Component Level Testing in Component-Based Software Development

Umesh Kumar Tiwari
Graphic Era University
Department of Computer Science and Engineering
Dehradun

Santosh Kumar
Graphic Era University
Department of Computer Science and Engineering
Dehradun

ABSTRACT
Whether it is traditional software or component-based software, software testing is an essential process of software development. Testing is performed not only to make the underlying software error free but to enhance and assure the development of reliable and quality software product. To measure the cost of testing in terms of effort and time we need some well defined metrics and methodologies. We have a well specified and established level of testing in the context of traditional software. We have unit level to customer acceptance levels of testing in this domain. But in the context of component-based software, we have no specified and clear testing levels for individual components. In this paper we are proposing the strategy for components testing in component-based development.

Keywords
Software testing, Reliability, Unit level, User Acceptance.

1. INTRODUCTION
“Testing is the process of executing a program with the intent of finding errors.” [1]. Testing is an attempt to make the under consideration software error free. Identifying and fixing errors in early phases of development is always helpful to minimize the overall cost of the development. Testing software is an expansive endeavour by means of cost, effort and time [2]. CBSE supports the development of systems as assemblies of components, the developments of, and with reusable entities [3], as shown in Figure 1. CBSE supports the development of software application with the characteristics of reusability. Testing levels of conventional software where the test system is developed from the scratch and then integrated is not suitable for the assessment of software components. Specifically in environments where system is developed with the help of off-the-shelf components in different contexts [4], [5], [6].

2. COMPONENT-BASED SOFTWARE TESTING
Component-based system testing must help not only in removing bugs and errors but also in enhancing quality assurance, productivity and reusability of components in CBS development [7], [8], [9]. As component-based software systems are composed of individual, identifiable pieces of code known as components, we need some techniques to analyse these components as individual as well as the whole integrated system. We simply can’t apply traditional measure and metrics of convention software testing to test the applications developed by CBS paradigm. In this work we propose a testing technique for individual components and a method of testing documentation for these components.

In the first section, we are giving the basic introduction of components and component-based software development, Second section describes the testing requirements of component-based development. Third section is for discussing the challenges and issues of testing in component-based development. In the fourth section, we propose a revisited version of test case process documentation.
3. REQUIREMENT OF TESTING LEVELS FOR COMPONENT-BASED SOFTWARE SYSTEMS

The fundamental concept behind CBSE is to develop new software applications by reusing available code constructs, rather than by developing from first line of code to the end. Component-based software promises savings in overall development cost of the application by reusing components, in terms of development effort and testing than traditional software development.

The basic difficulty with component-based software testing is that when a pre-built, pre-tested and error-free component is integrated with other components, it may fail to provide the intended and expected function. Because these off-the-shelf components may not satisfy the requirements of the application and may be incompatible with existing components since they are developed for some other intentions, and possibly have varied usage contexts. Available testing methodologies may facilitate to validate the interconnectivity of connected components, but they do not provide clear and defined levels of testing of components and levels of integrated software. Software developers may therefore be facing the testing problems in CBS development especially in distributed environment. Although conventional development testing techniques may help to promise the quality of individual components, they cannot guarantee the quality assurance of the integrated applications. Thus there is the requirement of defining clear and well specified testing levels.

4. CHALLENGES AND ISSUES IN TESTING OF COMPONENT-BASED SOFTWARE

It is clear that the verification and validation of component-based systems is different from testing traditionally developed systems. Component-based application testing includes each and every action which is concerned with the testing of specific component and also the test development activities of these components. It also includes testing activities during engineering of applications in early stages of development [10].

Typically components are developed and used in various environments and contexts. Since the component owner/provider can assume only a limited number of scenarios of using the component he can never test that component in every possible context [11], [12]. Normally components are provided as black-box as their internal code is not accessible [11], [13], hence it lessen the controllability and testability.

Fig. 1. Component made up of different sub components
5. PROPOSED COMPONENT TESTING FOR COMPONENT BASED SOFTWARE SYSTEMS

Testing levels of component based software must integrate software testing methods into a pre-defined and planned sequence of actions. Testing methodologies must incorporate planning of testing, test-case design, implementation of testing, testing results and the assessment of collected data. Typically a component is made up of different parts or sub components. And these subparts are not only independent but are from various contexts. Therefore we need a testing technique which must address these components not only in individual but as a whole is required. Conventional testing methodologies are applicable to small scale programs and software. It seems inefficient when we apply these methodologies in the context of component-based software.

5.1 Component testing strategy at individual component level

In component-based software, we integrate components with the help of well specified and unambiguous interfaces. Interfaces provide the platform for the assembling of individual components. First we test these interfaces to make sure that communicated information, and data should be properly convey to and from these components. The internal data residing in these components are inspected to certify the correctness. Conditions to check boundary logics and boundary values are experienced to guarantee that the component is working at its extreme values without any problem. Every independent path which represents the independent logics of the component should be tested to ensure the reliability and correctness of the component. During the testing of individual components, Syntactical Faults, Semantic Errors, Logical Errors, and Coding Mistakes are checked before any other test is initiated. In addition, Independent paths, In-Out Interfaces, Local data structures (within sub component and outside the subcomponents), Boundary conditions, and Error handling paths are tested during components testing in component-based software, shown in Fig 2. All these testings must be backed up by the test cases which keep the track record of these different categories of testings.

5.2 Test specification

Every test has its attributes and characteristics. A Test Specification is a kind description that defines and specifies the attributes of a test. A test specification consists of - complete description and layout of each individual test, the cardinality and the cardinality ratio of test items belonging to each domain, the kind of test elements and their formats, the process of testing for each independent test item, method of recording these items in test the form of test cases, the semantics of test cases that is the interpretation of test outcomes, the starting and ending dates of testing, the minimum and the maximum period of time to conduct these tests. In general, A test specification provides the complete structure of the testing elements and its corresponding attributes.

5.3 Test plan

The software under consideration has some requirements and testers try to verify the software in accordance with those requirements. A test plan is like a manuscript and directive which addresses the verification of particular software’s requirement and the design specifications. An input for test plan comes from the testers, developers and sometimes from the end users. A test plan is a document of planning and consists of-

- What will be the process of testing?
- What will be the basic inputs and what will be the final outcome?
- What elements of the software will be tested?
- What quality levels are required?
- Who will involve in the testing process, i.e., tester, developer and end user.

5.4 Test cases

In addition to other elements, component-based test case is a complete document that contains three basic constructs-

- Actual inputs to the system,
- Actual outputs achieved from the system,
• Testing logic through which input is transformed into output.

A test case is an execution of test plans and recording the outcomes to verify the actual outcomes in comparison with expected ones. Test cases are the documents that are used at various levels of testing.

5.5 Test documentation

During Component level testing our major focus is on input, output and internal as well as related and supporting logics of the component as shown in Fig. 2. Test documentation is used as testing repository not only for the individual component but it is used to document the complete testing data of component-based software, during integration and system level testings.

**Syntactical Faults:** In component-based software, components interact through parameters. These parameters are passed through the component’s language. Then the communicating syntax used between different components must be unambiguous and clearly defined. In this communication, interfaces are used to make compatibility between different components, having diverse syntactical problems.

**Semantic errors:** A semantic error basically means the ambiguity or the unclear specification interpreted by the system or the user, which may cause undetermined behaviour, incorrect output or no output.

**Logical errors:** In component-based software, a logical error denotes the faulty implementation of algorithms, process logics and similar mistakes that causes it to operate incorrectly. Logical errors normally do not reflect in the form of errors, but they produce wrong solutions.

**Coding mistakes:** It may be a syntax error, semantic error, logical error or may be a combination of these.

**Identifying and testing independent paths:** An independent path is a path of component’s execution from source to termination. It includes the inclusion of any new statement at component level or the inclusion of a new component at the system level. Generally the execution of a component or the component-based software follows a predefined path. Each independent path designed and defined in the component structure must be tested to identify and test the execution path.

**In-out interfaces:** In component-based software, interfaces are used by components, and these interfaces provide data and information to each component. So in this testing input and output data related to each component is verified.

---

**Fig. 2. Component Level Testing**
Local data structures (within sub component and outside the sub components): In In-Out interface testing we test inter-component data to verify the behavior of components whereas in ‘local data structure’ testing our focus is on testing intra-component data.

**Boundary conditions:** It is experienced and expected that the logical conditions and the components normally fail at their boundaries. In this testing testers try to check these boundary conditions.

**Error handling paths:** Components execute according the pre specified path. These paths must be checked not only for normal execution paths but for the abnormal execution paths.

6. **CONCLUSION**

Component level testing concentrates on the testing and assessing the smallest piece of the underlying software, that is, the software component. Using the specifications of the architectural design and the test documentation as a directive, important internal logics and in-out data and control paths are tested to unhide faults and mistakes in the component. The component level testing uses equivalent considerations of Black-box and White-box testing strategies during components testing.

The most common errors in testing of components are-

1. Misinterpretation or misunderstood or incorrect requirements,
2. Incorrect design or operations,
3. Inaccurate initialization of selected data,
4. Inaccuracy in specification at any level or in any phase,
5. Incorrect representation of an expression.

**REFERENCES**