CHAPTER 8

COMPONENT-BASED TEST PROCESSES

8.1 Introduction .................................................................131
8.2 Software Test Processes ..................................................131
8.3 Component-Based Software Testing Requirements ...............132
8.4 Modified Component-Based Test Process Documentation ........134
8.5 New Process to Construct Testable Component ...................138
8.6 Summary ......................................................................143
CHAPTER-8 Component-Based Test Processes

8.1 INTRODUCTION

The Component-Based Software Development (CBSD) focuses on building large software system by integrating previous existing reusable and testable component. This approach can potentially be used to reduce software development cost, testing efforts, assemble system rapidly, and reduce the maintenance burden. CBSD reduces testing efforts because software developer already uses testable components. Component-Based Software Testing (CBST) of component is maturing as a new sub-discipline of Component-Based Software Engineering (CBSE). CBST is a process used to identify the correctness, completeness, quality and documentation of developed component. So moving towards this direction this study proposes two component-based test processes. First is modified component-based test process documentation and second is a new process to construct testable component.

8.2 SOFTWARE TEST PROCESSES

Software test processes helps in increasing productivity and quality assurance in software development. The increased size and complexity of software systems has led to the current focus on developing distributed applications that are constructed primarily using components so the Component-Based System (CBS) require efficient and effective component-based test processes to test these systems and need to develop effective techniques for testing various aspects of the components, viz. reusability, security, dependability and safety. Study on the testing by several researchers indicates that more than fifty percent of the cost of software development is devoted to testing and it results into very high cost for testing complex software. This percentage is even higher for critical software such as that used for avionic systems (Mary, 2000). CBS, which is developed by using component-based approach, is composed primarily of testable components - modules that encapsulate data, functionality and are configurable through parameters at run time (Lewis, 1996). Our first main concern in this chapter is to improve component-based test processes documentation while considering two
factors viz. CBST requirement and test case process documentation; and second is to present a new and innovative process to construct testable component.

8.3 COMPONENT-BASED SOFTWARE TESTING REQUIREMENTS
Component-Based Software Testing (CBST) requirements help in enhancing testing of software component in CBSD if these requirements are specify at the right time. CBST requirement is required to know what are the properties of the component according to which the test plans and test case is designed to test the reusable and testable component. The main requirements of the reusable and testable component is availability source code with White-Box Testing (WBT) and Black-Box Testing (BBT), component functionality, component compatibility, middleware of software component, interaction amongst testable and reusable component, component specification.

8.3.1 Availability of Source code with WBT and BBT
White-box testing is a software testing approach that examines the program structure and derive test data from the program logic and black-box testing in which the internal working of the item being tested are not known by the tester (Hans, 2002). WBT and BBT are used to test the main component. The main component has to typically be tested first using test cases generated on the basis of the source code, and then using test cases generated on the basis of its specification (Gill, 2002). When source code is not available for some components then, BBT of components is usually sufficient, because at this level there is often no need for low-level testing techniques based on source code.

8.3.2 Testing of Component Functionality
Component functionality is of an existing component that cannot be reused without change and testing. Existing component and a component to be newly developed match according to their functions. The new component to be developed may require some changes to corresponding functions in the existing component, or may require additional functions (Sami and Volkar, 2004). Often the main component requires only a subset of the functionality a component
provides. Thus, testing techniques for CBS have only to test these required subsets. Experience shows that these subsets have to actually be tested (Gill, 2002), because components are often tested with respect to a special application domain, and thus subsequently fail in new environments.

8.3.3 Testing of Component Compatibility
In CBSD the most important factors for successful reusability of component is the compatibility between different versions of the components. A component can be replaced easily or added in new parts of a system if it is compatible with its previous version. The compatibility requirements are essential to test for running software system, for many years. Compatibility issues are relative simple when changes introduced in the software systems are of maintenance and improvement nature only. Using appropriate test plans, including regression tests, functional compatibility can be tested to a reasonable extent. More complicated problems occur when new changes introduced in a reusable component which eliminate the compatibility (Won, 2005). To solve such problem, additional software must be written and tested separately which can help in managing the compatibility of both versions.

8.3.4 Testing the Middleware of Software Component
Middleware is used to manage the connections between components at run time and the moving of data from one component to another. Test cases have to be provided for testing the middleware used and these test cases are once created and used again and again. It is obvious that failures of middleware are likely to influence the behavior of the entire system because ideal middleware would shield the complexities of handling communication protocols and different operating systems (Hans, 2002). Therefore, testing of CBS has also to aim at ensuring the absence of faults in the middleware layer.

8.3.5 Integration Testing for Interaction
Integration testing consists of testing the interaction of the main component with other components to make successful reusability. Integration testing focuses on
CHAPTER-8 Component-Based Test Processes

testing multiple modules working together by using top-down and bottom-up integration testing approaches. These approaches have great advantages that the basis interaction of the component can be seen and tested early. If all components have faults and bugs, a system consisting of them may fail due to faults and bugs.

8.3.6. Testing of Component Specification

Testing of software component specification requirement is most important because when either the specification or the code is modified, this information makes it easy to trace and complete associated modifications, thereby helping to guarantee that the specification remains up-to-date. The component specification and part of the test suit designed to validate a particular functionality described in a given part of specification. These test suites should include both inputs and outputs and again reused for component specification testing of another simple and reusable component (Crnkovic et al., 2002) and this will facilitate the regression testing of change to a component.

8.4 MODIFIED COMPONENT-BASED TEST PROCESS DOCUMENTATION

The test process documentation shall define the testing activities to be performed with the inputs and outputs of each activity. This study presents the different phases as shown in Figure 8.1 according to modified component test process documentation in sequences viz. component test strategy, component test planning, component test specification, component test execution, component test recording, component test completion, component test results. Component testing with component test case strategy shall specify with the emerging popularity of the CBSD, to ensure the reliability and reusability of the software systems and component test results compare the actual value with the expected value by which can easily develop strong and efficient tools and techniques of CBST. Starting from component test strategy and planning to component test completion and results are carried out for the whole component. Component test specification, component test execution, and component test recording may however, be carried out for a subset of the test cases associated with a component. Component-based
documentation process helps in making a reusable and testable component so researchers and practitioners can use these components in future due to the good documentation. Reusable and testable component are tested by this modified component-based test processes documentation which are helpful in making better testing tools, techniques and methodologies.

8.4.1 Component Test Strategy

Component test strategy is a statement of the overall approach to testing, identifying what level of testing methods, techniques and tools to be used and what is the expected value of component after going through the whole phases for testing of modified component test process documentation. The component test strategy shall specify the techniques to be employed in the design of test cases (Sami and Volkar, 2004). The component test strategy shall specify criteria for
CHAPTER-8 Component-Based Test Processes

test completion and component test results. These test completion criteria should be test coverage levels whose measurement shall be achieved by using the test measurement techniques. The component test strategy shall document the degree of independence required of personnel designing test cases from the design process, such as, the test cases are designed by the person, persons from a different section, and persons from a different organization (Weyuker, 1998). The component test strategy shall document whether the component testing is carried out using isolation, bottom-up or top-down approaches, or some mixture of these. The component test strategy shall document the environment in which component tests will be executed. The component test strategy shall document the test process that shall be used for component testing.

8.4.2 Component Test Planning

The component test planning shall specify how the components test strategy to apply component under test. This shall include specific identification of all exceptions to the component test strategy (Sami and Volkar, 2004) and all software with which the component under test will interact during test execution, such as drivers and stubs. The component test plan shall list the test case design techniques to be used when executing tests and specific test completion criteria.

8.4.3 Component Test Specification

Component test specification shall be reviewed for completeness. Test cases shall be designed using the test case design techniques selected in the test planning activity. Each test case shall be specified by defining its objective, the initial state of the component, its input, and the expected outcome. The objective shall be stated in terms of the test case design technique being used, such as the partition boundaries exercised.

8.4.4 Component Test Execution

Before the component test execution, the test specification shall be reviewed for completeness. Component test shall be prepared by coding the drivers and stubs specified in test plans, compiling and linking with the component under test. Each
test case shall be executed. Test execution should not be halted for minor errors. Test driver shall include code to control tests and create a log file, and this log file is examined by tester to compare the actual test outcomes with the expected test outcomes.

8.4.5 Component Test Recording
A component test record shall be produced each time a test run, containing the component specification and component under test, number of test discrepancies, coverage measurement and fault reports (Weyuker, 1998). The test records for each test case shall unambiguously record the identities and versions of the component under test and the test specification. The actual outcome shall be compared against the expected outcome. Any discrepancy found shall be logged and analyzed in order to establish where the error lies and the earliest test activity that should be repeated in order to remove the discrepancy in the test specification or verify the removal of the fault in the component.

8.4.6 Component Test Completion
The test records shall be checked against the previously specified test completion criteria. If these criteria are not met, the earliest test activity that must be repeated in order to meet the criteria shall be identified and the test process shall be restarted from that point. It may be necessary to repeat the test specification activity to design further test cases to meet a test coverage target.

8.4.7 Component Test Results
Component test results are used to check the final report of the component after comparing the actual test outcomes with the expected test outcomes and component test results fulfil the test strategy specified during the start of component test documentation process viz. techniques to be employed in the design of test cases (Hans, 2002), criteria for test completion, personnel designing test cases from the design process, such as, the test cases are designed by the person, persons from a different section, and persons from a different organization the expected value of the components, check the results and test the component is
pass or fail according to the environment in which component tests will be executed.

8.5 NEW PROCESS TO CONSTRUCT TESTABLE COMPONENT
Component-Based Software Engineering (CBSE) techniques are gaining substantial interest because of their potential to improve productivity and lower development costs of new software applications, yet satisfying high reliability requirements. High reliability requirements are helpful in developing and reusing reliable components in software development. To merit the attribute "reliable", a component should be extensively validated. Testing is the technique which is most commonly used for validation. Testing technique should be applied efficiently on time to make a testable component. A testable component is not only deployable and executable, but also testable. It should be constructed in a way to facilitate component testing and automation to reduce the validation cost of component testers and users. Unlike normal components, testable components must constructed using a well-defined process and consistent test interface and traceable component must be traceable. As defined in (Jerry and Youjin, 1999), traceable components are ones constructed with a built-in tracking mechanism for monitoring various component behaviors in a systematic manner. Moving this direction this study proposes a new and innovative process to construct testable component with following steps.

8.5.1 Testable Component and Need of Testable Component
This study already discusses and define testable components in previous chapter. Now this study presents the need of testable component to make high quality CBS with desired to find an approach to build software components with sufficient testability so that they are easily executed, traced, observed, controlled, and tested. Testable components enhance component testability, achieve component test automation and reduce cost and time of development.

8.5.2 Create Component according to Requirements
As shown in the Figure 8.2, start creating a component from the scratch according to requirements. A component is built to be reused and reusability implies
CHAPTER-8 Component-Based Test Processes

generality and flexibility. The development process of building testable components can follow an arbitrary development process model. The generality requirements imply often more functionality and require more design and development efforts with more qualified developers. So the component development will require more efforts in testing and specification of the components. So to make testable components, component should be tested in isolation, but also in different configurations. Finally the documentation and the delivery will require more efforts since the extended documentation is very important for increasing understanding of the testable component (Gill and Tomar, 2008).

8.5.3 Component Test Plan

The component test plan helps in specifying how the components test strategy applies for a component which is under test after completion of previous step of creating a component from the scratch according to requirements as shown in Figure 8.2. This shall include specific identification of all exceptions to the component test strategy (Sami and Volkar, 2004) and all software with which the component under test will interact during test execution, such as drivers and stubs (BCI, 2004). The component test plan shall list the test case design techniques to be used when executing test and test completion criteria for developing testable component.

8.5.4 Define the External Interface of Component

This step is used to define the components internal interface and describe most of the test framework can wrap components with the external interface automatically. Testable components must include consistent and standard interfaces that regulate the testable component's interaction with testing tools to facilitate component testing. Each testable component has to implementing the internal interface that directly interacts with the original component interface, and supporting the external interface that interacts with external test tools. Component has two operational modes - (Yi Tien Lin, 2004) the test and normal modes. In the test mode, the component's functions are executed based on its test interfaces. In
the normal mode, the component is executed as its normal functions through its interface. It helps in setting-up a test case including its test classes, test functions and test data. After execution, function checks the actual results against the expected results for the test case. Testing and validation results are - pass, fail, exception, and unknown. The result of each test case is saved as the result test data format in the test case and generate test reports with testing information from test case.

Figure 8.2 Step wise Process to Construct Testable Component

8.5.5 Testing of External Interface of Component
This step is used to test the component external interface and describe how test used to test. Each component has a single test driver in the reusable test
framework. In addition, each test driver has a component identify number. Testing tools is used to prepare test cases and run them through the external interface. Actually, the procedure is done by calling test deriver. Four type of test driver instantiates the external interface that support component testing.

- **Functional Test Derivers**
  These are created for accessing component functions through component interface. In general, each component function needs one functional test driver.

- **Condition-Setup Test Derivers**
  These are created as control programs that set-up a pre-defined binary search tree before running the tests according to condition.

- **Case-Oriented Test Derivers**
  These are created for setting up test cases and test data with expected outputs.

- **Genetic Test Deriver**
  These are set up to control and execute test suites.

### 8.5.6 Define the Internal Component Interface

This step is used to define the component internal interface by specifying which component classes will be using for testing. This operation instantiates every constructor and function of the selected component classes and maintains links or pointers with them. The component will then be in the test mode and external interface calls to cancel the test mode. Pool would return the specified component function by supplying component interface signature. Component function in the pool returns the test function as an object. The component function should have exactly the same component interface signature with that of the test function.

### 8.5.7 Testing of Internal Interface of Component

This step is used to test the component internal interface after defining the component internal interface. The instantiation of the component internal interface is known as a component test wrapper. The actual implementation is a
test adapter. The test adapter usually is dependent on the used technology of the component. However, it is totally possible to reuse a test adapter for the same programming language (Yi Tien Lin. 2004). Test Functions can be set up dynamically; therefore, the impact of changing code inside the component interface is very small to the test framework.

8.5.8 Design a Testable Test Bed to make Testable Component
This step is used to achieve the objective of Plug-in and Test; describe here three different stages for development and packaging of testable components;

8.5.8.1 Development Stage
The test definition library including the component profile and interface during development stage. Test definition library classes are regulated and developed by a graphic user interface helper program.

8.5.8.2 Testable Stage
A testable component has two additional classes for the internal interface - the component test wrapper and test adapter classes. In addition, it writes a property file to indicate the files names and locations where the component profile and interface have been saved.

8.5.8.3 Executable Stage
Once the testable component has been plugged-in, the test framework first reads the property file from it and restores the component profile and interface to their corresponding class type.

8.5.9 Execution of Testable Component
Before the component test execution, component test specification helps in reviewing for completeness of software. Compiling, linking and creating testable component under test are helpful by coding the drivers and stubs specified in test plans and also help in preparing component test. Each test case shall be executed and for minor error test execution should not be halted. Test driver shall include code to control tests and create a log file and this log file is examined by tester to
compare the actual test outcomes with the expected test outcomes *(BCI, 2004),
(Gill and Tomar, 2007).*

8.5.10 Create Recording and Repository of Testable Component

This is the last step of the process as shown in Figure 8.2. A component test record shall be produced each time a test run. It includes component specification, component under test, and number of test discrepancies, coverage measurement and fault reports *(Gill and Tomar, 2007).* The test records for each test case shall unambiguously record the identities and versions of the component under test and the test specification. The actual outcome shall be compared against the expected outcome which helps in determining discrepancy. If any discrepancy found, it shall be logged and analyzed in order to establish where the error lies *(BCI, 2004)* and finally create the repository of testable component for future use.

8.6 SUMMARY

This chapter is divided into two main sections. First section introduces the process of improving CBST while considering two factors viz. CBST requirement and test case process documentation with modified component-based test process documentation with the following phases - component test strategy, component test planning, component test specification, component test execution, component test recording, component test completion, component test results. Second section introduces a new innovative process of building testable component with better interface through the following phases - testable component and need of testable component, create component according to requirement, component test plan, define the external interface of component, testing of external interface of component, define the internal component interface, testing of internal interface of component, design a testable test bed to make testable component, create recording and repository of testable component.