Volume 7, Issue 4, April 2017 ISSN: 2277 128X
International Journal of Advanced Research in Computer Science and Software Engineering
Research Paper
Available online at: www.ijarcsse.com

Testing and Quality Assurance of Component Based Software Using Traces

Pallavi
SKITM, M.tech (CSE, Student)
India

Kirti Bhatia
HOD of CSE Department, SKITM,
India

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. By enhancing the flexibility and maintainability of systems, this approach can potentially be used to reduce software development costs, assemble systems rapidly, and reduce the maintenance burden associated with the support and upgradation of large systems. Components re-used across various product and product-families, possibly in different environment, therefore it must be tested adequately. In this paper, we explore the testing and maintenance aspects related to Component- Based Software Systems.

Keywords— Component based software, GUI Component testing, testing techniques

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. To ensure the delivery of quality component based software, it is essential that individual components are tested effectively and efficiently. A lot of research efforts have been devoted to analysis and design methods for component based software.

The most popular means of testing these days id Graphical user interfaces (GUIs). 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.

"Event Forest" is created to simulate the hierarchical nature of GUI. GUIs differ from conventional software in their basic characteristics as they're based on series of events that run and their hierarchical nature. In this research we present some testing techniques, and the issues related to testing. It also mentions the conclusion and future work.

II. TESTING AND TESTING ISSUES IN SOFTWARE COMPONENTS

Testing is critical in the development of any software system. Testing is required to assess a system's functionality i.e. the system is performing what it should intent to perform and quality of operation in its final environment. Basically testing is done to reveal errors, faults and after detecting failures, debugging techniques are applied to isolate and remove errors and faults [3].

Some of the issues in testing software component [1] are as follows:

2.1 Redundant Testing

Software components are tested individually and then in an incremental manner when they are integrated. This is leading to lot of redundant testing which is a waste of efforts. Therefore, it is required that studies are conducted to help testers identify and quantify how much testing is redundant so that it can be eliminated.

2.2 Availability of Source Code

Different testing techniques need to be used for testing based on the availability of the code.

2.3 Heterogeneity of Language, Platforms and Architecture

Since components written in one programming language can be used with components in other programming language and across different hardware & software components, therefore testing techniques, methodology and tools used should not be dependent on the language and platform.

2.4 Monitoring and Control Mechanism in Distributed Software Testing

Testing in distributed software is much more complex than testing in centralized software system because of the monitoring and control mechanism.

2.5 Deadlocks and Race Conditions

Distributed or concurrent systems occasionally have problems related to race conditions and deadlocks. It is desirable to detect such problems while testing.

After listing some of the testing issues of software components, we will discuss about the testability of a software component.

III. TESTABILITY OF SOFTWARE COMPONENTS

Ideal testable software is not only deployable, executable, but also testable with the support of standardized components test facilities [2]. The testable component has the following features.

Testable components must be traceable. Traceable components are constructed with an in-built tracking mechanism for monitoring various component behaviors in a systematic manner.

Testable components must have a set of in-built 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 in-built interfaces to support their interactions to component test suites and a component test-bed.

IV. COMPONENT TRACEABILITY

Traceability of a software component refers to the extent of its build-in capability of tracking the status of component attributes and component behavior [4].

Traceability of a component includes the following two aspects -

4.1 Behavior traceability

It is the degree to which a component facilitates the tracking of its internal and external behaviors.

4.2 Trace controllability

It refers to the extent of the control capability in a component to facilitate the customization of its tracking functions. We can classify the component traces into five types -

Operational trace – it records the interactions of component operations.

Performance trace – it records the performance data and benchmarks for each function of a component in a given platform and environment.

State trace – it tracks the object states or data states in a component. We allowed developer/tester to define state of an object.

Event trace – it records the events and sequences occurred in a component. This trace is mainly helpful for GUI intensive systems.

Error trace – it records the error messages generated by a component. The error trace supports all error messages, exceptions, and related processing information generated by a component.

V. TEST ADEQUACY CRITERIA

Testing is an important part of the software development cycle. The motivation for testing is to raise the quality and reliability of the program. Since, there can be no proof that, at a point of time, the program has no bugs, the question arises - How much testing is sufficient? A test adequacy criterion is a systematic criterion that is used to determine whether a test suite provides an adequate amount of testing for a program under test [5].

Often a testing methodology provides a measure that quantifies "how much" code was tested. The term coverage domain is a set of program related entities that are being accounted for in order to calculate or measuring coverage. Requirements, functions, statements, decisions and decision use pairs are some of the entities used. For example, the set of all functions in a C-program forms a coverage domain.

We can roughly estimate the code coverage for the component by using following expression -

Method Coverage = E_m / D_m

Where, E_m denotes the number of methods executed in the component, D_m denotes the number of methods defined in the component interface. We have calculated the E_m using the operation trace described above.

The adequacy of test cases can also be measured by metering the exception coverage –

Exception Coverage = T_e / D_e

Where, T_e denotes the number of expressions thrown the component, D_e denotes the number of exceptions defined in the component interface. We have calculated the T_e using the operation trace described above.

With exceptions, we want to make sure that, not only all exceptions are thrown, but that they are thrown at each possible condition. This is sufficient for component testing. Additionally, for integration testing, we would like to do this for each internal state of components with which they interact.

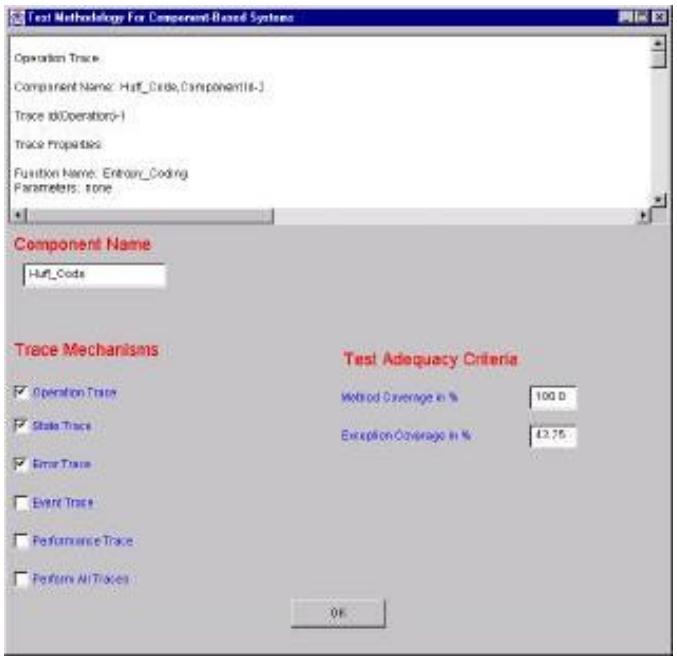
We have calculated the test adequacy for software components (Unit, Integration testing) using the above two expressions.

VI. EXPERIMENTAL SETUP AND PROTOTYPE IMPLEMENTATION

For experimentation, we have identified and developed Java-Beans components [6] for JPEG (Joint Photography Experts Group) [7]. We have implemented the above mentioned five traces for these components. We also

calculated test adequacy for same set of components. We have identified the effectiveness of these traces in locating errors by injecting errors manually into these components.

The prototype is implemented based on event based model. The Graphical User Interface for the prototype is shown below



The GUI has a field named as Component Name, where the user has to enter the component name. Then the user has to select the traces among the five shown as check boxes. After pressing OK button, the selected traces for the particular component will appear in the window. Along with the traces, test adequacy (Method and Exception coverage) "meter" also appears on the screen. The above figure shows the selected traces and test adequacy meter for one of the components, "Huff Code" in JPEG system.

VII. CONCLUSION AND FUTURE WORK

When a software system is developed then to ensure the quality, reliability, robustness and functionality of system testing is necessary. 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 conclude that components are likely to be tested for each new environment. Future work should include consideration of studying the re-use of existing components which can prove to be cost effective such that they can considerably reduce the cost of the software as today's industrial environment requires.

REFERENCES

[1] Sudipto Ghosh, Aditya P. Mathur, *"Issues in Testing Distributed Component-Based Systems"*. Software Engineering Research Center, Purdue University, March 22, 1999.

[2] 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.

[3] Jerry Gao, *"Tracking Component-Based Software Systems"*, San Jose University, CA,2000.

[4] David S. Rosenblum, *"Challenges in Exploiting Architectural Models for Software Testing"*, Proc. International Workshop on The Role of Software Architecture in Testing and Analysis, 1998.

[5] Mila Keren, *"Test Adequacy Criteria for Component-Based Visual Software"*, IBM Haifa Research Lab, 1999.

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

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

