ISSN (Print) : 2319-5940
ISSN (Online) : 2278-1021

International Journal of Advanced Research in Computer and Communication Engineering
Vol. 2, Issue 6, June 2013

Component Based System and Testing Techniques

Neelam Sirohi[1], Anshu Parashar[2]

Research Scholar, Department of CSE, H.C.T.M Kaithal, India [1]

Associate Professor, Department of CSE, H.C.T.M Kaithal, India [2]

Abstract: Today Component Based Software Engineering (CBSE) is more generalized approach for software development. Component based software systems are developed from build in components or third party components. Hence the quality of system depends on the quality of involved components. To ensure the quality of component based system testing is necessary. In this paper we discuss component based software engineering and some testing techniques. We notice that third party components include only specification or we can say that black box components. Due to black box nature of components source codes are not available then to test the black box component is a very difficult task. That's why several testing techniques have been proposed by several authors for testing the component based systems. Some of them are discussed by us in this paper.

Keywords: Component Based Software, GUI Component Testing, Testing Techniques, RCTM Framework

I. INTRODUCTION

Component Based Software System (CBS) are mainly constructed from reusable components such as third party components and Commercial-Of-The-Shelf (COTS) components. Due to this, component based systems are developed quickly with minimum engineering efforts and resource cost.

Component Based Software Engineering covers many software engineering disciplines and different techniques. Many of them have been developed and successfully implemented. CBSE has been successful in certain engineering domains, such as office applications and distributed internet-based applications but it is still in the early stage of utilization in many other domains, in particular those which have specific requirements on different quality attributes [1].

Component-Based Development (CBD) offers a radically new approach to the design, construction, implementation and evolution of software applications. Software applications are assembled from components from a variety of sources; the components may be written in several different programming languages and run on several different platforms or we can say that components are heterogeneous in nature.

To ensure the delivery of quality component based software, effective and efficient testing is the key process in software development. Component Based Software has implementation transparency property and its frequent heterogeneity raise some difficulties when the software is to be tested. Since components are intended to be reused across various products and product-families, components must be tested adequately.

Graphical user interfaces (GUIs) are by far the most popular means used to interact with today's software. Most of the software's being used today is GUI based and the reliability and robustness of the software depends on how each component works as GUI is one of the most important component so it needs to be tested for functional correctness.

The characteristics of the GUI are different from that of conventional software as it is completely event based and hierarchical in nature. So a different representation "event forest" is created which is completely capable of simulating the hierarchical nature of the GUI and also stores the event interaction information.

Most of the software's today have GUI front-end and the newer versions are released year by year. The newer versions have modifications as compared to previous version. So to test the modified GUI, regression testing is performed which reuses the test cases from the old GUI.

In this research we present component based software system and some testing techniques, their advantages and disadvantages addressed by the researchers. Next section covers the component based software engineering. Section III and IV covers the testing and testing techniques and then section V is conclusion.

II. COMPONENT BASED SOFTWARE ENGINEERING

Component Based Software Engineering represent a new development paradigm: assembling software systems from components .Today complex, high quality component based system must be built in very short time period. This mitigates towards a more organized approach to reuse.

ISSN (Print) : 2319-5940
ISSN (Online) : 2278-1021

International Journal of Advanced Research in Computer and Communication Engineering
Vol. 2, Issue 6, June 2013

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." Today the trend in computer-based products, such as cars and mobile phones, is shorter and shorter lifecycles. As a consequence, time spent on development of new products or new versions of a product must be reduced. One solution to this emerging problem is to reuse software design and solutions in new versions of systems and products. Besides shortening development time, properly handled reuse will also improve the reliability since code is executed for longer time and in different contexts. However, reuse is not trivial and puts strong demands on development methods in order to be successful. When applying reuse in development of real-time systems, methods gets even more complex since both functional behaviour and temporal behaviour must be considered. Software systems are assembled from components. Mainly three improvements in computer and software-based systems can be expected by CBSE approach to system development.

- Improvement of system quality
- Achievement of shorter time-to-market
- Improved management of increased complexity of software

A. Component Based Software Development

The development of Component-Based Systems introduces fundamental changes in the way systems are acquired, integrated, deployed and evolved. This approach is based on the idea that software systems can be developed by selecting appropriate off-the-shelf components and then assembling them with well-defined software architecture. This new approach is different from the traditional approach in which the software systems can only be built from the scratch. 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 quite different from the traditional one. The following figure shows the component being checked out from a component repository and assembled into target software component [2].

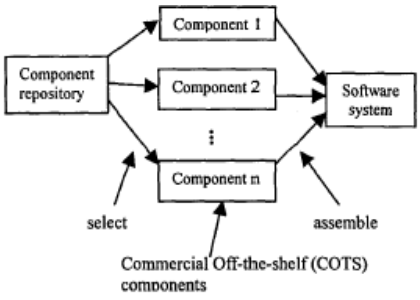

Fig.1 Component Selection Process

Different steps in Component Based System Development are [19]:

- Find components which may be used in system**.** Many appropriate tools are available for finding these components.
- Select the components which meet the requirements.
- Alternatively create a proprietary component to be used in the system. Components which includes core functionality of the product are 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 components with later versions of the components to maintain the system. Later versions are bug free and they include more functionality.

The characteristics of the component-based development are the following:

- Black-box reuse
- Reactive-control and component's granularity
- Using RAD (rapid application development) tools
- Contractually specified interfaces
- Introspection mechanism provided by the component systems

B. Component Based System Architecture

To ensure that a component based software system can run properly and effectively the system architecture is the most important factor. According to research community the system architecture of the component based system should be layered and modular. The top application layer is the one supporting business. The second layer consist of components engaged in only specific business , including components used in more than one application , the third layer is cross business middleware component consisting of common software and interfaces to other established entities and the lowest layer includes basic components that interfaces with the underlying operating system and hardware[2].

ISSN (Print) : 2319-5940
ISSN (Online) : 2278-1021

International Journal of Advanced Research in Computer and Communication Engineering
Vol. 2, Issue 6, June 2013

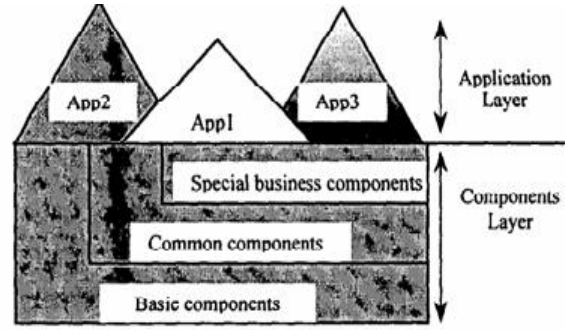

Fig.2 Component Based System Architecture

C. *Engineering Process*

The development cycle of the component based system is different from the traditional one i.e. waterfall, iterative, spiral etc. CBSE covers both component development and system development with component. When developing components other components can be incorporated but the main emphasis is on reusability. A component must be well specified, easy to understand, sufficiently general and easy to deliver, easy to adapt deploy and replace. The components interface must be as simple as possible and separated from its implementation. Development with components is focused on identification of reusable entities and relation between them.

The engineering process consist of the following steps:

- Find components (COTS and non COTS).
- Select the components that are most suitable to the system.
- Create a composed solution that integrates the selected components.
- Adapt the selected component so that they suit the existing component model.
- Compose or deploy the product.
- Replace the old version or maintain COTS and non-COTS parts of the system.

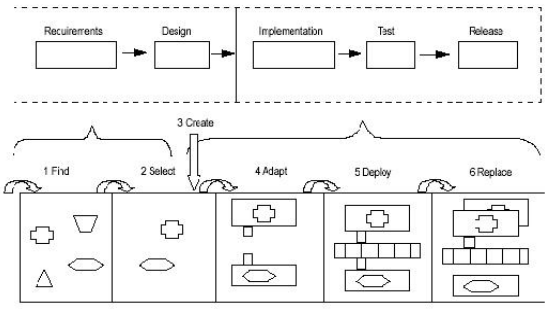

Fig.3 CBSD development cycle compared with waterfall model.

When the software is developed testing must be done to make bug free software.

III. TESTING

In the previous section we study the component based software engineering and development process. Testing is essential in the development of any software system. Testing is required to assess a system's functionality and quality of operation in its final environment. This is especially of importance for system being assembled from any self contained software components. Basically testing is done to reveal faults and after detecting failures, debugging techniques are applied to isolate and remove faults [6].

A. *Testability*

Software testability is how easily a computer program can be tested or we can say it is the effort required to test a program so that it performs its intended function. An ideal testable software component is not only deployable and executable, but also testable with the support of standardized components' test facilities. Unlike normal components, testable components have the following features.

- Testable components must be traceable.
- Testable components must have a set of built-in interfaces to interact with a set of well-defined testing facilities. This reduces the effort of tester to test the component and helps in controlling the *traceability* of the component.
- Although testable components have their distinct functional features, data and interfaces, they must have a well-defined test architecture model and built-in test interfaces to support their interactions to component test suites and a component test-bed.

B. *Component testing vs module testing*

Module testing in traditional software environment, software is divided into modules which are named and addressable components and these are integrated to satisfy the problem requirement. Modularity is a single attribute that allows a program to be intellectually manageable. Once the module has been created we are having its source code so it must be tested to cover as many errors as possible. The goal is to design the series of test cases that have high likelihood of finding errors. These techniques provide systematic guidance for designing test cases that:

- Exercise the internal logic of software module
- Exercise the input and output domain of the program to uncover errors in program.
- Techniques likes Black Box and White box testing can be applied.

Component testing, Component is an independent unit mostly made by third party and so called Commercial Off the Shelf Components(COTS). So it does not have the source code with it. Only specifications are available. So it is very difficult to test the component or apply White box testing. Some techniques have been proposed to test the

ISSN (Print) : 2319-5940
ISSN (Online) : 2278-1021

International Journal of Advanced Research in Computer and Communication Engineering
Vol. 2, Issue 6, June 2013

components. These techniques are defined in next section i.e testing techniques.

- It is difficult to exercise the internal logic of the component.
- Black Box testing can be applied.
- White box testing can be applied only if some of the specifications are available.
- Components are generally heterogeneous and made in different languages.

C. Testing issues

Testing issues are basically the factors which need to be taken in account while testing the system. Some of the issues in testing software component are as follows [8]:

- Redundant Testing during Integration of Components
- Unavailability of Source Code
- Heterogeneity of Language, Platforms and Architectures
- Monitoring and Control Mechanism in Distributed Software Testing
- Deadlocks and Race Conditions

IV. TESTING TECHNIQUES

Testing of component based system is different from normal software testing that is why the testing techniques are also bit different. There are basically two different approaches for testing white box and Black box. The black box testing is more prevalent in the component based systems because in most of the cases the source code is not available with the component; at most only the specification is available.

Some of the techniques are:-

- Adequate testing
- Integrated testing technique.
- Automated software robustness testing.
- Boundary value analysis.
- Self testing of component based software.
- Object oriented component testing.
- Event flow model.
- Regression testing.
- Modular Regression Testing.

A. Adequate Testing

An adequate testing approach for testing component based software was suggested by David s.Rosenblum. This technique provided the initial basis for testing of component based software. The main result of this technique is formal definition of the concept c-adequate –for p for adequate unit testing of the component and the concept c-adequate-for m for the adequate integration testing for the component based system.

B. Integrated Testing Technique

In integration testing technique relevant features of component based software are tested and it allows more rigorous testing. The basic constituent of this approach is graphical representation combining black and white box information from specification and implementation. The graphical representation can then be used for test case generation. In this technique specification of each component is given as component state machine which are special finite state machines which are similar to state machines discussed in theory of computation. The abstract states of the component are represented by circles and each transition is represented as arrow leading from the source state to its target state.

These transitions are 5 tuples {source, target, event, guard, action} where source and target are the starting and final state, event causing transition, a predicate guard has to be fulfilled before the transition can occur.

After the component state machine for all the components are constructed then the overall Component Based System Flow Graph (CBSFG) is constructed which is the basis of the testing technique. It visualizes information gathered from both the source code and the specification. After the graph is generated the structural techniques can be applied to generate the test data. The test data for both the white box and black box testing can be generated.

CBSFG is a directed graph visualizing control and data flow within a component based software, each method of the software is represented by two sub graphs. One of these sub graphs represents a control flow graph of the prototype and the other represent flow graph generated on the basis of source code. The sub graphs are interlinked by the control and data edges which are namely:

- Intra-method control and data flow edges.
- Inter-method data and control flow edges.
- Specification implementation edges.

This technique has one flaw that is very time consuming approach and there is no provision of automatic test case generation. Our next part shows the automatic test case generation testing techniques.

C. Automated Software Robustness Testing

The trend shifted towards automatic testing of software components. Automated Software Robustness testing approach is basically biased towards an automatic testing of software component robustness. Robustness is defined as the components ability to handle invalid input conditions. This technique also focuses on how large number of test cases can be generated from small number of test values. But there is demand for reducing this large mass of test cases so another technique that is static testing is introduced for test case reduction. It reduces the number of test cases to be executed without affecting test accuracy and reliability but it

ISSN (Print) : 2319-5940
ISSN (Online) : 2278-1021

International Journal of Advanced Research in Computer and Communication Engineering
Vol. 2, Issue 6, June 2013

is unable to reduce the numbers of test case and to generate test cases, human efficient expertise is needed.

D. Boundary Value Analysis

In automated software regression testing, large number of test cases are generated but the demand is to reduce the test cases. Boundary value analysis is a methodology for designing test cases that concentrates software testing effort on cases near the limits of valid ranges. Boundary value analysis is a method which refines equivalence partitioning. In equivalence portioning input data is divided into equal parts and then test cases are generated. Boundary value analysis generates test cases that highlight errors better than equivalence partitioning. The trick is to concentrate software testing efforts at the extreme ends of the equivalence classes. At those points when input values change from valid to invalid errors are most likely to occur.

... -2 -1 0 1..... 12 13 14 15.....

----------|----------|----------

Invalid partition1 valid partition invalid partition 2

Fig. 4

Applying boundary value analysis we have to select a test case at each side of the boundary between two partitions. In the above example this would be 0 and 1 for the lower boundary as well as 12 and 13 for the upper boundary.

E. Self Testing of Component Based Software

This technique is widely accepted that conventional test methods are not necessarily adequate for testing of component based software. Also conventional test tools cause same problem for the automation of the test automation of CBS, because the knowledge about the implementation of the CBS are essential to run the tests. The component manufacturer on the other hand is not willing to provide the component source code. This technique introduces the framework for the automation of user oriented component testing that significantly reduces the test cost and it is based on black box testing and utilizes common features of commercial capture / playback tools.

The novelty of this approach is the framework that enables automatic generation of test cases and test scripts without involving domain knowledge and knowledge about object code. However it also reveals a weakness, if the CBS is changed, e.g., produce a new release, the test frame, i.e. the test case, and consequently, the test script might become obsolete. So component model needs to be automatically updated.

F. Object Oriented Component Testing

Another approach Object oriented testing has been suggested by Fakhra jabeen et al. The unavailability of the source code precludes extrapolating standard testing approaches. In this technique object oriented component test framework is proposed that relies on the utilization of discrete descriptors to facilitate test execution and to enable a uniform information flow. The component provider, user and third party tester each have responsibility for descriptor unique to their perspective, eliminating the dilemma of missing testing information. It uses Component test framework descriptor like component descriptor, Component requirement descriptor, Component Test specification descriptor. At present it supports component unit testing and partial integration testing.

G. Event Flow Model

Graphical user interfaces (GUIs) are the most popular means used to interact with today's software. The functional correctness of GUI is of utmost importance and is required to ensure the safety, robustness and usability of an entire software system. GUIs are hierarchical in nature. GUI testing for functional correctness is a challenging research area. One of the most common ways to achieve functional correctness of GUIs is testing.

Event flow model represent events and event interactions. GUI is decomposed into a hierarchy of modal dialogues. This hierarchy is represented as an integration tree; each modal dialogue is represented as an event flow graph that shows all possible event execution paths in dialogue.

Event flow model is not tied to specific aspect of GUI testing process. It may be used to perform a wide variety of testing tasks by defining specialized model-based techniques i.e event forest algorithm. Construction of event forest is based on structure of GUI. Event forest structure automatically generates various kinds of test cases covering number of coverage criteria.

H. Regression Testing

Regression testing is based on the change information or change point**.** Due to lack of information about externally developed component, system testers can't perform effective regression testing on their component based system. Component users don't know the details about the change in component so they aren't able to select the proper test case to test the modified component. This technique provides a regression testing method based on enhanced change information of component version to test the software component to test the software system containing modified components. It needs a joint participation of the component developer and the user. Call for graph is used to calculate the change information. But this testing method needs to be verified by experiments on the large scale and real component based software systems.

ISSN (Print) : 2319-5940
ISSN (Online) : 2278-1021

International Journal of Advanced Research in Computer and Communication Engineering
Vol. 2, Issue 6, June 2013

I. Modular Regression Testing

GUI regression testing is to be performed for every modification; the continuous modification of GUI requires that test cases be reusable across versions, as it would be too expensive to generate new test cases for each version. Modular regression testing provides an advantage over regression testing. In this RTCM is used. RTCM reduces this cost by providing the concept of reusability. 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 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. GUI test cases can be unusable, usable or repairable. In RTCM we have followed the principle "do not throw away unusable test cases". The unrepairable and infeasible test cases are discarded but repairable test cases are repaired and reused. Regression Test Case Modeler capable of constructing regression test suite form old original GUI test suite file. Thus it will produce test case file for modified GUI.

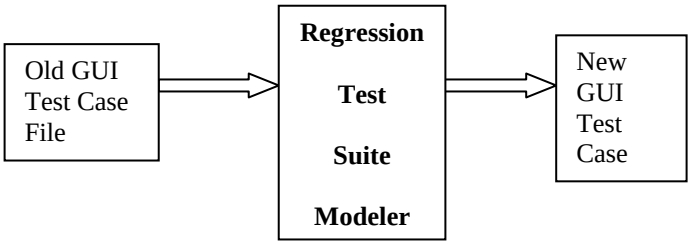

Fig. 5 The RTCM framework

The key component in the RTCM is regression test suite modeler. It takes old test suite file and makes it reusable for modified GUI. It can find out the unusable test cases, usable test cases, infeasible test cases, repairable test cases, unrepairable test cases as well as infeasible test cases.

V. CONCLUSION

When a software system is developed then to ensure the quality, reliability, robustness and functionality of system testing is necessary. In this paper we study about component based software engineering and some testing techniques. Component based software engineering is still at its young stage of life and there is much area for research in this field. From our research we explore some testing techniques. In our future work we will test the components with best one testing tool.

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] 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.

[3] 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.

[4] Alejandra Cechich and Mario Piattini-Velthuis, "*Component-Based Software Enginnering*", proceedings of The European Journal for the Informatics Professional UPGRADE Vol. IV , No 4 , August 2003, pp.15-19.

[5] 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.

[6] 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.

[7] Jerry Gao, Ph.D, "*Component Testability and Component Testing Challenges",* San Jose State University, One Washington Square, San Jose, CA 95192-0180.

[8] Nitin V. Koppalkar, Seshaiah Uppala and Mahesh Madugundu, "*Testing of Component-Based Software Systems",* Philips Research India-Bangalore.

[9] Fu Lingyun, "*An Approach for Component-Based Software Development*", International Forum on Information Technology and Application.2010,pp.22-25.

[10] 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.

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

[12] S Phani Shashank et.al "*A Systematic Literature Survey of Integration Testing in Component-Based Software Engineering*" , IEEE 2010.

[13] Bruce W. Weide, *"Modular Regression Testing": Connections to Component-Based Software*, Dept. of Computer and Information Science, The Ohio State University , 2015 eil Ave. Columbus, OH 43210,USA +1 614 292 1517, 2005.

[14] Fevzi Belli and Christof J. Budnik, "*Towards Self-Testing of Component-Based Software",* Proceeding of the 29th Annual International Software and Applications Conference (COMPSAC '05), IEEE, 2005.

[15] IEEE "*Transactions on Software Engineering*", Proceedings of Software Engineering Conference, 1999(APSEC '99).

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

[17] Muthu Ramchandran, "*Testing Software Components using Boundary Value Analysis*", proceedings of the 29th EUROMICRO conference New Waves in System Architecture" (EUROMICRO 08), Document number 1089-6503/03, IEEE, 2003.

[18] Fakhra Jabeen, Muhammad jaffar-Ur Rehman, "*A Framework for Object oriented Component Testing*", Document number 0730-3157/05, IEEE press, 2005.

[19] Ivica crnkovic, "*Component-Based Software Engineering – New Challenges in Software development*", journal of computing and Information Technology –CIT 11, 3,151-161, 2003, pp.151-160.

[20] A. M. Memon and M. L. Soffa, "*Regression testing of GUIs*", in Proceedings of the 9th European Software Engineering Conference (ESEC) and 11th ACM SIGSOFT International Symposium on the Foundations of Software Engineering (FSE-11), Sept 2003, pp. 118–127.

[21] Leonardo Mariani et.al "*Compatibility and regression testing of COTS-Component-based software"* , IEEE 2007.

[22] Atif M. Memon, "*An Event-flow Model of GUI-Based Applications for Testing*" Department of Computer Science University of Maryland, College Park, MD 20742 , 2007.