Volume 5, Issue 7, July 2015 ISSN: 2277 128X
International Journal of Advanced Research in Computer Science and Software Engineering
Research Paper
Available online at: www.ijarcsse.com

Optimization of Testing Techniques in Component Based Systems

Deepika Monga*
Student (CSE Deptt) Ganpati Institute of Management and Technology, India

Narender Rana
Assistant Professor (CSE Deptt) Ganpati Institute of Management and Technology, India

Abstract: ***Today complex, high quality computer based system must be built in very short time period. This mitigates towards a more organized approach to reuse. Component Based software engineering is changing the way large software systems are developed. It embodies the "buy, don't build" philosophy. The component based software engineering is based on the following principle: "Software reuse is the process of creating software systems from existing software rather than building them from scratch." This new approach is different from the traditional approach in which the software systems can only be build from the scratch.***

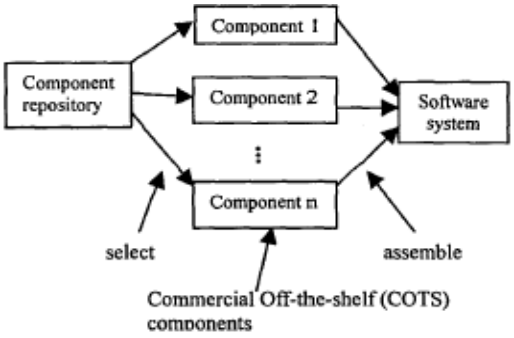
Keywords: GUI (Graphical user Interface), testing, Regression Test Case Modeler (RTCM), Component Based Systems

I. INTRODUCTION

CBSE can reduce development cost and time to market, and improve maintainability, reliability and overall quality of the software systems. Even the life cycle of CBSD is quiet different from the traditional one.

At the foundation of CBSE approach is the assumption that certain parts of large software systems reappear with sufficient regularity that common parts should be written once, rather than many times, and that common systems should be assembled through reuse rather than rewritten over and over .

The following figure shows the component being checked out from a component repository and assembled into target software component



A. Components and its Basic Elements

Real-Time Components
Common off the Shelf Components (COTS)
Other kinds of Components

- A JAVA class.
- A cluster of JAVA classes, with a particular class serving as the exported interface, and the other classes functioning as part of the implementation.
- A Windows DLL.

Different steps in Component Based System Development are

- Find components which may be used in system**.** Vast number of possible candidate components and appropriate tools for finding them must be available.
- Select the components which meet the requirements.
- Alternatively create a proprietary component to be used in the system. However the components that include core functionality of the product are likely to be developed internally as they should provide competitive advantage of the product.
- Adapt the selected component so that they suit the existing component model or the specification of the requirement**.** Some components can be directly integrated into the system, some would be modified through parameterization process, some would need wrapping code for adaptation etc.

- Compose and deploy the components using a framework for components.
- Replace earlier with later versions of the components. This corresponds with system maintenance. Bugs may have been eliminated or new functionality added.

II. PROPOSED WORK

Test cases contain the input to be supplied to the software being tested. The test case in GUI would be mouse clicks, performing events through mouse, selecting events and providing text input through GUI. The test case generation is normally done in GUI through capture/replay tool. But generating test cases through capture/replay tool is time consuming as user has to actively perform the events on the GUI so that capture tool can record the events. Later those events can be replayed on the GUI through replay tool. Using this approach has a flaw, as user can always miss some important events. As it will take a long time to generate test cases. These limitations shifted the manual test generation approach to automatic test generation. Record/play back has some issues in that any slight change in the GUI functionalities requires rerunning all test scripts and an experienced detailed editing. When the GUI changes, input sequences previously recorded may no longer be valid.

For automatic test case generation a model is capable of representing GUI needs to be developed and some finite state machine models (FSM) were proposed earlier. In these FSM, each input event triggers the state of GUI. Therefore a path in FSM represents a test case. But the GUIs today are very large having large number of events and components, there is no doubt that automatic test case generation is easy with FSM but these have scaling problem, as are not easily scalable.

All criteria's can be used to generate those test cases.

To test a GUI, testers create test cases consisting of sequences of GUI input events. The tester needs to answer the questions:

(1) How many test cases should be generated?
(2) What should be the length of each test case?
(3) What events should be put into a test case?

Algorithm: event _coverage (T)

```
{
 T: event forest
 filename: is name of file where test cases need to be stored
 i: top-level component of event forest
 for each (i in T)
    {
       ichild_set=children of i;
        for each (event in ichild_set)
                    {
                   Wrtite_in_file (event);
                    If (event (son) is not empty)
                          {
                            for each (son in event)
                              write_in_file (son);
                          }
                    }
     }
}
```

Algo 1: Algorithm to generate Length one test cases

Test cases of any length can be generated with little changes in the above algorithm. In above algorithm we have enumerated single node at a time to generate length one test cases or event coverage. If we want to generate length two test cases we just need to consider two successive nodes at a time so in this way we can easily generate event sequence for event interaction coverage. Similarly test cases of any length can be generated.

C. Implementation with WordPad 5.1 (XPSP2)

The algorithms for constructing the event forest and generating the test cases were implemented on WordPad 5.1 menu component. For implementing these algorithms the first task which needs to be done is

- Identifying the components and events (figure 6.12)
- Creating event forest(Figure 6.10: Algorithm to Construct Event Forest)
- Applying algorithms for test data generation.

Table 1. Showing number of test cases.

Component Main	Length one Test Case	Length Two Test Case
Main	102	592
File	28	175

Black box test cases for whole WordPad at higher level = GUI's*components*events
For WordPad it counts to be = 1*6*55*18

If we move at lower level that is we include all sub GUI's events this number will increase drastically. But it's not possible to test all these number of test cases.

Regression Test Case Modeler (RTCM) is a framework that generates the test cases automatically from event forest and also generates regression test suite.

RTCM focus on the development of regression test suite from a chosen old test suite which represent correct input and is necessary to validate the modified software. When the structure of the original GUI is modified, test cases from the original GUI are either usable or unusable on the modified GUI. Test cases which can't be used in modified GUI are discarded and test cases which can be used in modified GUI are kept. Test cases which can't be rerun are known as obsolete test cases

III. CONCLUSION

We started with the study of component-based software engineering; engineering process of Component based system and some of the existing testing techniques and then implemented some of the techniques. Next comes the need to generate test cases from the event forest based on various testing criteria's and coverage criteria and test case generation was automated with the help of algorithms which can traverse the event forest in different ways to generate variety of test cases like length one , length two test cases and smoke test cases etc. Since test case generation is a costly job so the RTCM framework helps reducing this cost by providing the concept of reusability. The algorithms for all the modules are included in this thesis

IV. FUTURE SCOPE

In this dissertation the area of research was GUI's but all the algorithms and techniques discussed apply on static components. Dynamic objects are another thing which keeps on changing like we see on some of the website, text logo's keep on changing in the same area.

Algorithms discussed in this dissertation are not extended able for dynamic objects handling so another important area of research is testing of GUI applications having dynamic objects and components.

ACKNOWLEDGEMENTS

I express my gratitude to all those who have encouraged me for my thesis. I would like to thank my thesis supervisor Er. Narender Rana, Head of Department, Computer Science and Engineering, Ganpati Institute of Technology and Management, who always helped in my work and fulfills all essentials.

.He has been highly available throughout the whole process of my thesis. I would be thankful to my family to be there as my biggest support .

REFERENCES

[1] Jim Q Ning, "Component-Based Software Engineering", Proceedings of U.S National Institute of Standards and Technology's Advance Technology Program on Component Based Software, Document number 0-8186-7940-9/97, 1997, pp. 34-43.

[2] David S. Rosenblum, "Adequate Testing of Component-Based Software", Technical Report 97-34 Department of Information and Computer Science University of California, Irvine, CA 92697-3425, 11 Aug 1997.

[3] Alan W.Brown, Kurt C.Walinau, "The Current State of CBSE", In Proceedings of IEEE Software, Document number 0740-7459/98, September/ October 1998, pp. 37-46.

[4] William T.Councill, "Third-Party Testing and the Quality of Software Components", Proceedings of IEEE Computer Society, Document number 0740-7459/99, July/August 1999, pp.55-57.

[5] Xia Cai, Michael R.Lyu, Kam-Fai Wong and Roy Ko, "Component-Based Software Engineering: Technologies, Development Frameworks, and Quality Assurance schemes", IEEE Computer Society, 1530-1362/00, 2000, pp.372-379.

[6] Sami Beydeda and Volker Gruhn, "An Integrated testing Technique for Component-Based Software", IEEE Computer Society, 2001.

[7] Philip T Cox and Baoming Song, "A formal Model for Component-Based Software", IEEE Computer Society, Document number 07695-474-4/01, 2001, pp.304-310.

[8] A. M. Memon, M. L. Soffa, and M. E. Pollack, "Coverage criteria for GUI testing", in Proceedings of the 8th European Software Engineering Conference (ESEC) and 9th ACM SIGSOFT International Symposium on the Foundations of Software Engineering (FSE-9), Sept 2001, pp. 256–267.

[9] A. M. Memon, "GUI testing: Pitfalls and process", IEEE Computer, vol. 35, no. 8, pp.90–91, Aug 2002.

[10] Marcel Dix. Holger and D. Hofmann, "Automated Software Robustness Testing", proceedings of the 28 th Conference (EUROMICRO'02) IEEE, Document number 1089-6503/02, 2002, pp.1-6.

[11] A. M. Memon, I. Banerjee, and A. Nagarajan, "GUI ripping: Reverse engineering of graphical user interfaces for testing", in Proceedings of the 10th Working Conference on Reverse Engineering, November 2003, pp. 260–269.

[12] Muthu Ramchandran, "Testing Software Components using Boundary Value Analysis", proceedings of the 29th EUROMICRO conference New

