, but each object may be used by many functional requirements. This transition from a functional to an object point-of-view is accomplished with Use Cases and Scenarios. Parallel to the mobile software development effort, the software test team can take advantage of the Use Case format by deriving Test Cases and Test Scenarios from them. Figure 2, depicts this process.

The Use Cases and Test Cases are so closely coupled, the linked programs, the Test Team takes ownership of the Use Cases. Both the software development and the test development remain synchronized if changes to the requirements specification occurs. Describing the elements of a Use Case, and provides example of Test Case development based on Use Cases.

Testing in software means dissimilar things to different people. To help clarify our position it will be useful to define the following:

- Verification Testing: required functionality developed
- Validation Testing: error free software; robust

In our organization, Validation Testing during the Unit Testing process. The Use Case derived Test Cases are developed for Verification Testing, and treat the system under test as a Black Box.

- Test Case: A portion of the overall test that is conducted to verify a required functionality.
- Test Scenario: a portion of a functional test conducted by one of the operational variations specified.

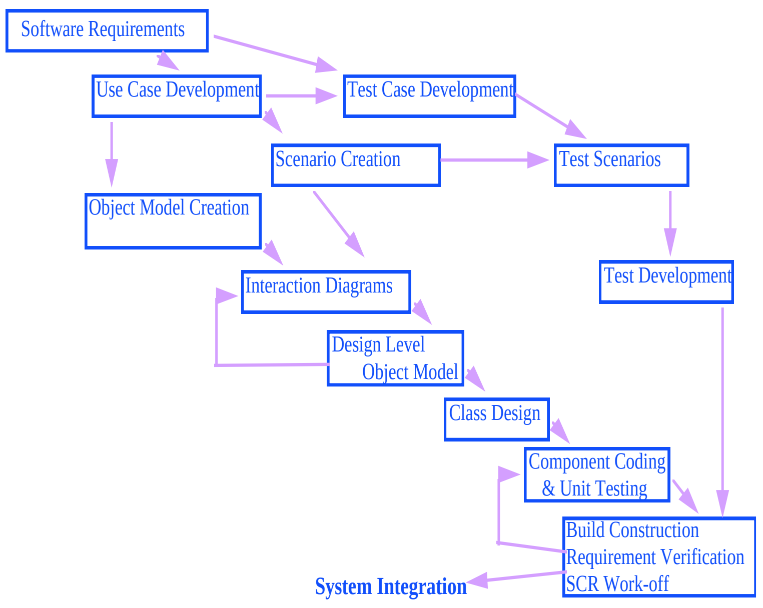


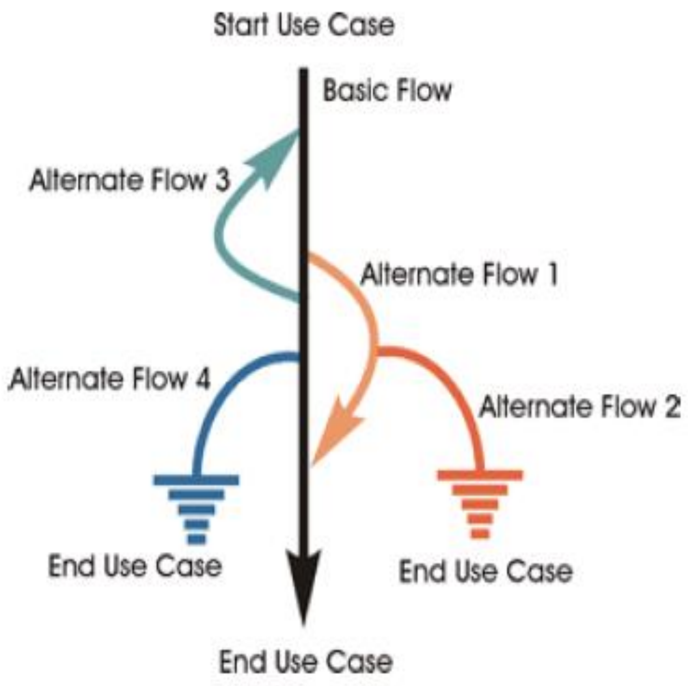
Figure 3. A Parallel Process of Mobile Applications Development and Test Development Starts With a Common Set Of Use Cases

Figure 3, shows the first step is to develop the Use Case topics from the functional requirements of the Software Requirement Specification. The Use Case topics are presented as an oval with the Use Case name. The Use Case diagram just provides a quick overview of the relationship of actors to Use Cases. The main idea of the Use Case is the text description are as follow.

The most important unit of a use case for generating test cases is the flow of events. The two main parts of the flow of events presented are as follows:

1. basic flow of events
2. alternate flows of events.

This basic flow of events should cover what "normally" happens when the use case is performed. The alternate flows of events covers behaviour of an optional or exceptional character relative to normal behavior, and also variations of the normal behavior. Look for some alternate flows of events as "detours" from the basic flow of events.



```
1. S = empty set
2. **foreach** uninterpreted function call *i* do
3.   analyse the interface of *i*, and let the return type of *i* be *t*
4.   add a stub function $m_f$ with the same interface as *i*
5.   add a variable $m_i$ of type *t* to the input domain of the SUT
6.   replace the call to *i* with a call to $m_f$
7.   add $m_i$ to S
8. **end foreach**
9.   **while** not done do
10.      execute the SUT dynamically and symbolically in parallel with an input *I*
11.    form a path condition pc from the result of the symbolic execution
12.    **if** the dynamic execution raises an assertion violation in the SUT then
13.        add the assertion *a* to the path condition pc, describing the violation which raised the exception
14.        **if** the symbolic variables in *a* are (transitively) dependent on a stub variable $m_i$ then
15.            replace every symbolic variable in pc with its concrete value, except those in S
16.            **if** pc is satisfiable then
17.                add the simplified constraints over $m_i$ as a postcondition to the stub function
18.            **else**
19.                raise likely bug exception
20.            **endif**
21.        **else**
22.            raise bug
23.        **endif**
24.   **endif**
25.**end while**
```

4. Conclusion

The testing methodologies presented, prove that users can trust on applications that come with mobile devices, all applications are totally tested with many testing methodologies. Take note to be careful before using applications on mobile devices, if mobile applications involve internet connection, then make sure that the device is already carrying antivirus. Antivirus should exist on your device then it won't be the fault of the applications installed on the mobile device. Errors will count in your mobile system.

Mobile automation testing is not so easy task because it requires preparation, research, and practices to make the testing effective. Mobile application ecosystems are extremely changing, but they also suffer from both software and hardware disintegration. Most of these existing apps were made clumsy on updating devices.

Before starting the testing process; tester should be having good knowledge of information technology skills, the application on which you are about to perform the testing operation, and the tools that you need to use in testing the applications.

Acknowledgments

This Research was supported by the MSIP (Ministry of Science, ICT and Future Planning), Korea, under the C-ITRC (Convergence Information Technology Research Center) support program (IITP-2016-H8601-15-1007) supervised by the IITP (Institute for Information & Communication Technology Promotion).

This Research was supported by the International Research & Development Program of the National Research Foundation of Korea (NRF) funded by the Ministry of Science, ICT & Future Planning (Grant Number: K 2014075112).

References

[1] http://www.softwaretestingclass.com/introduction-to-mobile-application-testing /.
[2] N. Tillmann and W. Schulte, "Stub-object generation with behavior", In Proceedings of the 21st IEEE/ACM international.
[3] Conference on Automated Software Engineering. Automated Software Engineering. IEEE Computer Society, Washington, DC, **(2006)**, pp. 365-368.
[4] F. BasaniAMTG, A. Bertolino and E. Marchetti, "The Cow_Suite Approach to Planning and DAMTG ving Test Suites in UML Projects, Proc. Fifth International Conference on the Unified Modeling Language - the Language and its applications UML 2002, LNCS 2460, Dresden, Germany, **(2002)**, pp. 383-397.
[5] F. BasaniAMTG, A. Bertolino and E. Marchetti, "The Cow_Suite Approach to Planning and DAMTG ving Test Suites in UML Projects", Proc. Fifth International Conference on the Unified Modeling Language - the Language and its applications UML 2002, LNCS 2460, Dresden, Germany, **(2002)**, pp. 383-397.
[6] H.323 Standard http://www.microsoft.com/windows/NetMeeting/Corp/reskit/Chapter11/default.asp.
[7] T. J. Ostrand and M. J. Balcer, "The Category Partition Method For Specifying and Generating Functional Tests", Communication of the ACM, vol. 31, no. 6, **(1988)**, pp. 676-686.
[8] M. Paul, CMM v2.0 Draft C, **(1997)**.
[9] Rational Unified Process version 2000.02.10. Rational Software Corporation. On-line.
[10] http://www.rational.com/products/rup.
[11] UML Documentation version 1.5 Web Site. On-line at.
[12] http://www.omg.org/technology/documents/formal/uml.htm.
[13] http://www.drdobbs.com/architecture-and-design/test-case-generation-uml-and-eclipse/211600996.