Reverse Engineering from Exploratory Testing to Specification-based Testing

Dea-Kwang Kim and Lee-Sub Lee*

Department of Computer Engineering, Kumoh National Institute of Technology,
61, Daehak-ro, Gumi-si, Gyeongsangbuk-do, 730-701 South Korea
{aav7co, eesub*}@kumoh.ac.kr

Abstract

GUI testing takes a crucial role of acceptance testing. SBT (Specification-based Testing) and ET (exploratory testing) are mainly used for GUI testing. SBT is not practical because it requires high level experts and a lot of efforts for creating specifications. Therefore, in most cases, test cases are manually written based on ET by testers. However, ET requires heavy labor cost for enhancing coverages. Thus, it is not programmatic for better coverage than SBT. In this paper, to take advantages of SBT we proposed a method applying a reverse engineering concept which automatically generates formal specifications from manually written test cases. The paper also proposed a method to expand uncovered test paths and test data. The generated specifications could contribute to enhance the quality of software with a few additional manual tasks.

Keywords: software engineering, software testing, reverse engineering, GUI testing, specification based testing, exploratory testing

1. Introduction

Graphical user interface (GUI) is a very important method for interacting between the user and the system. Because it is the last resort to verify the system under test (SUT), GUI testing takes a crucial role of acceptance testing. However, GUI testing is done by hand so it is a highly labor-intensive part [1]. As a result GUI testing should provide sufficient coverages to ensure the quality requirements of the product. Figure 1 depicts a classification or taxonomy of testing techniques. As shown in the figure, SBT (Specification-based Testing) and ET (Exploratory Testing) are mainly used for GUI testing. SBT methods generate a set of test cases from specification documents such as formal requirements specifications [2-3]. ET methods unite less formal testing techniques, but some of them are still very powerful when are used by professionals. These techniques are Checklist-based Testing, Exploratory Testing, Error Guessing, and Ad-hock Testing. The advantage of the SBT is that if it is written appropriately, due to the easiness of automatic test cases generation, it provides sufficient test coverages. However, SBT requires high level professionals and a lot of efforts for creating the specifications. Furthermore when we consider the competitive market environment, to satisfy a very strict timeline for time to market, developers and tester should choose ET methods. Therefore, in most cases, test cases are manually written based on ET using related tools such as Record/Playback, Data-driven, and Keyword-driven tools.
ET requires highly labor-intensive work to expand even for little more coverage. It requires too much labor cost so QA (Quality Assurance) team cannot afford to. Therefore ET could not provide sufficient test coverage. As a result with ET GUI testing cannot provide enough quality of the product.

In this paper, to take advantages of SBT, we proposed a method of applying a reverse engineering concept that automatically generates formal specifications from manually written test cases. The paper also proposed a method to expand uncovered test paths and data. The generated specifications could contribute to enhancing the quality of software with little additional manual tasks.

The paper is structured in the following way: In the next section we presented related work of automatic test cases generation and reverse engineering. In the Section 3 our proposed method for test case generation is presented. The last section concluded the paper and gives an overview of our work.

2. Related Works

2.1. Exploratory Testing

ET plays an important role in effective software testing. A study shows that test design is to a considerable extent based on ET in most projects [4-5]. Record and playback is a popular and traditional method for which the tester records the manual execution of a test case for automatic regression testing [6]. ET is also crucial to the success of the Record and Playback.

Sikuli [7-8] is an approach to GUI testing using computer vision for testers to automate their tasks. Testers can write a visual test script that uses images to specify which GUI components to interact with and what visual feedback to be observed. A variety of GUI behavior can be tested using this approach. We choose this tool because with visual test codes testers can handle various environments such as Web, Android, IOS and etc.
2.2. Model-based Testing

MBT (Model-based Testing) is also known as Specification-based testing. It focuses on test cases generation by means of formal models such as state transition diagram and UML Sequence diagram [9]. Traditional methods adopt a very straightforward approach; define a suitable formal model and generate test cases automatically. The representative tool is Spec Explorer[10] that is a tool developed at Microsoft Research. Most of them define state transition diagram and convert it to an intermediate model such as TFG (Testing Flow Graphs). From the intermediate model it generates various test cases [11-12].

To avoid the cost of defining a perfect formal model, several automatic formal specification extraction methods are introduced. For instance, by analyzing static HTML tags, a UML model of Web applications can be generated [13]. The model is a starting point for several analyses, which can help in the assessment of the static site structure. It drives Web application testing, in that it can be exploited to define white box testing criteria and to semi-automatically generate the associated test cases. Similarly, for the windows application testing an automatic model generation from MFC (Microsoft Foundation class) was introduced [14]. MFC has a metadata for GUI components. From this information the tool can reconfigure GUI model and from it with a simple algorithm a various test cases can be generated. The problem of this type of approach is the dependence on a specific platform. This means that a new method is required for a new platform and testers cannot reuse their knowledge to new context. Another crucial problem is that test cases are generated from very restricted GUI information that is not includes neither required quality information nor testers’ exploratory.

3. From Exploratory Testing to Specification-based Testing

3.1. Reverse Engineering in the Testing Domain

Figure 3 presents an overview of foreword and reverse engineering in the software development domain. Reverse engineering is focused on the challenging task of understanding legacy program code without having suitable documentation. Using a transformational forward engineering perspective, we gain the insight that much of this difficulty is caused by design decisions made during system development [15].

Figure 2. An Example of Visual Testing Scripts of Sikuli to Automate Labor-Intensive Task
To identify the concept of reverse engineering in the test domain we should consider forward engineering in that domain. Figure 4 shows the concept of forward engineering in the test domain. ET adapts very a straight foreword method; write test cases with testers’ exploratory that is based on tacit knowledge. On the other hands, in the case of SBT, most of forward engineering methods define state transition diagram and convert it to an intermediate model such as TFG (Testing Flow Graphs). From the intermediate model it produce useful test cases [11-12].

There are several differences between software development reverse engineering and test reverse engineering. Firstly, the purpose is for testing. Thus the same testing person performs test development and test analysis. Therefore it is possible to improve the code efficiently. Therefore the tester can use many convenient testing functions for analysis phases when developing test cases. Data analysis is focused on persistent data in the existing reverse engineering. On the other hand in the reverse engineering domain data analysis is focused on transient data.

3.2. Foreword and Reverse Engineering in the Test Domain

Figure 5 presents a process that is a combination of ET foreword engineering and SBT reverse engineering. QA officers write testing requirements in the step 1. In the step 2, testers write test scripts to meet the requirements. For an efficient analysis during the later steps, testers use helper functions written in python codes that are library scripts of the Sikuli. Manually written test scripts are parsed so that a test model, that is a kind of TFG, produced in the step 3. Because the hand written scripts have not enough information for the state transition diagram, reverse engineering concept is applied until TFG, not the state transition diagram. During the step 4 constructed models are grouped into a small number of test models. The generated test models reflect the manually written test coverage. In the step 5 the model is expanded to cover a better coverage in terms of paths and data. In this step testers could add more information about test data. Consequently for better enhancement the generated test scripts could be feedback to the step 3. It goes iteratively until the test cases can meet the quality requirements.
Figure 4. Foreword Engineering in the SBT

3.3. Detail Steps in the Reverse Engineering

- Step 2: Writing test scripts

Sikuli provides a few number of but powerful commands such as click, drag and drop, type, assertExist, assertNotExist, and find commands with OCR (Optical Character Recognition) feature to accommodate various environments. However it is very difficult to identity semantic information for analysis so that we propose to use several convenient methods that contain semantics for the testing. It will help human readability, productivity and analysis. The followings are descriptions of the convenient methods.

- Precondition is used to define an initial screen and predicate for a test case precondition. Screen has a same role of the node in the TFG. The first occurrence of a screen is the screen definition and the later occurrences mean the reference of the screen.

- Move contains transition or edge information in the TFG. It contains the movement id, type and value. For the context information it also has a source screen and a destination screen. The movement type could be various moving actions such as Click, Swipe, and menu select, etc.

- Check is the intermediate checkpoint to verify a proper progression of a test case. Actually it is a screen definition or reference.

- Input is an action that accepts various test data. This includes input type, and input value in a specific screen. Normally test data tends to be redundant and a little variance for the quality assurance purpose. They are very important elements for extract test data.

- Postcondition is a test oracle. Several test scripts can be grouped according to same precondition, subsequent screens and same postcondition with a different predicate.
Figure 5. An Overview of Foreword and Reverse Engineering in SBT and ET

Figure 6 presents an architecture of the model that is constructed from the written test scripts during analysis. Script group classifies scripts into several groups in the step 5 according its pattern.

The following example script code shows only semantics and it can be translated to the model syntactically. It is a test case of adding a contact information in a phonebook app.

1. def testAddContact (self):
2.     precondition (“ContactList.png”, AssertNotExist (“John.png”));
3.     move (click, “add.png”, “AddContact.png”);
4.     input (“Name.png”, “John”);
5.     input (“PhoneNumber.png”, “010-452-4875 “);
6.     input (“Group.png”, “Friend.png”);
7.     move (click, “Save.png”, “ContactList.png”);
8.     postcondition (“ContactList.png”, AssertExist (“John.png”));

Figure 6. Test Model Architecture
Step 3: Construct Model

A model is constructed from test scripts in the step 3. The progress of model construction is described below. By reading each script, the model is constructed. In the version 1, line2 is processed. As a result from the precondition we can have a screen definition. Version 2 shows that after line 3 is processed screen s2 and move m1 are defined simultaneously. The three input elements are defined after input data sequences, line 4, 5, and 6, are read. The screen context is also added automatically. Version 4 shows that trivial processing is done by line 8 and line 9.

Version 1.
Precondition: [s1] // by line 2
Screen:  [[s1, ContactList.png, AssertNotExist ("John.png")]] // by line 2
Move: []
Input: []
Postcondition: []

Version 2
Precondition: [s1]
Screen:
[[s1, ContactList.png, AssertNotExist ("John.png")],
[s2, AddContract.png, null]] // by line 3
Move: [[m1, click, Add.png, s1, s2] // by line 3 ]
Input: []
Postcondition: []

Version 3
Precondition: [s1]
Screen:
[[s1, ContactList.png, AssertNotExist ("John.png")],
[s2, AddContract.png, null]]
Move: [[m1, click, Add.png, s1, s2]]
Input:
[[i1, "Name.png", "John", s2], // by line 5
[i2, "PhoneNumber.png", "010-452-4875", s2] // by line 6
[i3, "Group.png", "Friend.png", s2]] // by line 7
Postcondition: []

Version 4
Precondition: Precondition: [s1]
Screen:
[[s1, ContactList.png, AssertNotExist ("John.png")],
[s2, AddContract.png, null]]
[s3, ContactList.png, AssertExist (John)] // by line 9
Move: [[m1, click, Add.png, s1, s2]]
[[m2, click, Save.png, s1, s2]] //by line 8
Input:
[[i1, "Name.png", "John", s2, m2],
[i2, "PhoneNumber.png", "010-452-4875", s2, m2],
[i3, "Group.png", "Friend.png", s2, m2]]
Postcondition: [s3] // by line 9
Step 4: Grouping

In many cases, tester performs similar test paths and data with a little variation. Enhancing coverage is achieved with more suitable test path and data. For the automatic test cases expansion we need to group similar test paths and data. Intuitively similar test paths can be identified by identical sequence of screens with similar input sequences. Figure 7 shows that how two similar test paths are integrated into a test path. In the Figure 8 shows that two test paths can be grouped into a test case with the same source and destination with different input data.
Step 5: Test cases expansion

After grouping, test cases can be expanded. Figure 9 depicts the test path expansion in the case of the same screen sequence with different test data. In this figure, two hops are existed and in each hop there are two alternative input data. This expansion can be generalized by following equation (1). Product operation can cause many expansion.

Figure 10 shows that test data expansion can be possible when the multiple data input are existed. This shows that we can extract much more test cases that can be missed by testers from multiple data alternatives. This expansion can be generalized by following expression (2).

\[
total\ number\ of\ test\ cases = \prod_{i=1}^{h} N_i
\]  
(1)

Where \( h \) is the number of hops and \( N_i \) is the number of input data of \( i \)-th hop.

\[
total\ number\ of\ test\ cases = \prod_{i=1}^{h} \prod_{j=1}^{n} N_{ij}
\]  
(2)

Where \( h \) is the number of hops and \( N_{ij} \) is the number of input data of \( i \)-th hop and \( j \)-th input.

Normally testers tries test with same test sequence with different test data to find defects. We can categorize scripts to script groups. The definition of membership function of same category is that both have identical precondition and postcondition. They also should have same node transition sequence except data input.

For the generated test cases, it is very difficult to define suitable test oracles automatically. The reason is that automatic generators cannot understand semantics. Thus in this method we define default oracle as Assert(False) to ensure that it is generated by automation and testers should redefine test oracles. Tester can reduce this task by using more sophisticated oracle expression and the feedback mechanism.

4. Conclusion

We have presented the concepts and foundations of reverse engineering from ET to SBT for automatic test cases expansion. Although SBT is suitable for more coverage of the software quality assurance, it suffers from the lack of experts and cost for writing formal specifications.
Although the generated model is not perfect for the testing it contains all the hand written test coverage and has the following advantages. Firstly, it did not require modeling experts who have very deep knowledge about formal method concepts. Secondly, there are too many testing models that are specific to various testing environments. ET is not perfect but all round player or testing domain neutral solution in most cases. Thirdly ET have been used in most cases so only a little learning curve is required. Fourthly, it provides more readable outputs that help readability, enhancement, and easy management of test assets. Thus, it encourages better reuse of the test assets. Lastly, it provides mechanical process for uncovered test paths and data that testers difficult to find.

There are several interesting directions for further research in which the technology can be improved. Some of the main directions are reducing the number of test cases for better performance, more detail input User experience, and reverse engineering to state transition diagram.

Acknowledgements

This paper was supported by the Kumoh National Institute of Technology Research Grant. This article is a revised and expanded version of a paper entitled “Test case Expansion Method from Experience-based Techniques to Specification-based Technique” presented at The 3rd International Conference on Next Generation Computer and Information Technology on October 24-26, 2014 at Liberty Central Saigon Hotel, Hochimin, Vietnam.

References


Authors

Dae-Kwang Kim, is being served as application developer at Kyungpook National University, Research Center for Embedded Software. He received M.S. degree in Computer Engineering from Kumoh National Institute of Technology. His research work has been on the Software Engineering. His recent interest focuses on Software Architecture.

Lee-Sub Lee, is an Associate professor of Department of Computer Engineering at the Kumoh National Institute of Technology. He received B.S. in Mathematics and M.S. degree in Computer Engineering from Sogang University, Seoul, Korea. He received his Ph.D. in Computer Engineering from Korea University, Seoul, Korea. He has worked as a senior researcher at Samsung SDS 1990 to 2004. His research work has been on the Software Engineering and Database. His recent interest focuses on Software Testing.