International Journal of scientific research and management (IJSRM)
||Volume||3||Issue||3||Pages|| 2236-2238||2015||
Website: www.ijsrm.in ISSN (e): 2321-3418

Model Based Reverse Engineering With Restructuring Algorithm

Ms.G.S.Dhurgalakshmi[1], Ms.P.Ramya[2]

[1]Department of Computer Science and Technology,
Sona College of Technology
Salem - 636005
ammusrinivasan1992@gmail.com

[2]Department of Computer Science and Technology,
Sona College of Technology
Salem-636005
shriramyabe@gmail.com

Abstract: *GUI testing represents a significant amount of the overall testing efforts. Performing software testing through GUI in order to find defects in the application. In GUI it is more difficult than testing the application through its API because it requires additional programming effort to simulate user actions to observe the output produced and to check its correctness. The incorrect behavior of Graphical User Interfaces can compromise the effective use of the overall software application. One way to discover defects and increase the quality of GUIs is through testing. Test cases can be created manually or produced automatically from a model of the GUI. The size and complexity of GUIs makes it unpractical to do extensive manual testing. The GUI is exercised by a combination of manual and automatic exploration, and information about its structure and some of its behavior is automatically extracted, resulting in an incomplete GUI model.*

Keywords: Refactoring approach, GUI testing, Test generation and execution**.**

1. Introduction

Reverse engineering is the process of extracting knowledge or design information from anything man-made. The process often involves disassembling something (a mechanical device, electronic component, computer program, or biological, chemical, or organic matter) and analyzing its components and workings in detail. The reasons and goals for obtaining such information vary widely from every day or socially beneficial actions, to criminal actions, depending upon the situation. Often no-one's intellectual property rights are breached, such as when a person or business cannot recollect how something was done, or what something does, and needs to reverse engineer it to work it out for themselves.

Reverse engineering is also beneficial in crime prevention, where suspected malware is reverse engineered to understand what it does, and how to detect and remove it, and to allow computers and devices to work together ("interoperate") and to allow saved files on obsolete systems to be used in newer systems. Used harmfully, reverse engineering can be used to "crack" software and media to remove their copy protection, or to create a (possibly improved) copy or even a knockoff; this is usually the goal of a competitor.

Reverse engineering has its origins in the analysis of hardware for commercial or military advantage. However, the reverse engineering process in itself is not concerned with creating a copy or changing the artifact in some way; it is only an analysis in order to deduce design features from products with little or no additional knowledge about the procedures involved in their original production. In some cases, the goal of the reverse engineering process can simply be a re-documentation of legacy systems. Even when the product reverse engineered is that of a competitor, the goal may not be to copy them, but to perform competitor analysis.

2. Related Work

In Tomi Raty et.al This paper propose to reduce the required manual effort and expertise in creating the models for MBGT by using dynamic reverse engineering to automate a significant part of the modeling process. In this paper we compare various approaches for automated GUI modeling and present the results of an empirical tool study, propose a GUI component classification suitable for GUI automation and present some examples of GUI automation strategies for efficient modeling of GUI applications.

In Paul Strooper et.al This paper proposes the Action-Event framework (AEF), another PAM-based approach. It is a two-layer approach. At the top layer is an action model which defines abstract actions. At the bottom layer is a mapping model, which maps abstract actions to sequences of concrete GUI events that implement the actions. An example of an abstract action in MS WordPad is opening a file which can be implemented as a sequence of GUI events such as click on menu File, click on menu item Open, and so on. As there are far fewer abstract actions than GUI events, the effort for defining an action model is also less than for defining an event model.

In Wei Yang et.al The model design is inspired by the UI design principles espoused by the Android team. The Android User Experience Team suggests that developers should make places in the app look distinct" to give users confidence that they know their way around the app. In other words, different screens of the app should and typically do have stark structural differences not just minor stylistic ones. In addition, we would like to capture and reflect important differences such as a button being enabled or disabled. Such differences are reflected in the attributes of GUI components that support user actions.

In Qing Xie et al. The primary goal of this paper is to develop efficient model-based GUI testing techniques that provide the best combination of fault detection effectiveness and cost. The characteristics of proposed framework are it will be automated so that the tester's work is simplified. The GUI model will be obtained automatically. Each module will use the model for automated testing. It will be efficient so that practitioners can use the framework even in the presence of tight deadlines. It will be extensible so that new techniques can be implemented and packaged as new modules of the framework. It will be general enough to be applicable to a wide range of GUIs.

3. Formal Models for GUI Testing

Reverse engineering approaches can be roughly divided into two categories:

1. Static approaches
2. Dynamic approaches

In static approaches the source code or other static representation of the system is analysed without executing the system. In dynamic approaches the system is executed and its external behaviour is analysed. Approaches for static source code analysis are already available in many software engineering tools.

Static approaches are well-suited for extracting information about the internal structure of the system and dependencies among structural elements, but require access to the source code of the system. The dynamic nature of object-oriented programs makes it very difficult to understand the behaviour by just examining the source code. However there are some approaches for static analysis of GUI software and presents an approach and GUI Surfer tool for reverse engineering source code of GUI applications written in Java or Haskell, and creating finite state machine (FSM) models representing the behaviour of the GUI. First an Abstract Syntax Tree (AST) is created by parsing the source code. Then code slicing is used to extract the GUI related parts of the code and a FSM model of the GUI is automatically created. Although the generated GUI models are not used for automated testing, the goal is that these models might be used to reason about the quality of the system.

4. Reverse engineering and Restructuring process

To implement the level of GUI automation that is required for dynamic reverse engineering of GUI applications, the analyzed GUI widgets have to be classified. The classification enables a different kind of handling for different types of widgets and using more advanced strategies in choosing in which order the enabled GUI widgets should be selected for interaction and introduces two different classification examples for abstracting GUI widgets to create platform-independent user interface specifications. Because neither of them was well suited for dynamic GUI reverse engineering, a new GUI widget classification was required.

We divided GUI widgets into four groups:

1. GUI controls,
2. GUI options,
3. GUI inputs, and
4. GUI infos.

GUI controls are widgets that the user selects or presses and a state transition, i.e., a change of GUI state is usually expected. For example buttons and menus belong to GUI controls. In some applications the user has to double-click a control to activate the corresponding action, and usually selecting a menu opens a list of menu items and another selection is required from the user to trigger the action from a menu item.

GUI options are widgets that are used to make a selection from a list of choices, such as a group of radio buttons, a list of items, a combo box or a pop-up box, or simply an on/off switch, such as a check box. Usually selecting an option does not trigger a change of GUI state. However, sometimes the selection of an item on a list enables a button or changes the values of other widgets shown on the GUI.

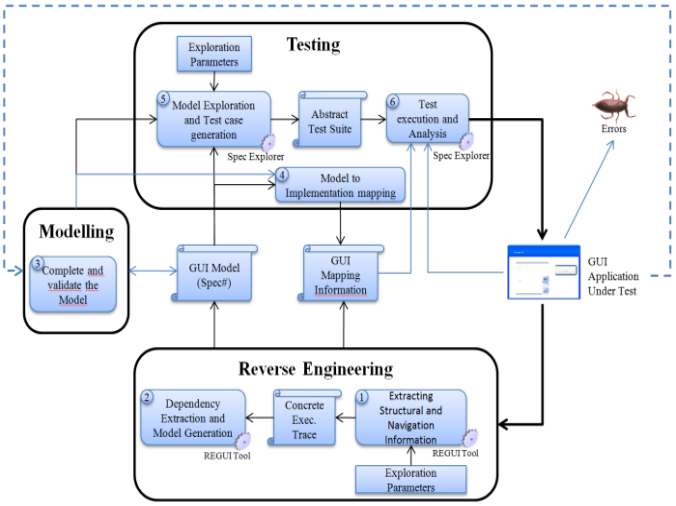

Figure 1: Reverse engineering, Testing and Refactoring process

5. Conclusion

Automated GUI testing has become tremendously important as GUIs become progressively more complex and popular. One way to automate and systematize more the GUI testing process is to generate automatically test cases from GUI models. However, the manual construction of these models requires a lot of effort. We have presented a new technique that, through a reverse engineering process, allows obtaining a model of a GUI. This model is kept in a XML file from which a Spec# specification is generated for testing purposes. However, it is possible to translate the XML file to another language if desirable. The reverse engineering process proposed combines automatic with manual exploration which solves some of the "blocking problems" found in the approaches described in the state of the art section. In our experiments, by using the REGUI tool, 50% of the Specification model was generated automatically. In the future, it is our intention to implement new algorithms to extend the set of dependencies among GUI controls that can be found automatically and translated into Specification.

References

[1] Microsoft. UI Automation. msdn 2009 [cited 2009; Available from: http://msdn.microsoft.com/en-us/accessibility/bb892133.aspx
[2] Paul Hamill, "Unit Test Frameworks", O'Reilly Media, p216, November 2004, ISBN: 9780596006891.
[3] Hendrickson, E., "Making the Right Choice" in Software Testing & Quality Engineering. 1999. p. 21-25
[4] Nyman, N., "Using Monkey Test Tools", in Software Testing & Quality Engineering. 2000. p. 18-21
[5] Memon, A.M., M.L. Soffa, and M.E. Pollack, "Coverage criteria for GUI testing", in Proceedings of the 8th European software engineering conference held jointly with 9th ACM SIGSOFT international symposium on Foundations of software engineering. 2001, ACM: Vienna, Austria
[6] Memon, A.M., M.E. Pollack, and M.L. Soffa, "Using a goal-driven approach to generate test cases for GUIs", in Proceedings of the 21st international conference on Software engineering. 1999, ACM: Los Angeles, California, United States
[7] Paiva, A.C.R., et al. "A Model-to-implementation Mapping Tool for Automated Model-based GUI Testing". in ICFEM'05. 2005
[8] Chikofsky, E.J. and J.H. Cross II, "Reverse Engineering and Design Recovery: A Taxonomy", in Software, IEEE. 1990. p. 13-17
[9] Moore, M.M. "Rule-based detection for reverse engineering user interfaces." in Proceedings of the Third Working Conference on Reverse Engineering. 1996
[10] Mori, G., F. Paterno, and C. Santoro, "CTTE: support for developing and analyzing task models for interactive system design." Software Engineering, IEEE Transactions on, 2002. 28(8): p. 797-813

Author Profile

<Author Photo>

G.S.DHURGALAKSHMI received the B.Tech degree in Information Technology from Sona College of Technology in 2013 respectively. Now I am doing M.E. Software Engineering in Sona College Of Technology.