Automating tasks in GUI Test Case Generation

December 10, 2009

Abstract

In a software project lifecycle, the ***software testing*** *phase is expensive. By efficiently automating the testing process, we can significantly decrease the overall cost of software development and maintenance. Today, testing Graphical User Interfaces (GUI) has become an essential part in project validation. Consequently, the need for automating GUI testing has gained importance but due to the large number of states to be tested, GUI test automation has become a major challenge. Most existing GUI test automation tools work on automating the process of test case generation but there is little research on automating the process of developing tasks for GUI test case generation. This proposal presents a novel approach for automating the process of developing the GUI test tasks. Our approach categorizes GUI controls into two sets based on whether they are interlinked or independent in the application. The independent controls can be tested individually while we create test tasks for the interlinked controls by deriving different sets of Linearly Independent paths that cover all the sequences of controls in the GUI application. This would create a set of sequences of controls that are valid and not redundant. Overall, we believe that the automation of creating tasks would provide an adequate coverage of the GUI system as well as support the process of Regression testing.*

1. Introduction

Software testing is a phase in the software process that is aimed at evaluating a software item (system, subsystem, unit etc.) and its features (functionality, performance etc.) against a given set of system requirements. It is a vital phase because it checks for errors and verifies if all the requirements are fulfilled. Software testing is quite expensive; in order to reduce the cost of testing, automation is used.

A graphical user interface (GUI) is a convenient human-computer interface that has become ubiquitous in today's software. Although a GUI makes software very user-friendly, it creates hurdles in the software development process. Especially, the GUI testing process is very complex because of the following reasons[1][4]:

- The GUI possesses an enormous number of states that may need to be tested. The input space size is extremely large due to the number of different permutations of inputs and events which affect the GUI.
- There may exist extremely complex dependencies in the GUI system.
- Another difficulty is faced when the

tester has to perform regression testing because the GUI may change drastically across versions of the application.

By automating the GUI testing process, we can attempt to solve most of the above issues. In general, a software test automation process consists of selecting and generating test cases, building the test oracle, executing and validating the results. In this proposal, our main focus is on test task selection and generation. Automation of test case generation has already been proposed by many researchers but selecting the test cases is an unaddressed issue. Creating and maintaining test cases manually is a very time consuming process. So, some automation is required when testing GUIs. Through this proposal, we propose an approach to automatically develop the tasks for GUI test case generation.

2. Background

A graphical user interface (GUI) is a human-computer interface that offers graphical icons, and visual indicators, instead of text-based interfaces, typed command labels or text navigation in order to completely represent the information and actions available to a user. Using a combination of technologies and devices, a GUI presents a user interface for various tasks like gathering and producing information. It consists of a series of elements that formed a visual language for representing information stored in computers. The most common combination of such elements is the WIMP ("window, icon, menu, pointing device") model. To carry out commands such as opening files, deleting files, moving files, etc, a GUI uses these elements which can be manipulated by a mouse and/or a keyboard as well. Because of its intuitive design the GUI Operating Systems are a lot easier for end-users to learn and use it. Consequently they have become the dominant operating system used by end-users today. A few examples of a GUI Operating Systems: Microsoft Windows 95, Apple System 7, Apple Mac OS. A few examples of a GUI interfaces are: GNOME, KDE.

Test case generation A process of testing a product that uses a graphical user interface in order to ensure it meets its written specifications is **GUI software testing**. Generally it is done using a variety of test cases but to generate a good set of test cases, the test designer should make sure that their set covers all the system functionality and that the set fully exercises the GUI itself. This task gets very difficult because of the problems due to domain size and sequences in GUI applications. Regression testing is another major problem in GUIs because drastic changes are made in the GUI across different versions of its application.

The domain size is a problem because a GUI application consists of an enormous number of operations each of which has to be tested. The problem with the sequencing is due to the complex dependencies in the GUI application. To explain in detail, some operations of the application may be accomplished only by following some complex sequence of GUI events. For example, opening a file has to be performed by first clicking

on the File Menu, then selecting the Open operation, and then using a dialog box to specify the file name, and then focusing the application on the newly opened window. So, as the number of possible operations increases, the sequencing problem increases exponentially. When manual test generation techniques are used, this problem becomes a serious issue.

Regression testing is another important method of testing. It is the process of testing changes in computer programs to make sure that the older functionality still works with the new changes. Performing regression testing on GUI is another major problem because GUIs may change drastically across different versions of the application. Due to this, a test designed to run on a predefined path through the GUI may not be able to do so if a button, menu item, or dialog box has changed location or appearance.

Many different techniques were proposed for resolving the issues in GUI testing. Memon [10] contributed in mentioning the pitfalls of GUI testing techniques and provided guidelines for the goals that should be met while testing a GUI. White and Almezen [1] proposed a method that concentrated on user sequences of GUI objects and selections which collaborate, called complete interaction sequences (CIS) that produce a desired response for the user. For testing CIS, they utilized a finite-state model to generate tests. They showed that considerable reduction in tests could still detect the defects in the GUI. Their approach was scalable but they could not prioritize testing related to the CIS testing when the time was a constraint. Also, in case of very complex GUI system CIS configurations, they could not find additional reduction transformations and components to further reduce the required testing.

Another technique was presented by Svetoslav, Ganov, Killmar, Khurshid, and Perry [2] who used symbolic execution to obtain data inputs and enumerated event sequences that were likely to maximize code coverage of a GUI application. By symbolically executing the code of GUI event handlers, they generated data inputs that maximized code coverage while minimizing the number of tests needed to systematically check the GUI. They proved that it provided significantly better performance compared to random input generation, in terms of statement and branch coverage. However, it placed architectural limitations on system designers. In order to present a promising approach for systematic testing of GUIs, they proposed an idea of combining their approach with other existing frameworks.

Another approach for test generation was presented by Qian and Jiang [3]. They presented an event interaction structure to model a GUI and an algorithm to generate GUI interactive test cases. Their method proved that, under the condition of assuring event-based coverage rate, the number of effective GUI interactive test cases generated were about 10 percent of the number of test cases generated by the permutation and combination methods.

Automating GUI test generation is another major area of study. As manual creation of test cases and their maintenance, evaluation, and conformance to coverage criteria is very time consuming, automation be-

came a focus. Memon, Pollack, and Soffa [4] presented an automated test case generation that derived hierarchical GUI operators, identified tasks and generated test case. They showed that hierarchical GUI performed 10 times better than a single level one because of its abstraction. However, if the structure or the tasks were poor, it did not produce good results. Memon and Xie [5][6] have done a lot of work on automated testing of GUIs. They have collaborated with Nagarajan [7] to work on automating the regression testing of GUIs. White, Almezen, Sastry [8], and Soffa [9] have also contributed to the study of regression testing of GUIs.

Limitations

A lot of research has been done on testing of GUIs but we can only find a few researchers who have focused on GUI test case generation. Usually, a software test automation process would include selecting and generating test cases, building the test oracle, executing and validating the results. Our main focus is on selection and generation of test cases. Manual creation and maintenance of test cases is an extremely time consuming process. So, some automation is required when testing GUIs. Most of the previous researchers have described automating the process of test case generation but there is very little research on automating the process of selection of test cases.

Memom. A, Pollack and Soffa [4][11] presented an automated technique of generating test cases in a GUI but their test case generator was largely driven by the choice of tasks that were manually chosen by the test designer. Choosing tasks was nothing but selecting the test cases that are to be generated. So, a poorly chosen set of tasks would result in a test suite that does not provide sufficient coverage. An interesting open problem here is automating the process of choosing these tasks in order to enhance the GUI coverage. Another technique presented by these researchers developed an automated GUI test oracle [12] which automatically derives the expected state sequences and compares the actual and expected states after each action in the test case. They could not automatically generate the preconditions and effects of the operators from a GUI's specifications. There are few other techniques that were proposed, but then had limitations such as the ones that placed architectural limitations on system designers [2], and those that influenced the maturity of GUI interactive test suite [3].

3. Challenges and Goals

The key open issue that we are going to work on is automation of the process of choosing the tasks for GUI test case generation.

Generally, the test designers describe the tasks or the scenarios by defining a set of initial and goal states for test case generation but when these tasks are manually chosen [4][11], there is a good chance that the set of tasks chosen will result in a test suite that does not provide sufficient coverage i.e. the test suite may not be able to cover the entire GUI input space and it may also have a problem sustaining the complex interdependencies in the GUI system.

Our goal is to automate the process of se-

lection of the tasks in such a way that it provides an adequate coverage of the GUI application. We plan to generate a method that would create tasks for test case generation that would effectively cover the entire GUI input space as well as the complex interdependencies in it. By this kind of automation we also hope to make the process of Regression testing much easier and faster.

4. Proposed Approach

Our proposal presents a novel technique for automating the task selection process in GUI test case generation.

Graphical user interfaces consist of many graphical objects like buttons, labels, textboxes, or lists with which users can interact. In this proposal, we use the word-**control** as a standard term for any such graphical object that a GUI application may contain. Here, we intend to study the GUI with respect to its controls and structure and classify its controls based upon two aspects: whether they are interlinked with another control(s) or if they exist independently in the GUI. An example of the controls classified as either the independent controls or a set of interlinked controls is given in Table 1.1.

Independent Controls	Set of Interlinked Controls
Title	Set1 = {Link 1, Link 2}
Image1	Set 2 = {Link2, Text 1}
Image2	Set 3 = {Image3, Link2}

Table 1.1 *We consider a GUI that is a web page that has a Title which is an independent control, 3 images (Image1, 2, 3 out of which Image 3 is associated with Link 2), 2 links and 1 Text area (Links 1, and 2 such that Link 1 navigates to Link 2 and Link 2 is associated to Text area: Text 1).*

Independent controls are easy to test as they have no interdependencies with any other objects in the GUI i.e. these controls would not affect other controls. So, individual test cases could be automatically generated on each of these controls using any of the previous approaches [4] that were proposed. Whereas testing the controls that are interlinked is not that simple. For generating tasks that test such controls, we first intend to find paths between each of the controls that are interlinked and then find the overall set of linearly independent paths that cover all the interlinked controls in the GUI application. Linearly Independent paths include a unique segment that is not covered in the other path. By considering such paths, we would create a set of sequences of controls that are valid and not redundant. Once we are done with extracting all the linearly independent paths we can use each of them as a separate sequence of tasks on which the automatic test case generation [4][11] could be performed. By such automation of creating tasks we can confidently state that the entire GUI space will be covered and we could also be certain that none of the interdependencies have been left uncovered. Illustrations of the paths formed from the interlinked controls in table 1.1 are demonstrated in figures 1.1, 1.2, and 1.3. The set of linearly independent paths that are derived from these paths are shown in Table 1.2 and their illustrations are shown in figures 2.1 and 2.2. The technique we propose is illustrated as an algorithm below.

Our Algorithm:

Begin
Parse all controls in the GUI application,
Classify them as independent or set of interlinked controls
If control is independent
    Test it individually
Else
For each set S of interlinked controls
    Make a path that connects each control in the set S
After all paths are formed, derive all the sets of linearly independent paths – set LIP
For each set in LIP, consider each set as a sequence of tasks to be tested
End

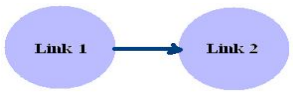

Figure 1.1 *Path 1 for Set1 = {Link 1, Link 2}*

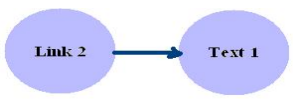

Figure 1.2 *Path 2 for Set 2 = {Link2, Text 1}*

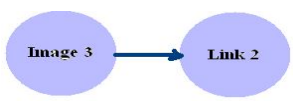

Figure 1.3 *Path 3 for Set 3 = {Image3, Link 2}*

Set of Linearly Independent Paths
L.I.Path 1 = {Link 1, Link 2, Text 1}
L.I.Path 2 = {Image3, Link2, Text 1}

Table 1.2 *Linearly independent paths derived from the paths of interlinked controls given in Table 1.1. L.I.Path represents linearly independent path.*

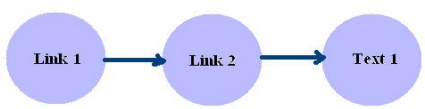

Figure 2.1 *Linearly independent (L.I.) Path 1: includes controls in Set 1 , Set 2.*

Figure 2.2 *Linearly independent (L.I.) Path 2: includes controls in Set 2 , Set 3.*

5. **Evaluation Plan**

Our main research questions regarding this work would be:

- Would the test cases generated for the auto generated tasks be effective in covering the GUI input space?
- Would the auto generated tasks be effective in covering the complex interdependencies in the GUI system?

To address these questions we will perform an evaluation that compares our approach with an approach that manually creates tasks. Both of these methods will be followed by an automatic generation of test cases, execution and validation of results. The automatic test case generator will be chosen from any of the previous research work [4][11]. We will perform an experiment on about 10 GUI applications which consist of different kinds of applications like Web, Java or .Net applications. We will also choose different sizes of each type of an application to make a thorough study of our evaluation. To check the effectiveness of the approach in covering the GUI input space we will compare the results that show the total number of controls covered in each of the applications by each of the approaches. Similarly, to check the effectiveness of the approach in covering the complex dependencies we will study and compare the results that show a list of the sequences of controls

that have been covered for each of the applications by each of the approaches.

6. **Summary of foreseen contributions**

   From the method we proposed in this proposal, we anticipate the following contributions:

   - We present a novel approach to automate the process of selecting the tasks for GUI test case generation
   - When developing the test cases, our technique ensures that it has covered the entire GUI input space.
   - Our technique also ensures that all the complex interdependencies have been covered in the automatically developed test suite.
   - It also helps the process of regression testing by having a mechanical process of test case development that is fast and easier for further repeated testing of newer versions of the GUI application.
   - Our work helps the society by providing a fully GUI test automated framework. This technique minimizes the need for human involvement by automating a large portion of the software testing process. By effectively utilizing our approach, a lot of time and energy can be saved.

References

[1] Generating test cases for GUI responsibilities using complete interaction sequences by WHITE, L. AND ALMEZEN, H. In Proceedings of the International Symposium on Software Reliability Engineering, IEEE Computer Society Press, Los Alamitos, CA, 110121 in 2000.

[2] Test generation for graphical user interfaces based on symbolic execution by Svetoslav R. Ganov, Chip Killmar, Sarfraz Khurshid, Dewayne E. Perry in Proceedings of the 3rd international workshop on Automation of software test in May, 2008.

[3] An event interaction structure for GUI test case generation by Siyon Qian and Fan Jian Beijing, China in 2nd IEEE International Conference on Computer Science and Information Technology in 2009.

[4] Hierarchical GUI test case generation using automated planning by MEMON, A. M., POLLACK, M. E., AND SOFFA, M. L. in IEEE Trans. Softw. Eng. 27, 2 (Feb.), 144155 in 2001.

[5] Using a Pilot Study to Derive a GUI Model for Automated Testing, by Xie, Q., and Atif M. Memon, in ACM Trans. on Softw. Eng. and Method in 2008.

[6] Designing and comparing automated test for GUI-based software applications by Xie, Q., and Atif M. Memon, in 2007.

[7] Automating regression testing for evolving GUI software by MEMON, A., NAGARAJAN, A., AND XIE, Q in 2003 International Conference on Software Maintenance: The Architectural Evolution of Systems.

[8] Firewall regression testing of gui sequences and their interactions by WHITE, L., ALMEZEN, H., AND SASTRY, S in Proceedings of the International Conference on Software Maintenance in 2003.

[9] Regression testing of GUIs by Memon and M. L. Soffa in Proceedings of the 9th European software engineering conference held jointly with 11th ACM SIGSOFT international symposium on Foundations of software engineering in 2003.

[10] GUI testing: Pitfalls and process by Memon, A.M. In Proceedings of the International Conference on Software Maintenance 2003.

[11] Using a goal-driven approach to generate test cases for GUIs. In ICSE 1999, by Memom, A.M., Pollack, M. E., and Soffa, M. L.

[12] Automated Test Oracles for GUls by Memom, A.M., Pollack, M. E., and Soffa, M. L. In Proceedings of the 20th ACM international Conference on Automated software engineering in 2000.