A Roadmap towards Analysis of Component-Based Software Testing

Parul Madan¹, Lata Nautiyal²

Department of Computer Science and Engineering
Graphic Era University, Dehradun - India
parulmadaan1@gmail.com, lata.nautiyal1903@gmail.com

Abstract
Software Development is a practice to meet the needs of customer in the form of software. Since time bound effort, huge amount of cost and lot of risk are key ingredients of its development process, testing is responsible to check the correct working of the functionalities thus makes it an important phase in software development. Although being an expensive task, it is important because it marks the success of the software being developed. In this paper we are analyzing and evaluating Component-based Software Testing techniques.

Keywords — Black box testing, Component, Component-Based Software, Functional Testing, Equivalence Class Partitioning, EEOCP,

1. Introduction
Component Based Software Testing application is used to reduce the complexity in software implementation. It reduces the delivery time of the product and increases the software productivity. From the last some decades, demand for high quality software has increased drastically by the customers and end users, and they want their software product to be developed and deployed within time. Also each and every organization wants to have it’s our software product within the specified budget. These are some of the reasons that make software developers to think about the Component - based software development to save time and cost by using already build components.

SOFTWARE TESTING
Software testing is a way of evaluation software program to find difference between given input data and expected output. Software testing assesses the quality of the product. In other way software testing is a process of verification and validation process.

Basically software testing is divided into three different types given blow.

1. Black box testing
2. Gray box testing
3. White box testing

♦ Black box testing:
To perform the black box testing we do not need any prior knowledge of codes or program designs, it is based on the functionalities and requirements of the program.

♦ White box testing:
To perform the white box testing, we also need prior knowledge of code or program required. Code statements, branches, path, and condition decide the coverage of test.
Gray box testing:
Combination of black box and white box testing combines to form gray box testing.

1.2 NECESSITY OF SOFTWARE TESTING
Human being makes error and error is also expected in software that is why software testing is required. Errors can be either important and expensive, or unimportant. Sometimes, there is a possibility that things can go wrong, that is why everything is needs to be checked.

Various goals and reasons are there to tell us them importance of the software testing that should be consider while any kind of product or application testing. Reasons which tell us the importance of testing are:

♦ Errors and defects which were made during the implementation can only be detected by software testing.

♦ Reliability of customers and their satisfaction can all be made sure only by software testing.

♦ Quality of the product can only be made sure by software testing. To gain the customer’s lack of confidence, quality product plays an important role.

♦ Software product and application’s performance is provided by software testing.

♦ Software testing is required to stay in the business.

1.3 SOFTWARE TESTING OBJECTIVE
Software testing has different objective and goals. The major objective of software testing is mentioned blow:

♦ Finding defects which may arise by the programmer fault while developing the software.

♦ To gain the knowledge and information in maintaining the level of software quality.

♦ To prevent errors.

♦ To make sure that the final result meets the business and user needs.

♦ We ensure that it satisfies the BSR (Business Requirement Specification) and SRS (System Requirement Specifications).

♦ By providing good quality product we can gain the confidence of the customer.

1.4 COMPONENT BASED SOFTWARE ENGINEERING (CBSE)
By reusing already build components, new application software can be developed by component based software engineering using third party components and in house built components. The reusable components are integrated to implement a new application.

![Figure 1.1 Framework of CBSE](image-url)
So the integration process of the components is very important in component based development and the quality of new developed application depends upon the quality of the components that are used in it.

Component Based Software Testing application is used to reduce the complexity in software implementation. It reduces the delivery time of the product and increases the software productivity.

From the last some decades, demand for high quality software has increased drastically by the customers and end users, and they want their software product to be developed and deployed within time. Also each and every organization wants to have it’s our software product within the specified budget. These are some of the reasons that make software developers to think about the component based software development to save time and cost by using already build components.

But problem may arise during the testing of such component based product because this type of new product is developed by integrating existing and also newly developed components that were made or used in any other product. These components may not work for certain requirements in the future means these were customized for other application and now these are deployed in another context and deployed in another context in new software product [2].Some software development approach have therefore emerged to analyze the reliability of component based applications. These techniques mainly fall into two categories [3, 4, 5].

- **System Level:**
  System level is very reliable, estimated for the software application as complete.

- **Component Level:**
  Reliability of the individual component and their interconnection mechanism can be estimated by reliability of individual program.

### 1.4.1 Strategies for Black-Box Testing

When the testing is performed, we tend to test our software for every possible task that it can perform which can make the testing phase much expensive. We would like to plan the test cases so that it covers every task that the customer has asked for and tests the software fairly that fulfills its objective to find more and more defects that may possibly arise in a fewer possible test cases [3]. To achieve this, the following strategies of black box testing have been proposed and are in wide use. It also includes the awareness that no test case is redundant that any previously defined test case has already revealed. Every test plan should be totally different in the conditions imposed for checking and should investigate entirely different mode of error or failure [4]. The test plans should be much simpler so that they may easily reveal the failure point.

The fundamental concept behind the functional testing is to assess all the available functions present in a program. This testing only takes the external view of the software. This testing is categorized into 3 different types.

1. **Boundary Value Analysis:**
2. **Equivalence Class Partitioning:**
3. **Decision Table-Based Testing:**

- **Boundary Value Analysis:**
  Boundary value analysis is also a black box testing technique to ensure that the software works correctly at the boundary values as input and output as the software tends to behave more abruptly at the boundary values when put to test
as compared to the values that are in the center of the limits of the conditions [10], [5]. One of the other reasons of the boundary value analysis is that defects and errors are easier to be depicted and caught at the boundary lines than in the middle of the range of the test cases. It focuses on data at the “edges” of an equivalence class [5]. Boundary value analysis techniques is a test functional testing technique where the boundary value are chosen.

° **Equivalence Class Partitioning:**
In the ECP input data of a software unit is divided into number of partitions of equivalent data from which test cases can be derived. Equivalence class partitioning (ECP) is always implemented to the inputs of an already tested software component, but sometimes it be implemented to the outputs in worst cases [5].

° **Decision Table-Based Testing:**
Decision table-based testing is used to evaluate and analyze the complex rules and regulations which must be applied in the program. Decision table-based testing works like if-then-else conditions and switch-case statements, which uses associate conditions with actions to be taken on these available conditions. Decision table-based testing is basically converted into four quadrants that are shown in below.

<table>
<thead>
<tr>
<th>The four quadrants</th>
</tr>
</thead>
<tbody>
<tr>
<td>All the Conditions</td>
</tr>
<tr>
<td>Associated Actions</td>
</tr>
</tbody>
</table>

![Figure 1.2 Equivalence Class Partitioning](image)

**Figure 1.2 Equivalence Class Partitioning**

**1.4.2 Strategies for White-Box Testing:**
White box testing is totally based on programming code because without knowledge of code, testing can never be complete, exhaustive efficient and effective. When the testing is performed, we tend to test our software for every possible task that it can perform which can make the testing phase much expensive. We would like to plan the test cases so that it covers every task that the customer has asked for and tests the software fairly that fulfills its objective to find more and more defects that may possibly arise in a fewer possible test cases. The test plans should be much simpler so that they may easily reveal the failure point. Execute every statement of a program in order to identify the error in each and every statement.
2. State of Art

This section spot light on the previous research done in the field of component based software testing. Vinita Arnica [5] analyzed the theoretical bounds of the size of test suites or the complexity of domain testing methods. Kirti Tyagi [6], uses Neuro-fuzzy model for estimating the reliability of component based software systems. The proposal is a combination of neural network and fuzzy logic. Leonardo Mariani [7], author addresses the problem of testing evolving software system; i.e. system obtained by modifying or substituting some of their components. Antonio Bucchiarone [8], the author presented how testing and fault tolerance can be jointly used in order to improve and validate component based software. Fault tolerance guide the construction of a fault tolerant architecture, which is successively validated with respected to requirements and submitted to testing.

A number of soft computing approaches for estimating components based software systems reliability have been proposed.[9] propose an approach to reliability analysis called scenario based reliability analysis. This approach introduces component dependency graphs. Using the same algorithm, sensitivity can be analyzed as a function of component reliabilities and link reliabilities. This approach is based on situation which can be captured with sequence diagrams and can be automated. A limitation of this approach is that it does not consider failure dependencies among the components. Dong et al. [10] proposed a new model for estimating component based software system reliability in which various complex component associations are analyzed. The Markov model is used to solve these complicated relationships, which have a large impact on a system’s reliability. Fiondella et al. [11] proposed an approach based on correlated component failures (COCOF). In this paper an approach to access the reliability of a software application, considering the component reliabilities, correlation and application architecture is proposed. This proposed approach is based on an algorithm that transforms a Multivariate Burnoulli distribution (MVB) into a joint distribution of the component outcomes.

N.C.W.M. Braspenninig [12], author defines how model of the components, developed for model-based integration, can be used for automated model-based testing. A Bertolino [13] this paper is based on the integrated use of Sequence and State Diagrams for deriving a “reasonably” complete reference model, which will then be used for automatically deriving the test cases. Bin Leia, in [14], author proposed a state machine based approach to detect robustness problem. Invalid input and inopportune method calls are fed to the component in the different states to test the robustness. When unexpected exceptions arise in the test runs, robustness failures are adopted.
Mourad S. Semary [15], proposed an approach for nonlinear boundary value problems with multiple solutions. This approach is based on the variation iteration method (VIM) and proposed with an auxiliary parameter to predict the multiplicity of the solutions of homogeneous nonlinear ordinary differential equations with boundary conditions. The proposed approach is capable to predict and calculate all branches of the solutions simultaneously. Anupriya Jain [16] proposed a boundary value analysis approach for non numerical variables such as strings. Numeral values, a set of related values, or any condition that results in true or false. This testing technique lowers down the cost of testing as it manages the test plans that check for the maximum output for the minimum input values. Once we have identified all the partitions, we select test cases from each partition.

CONCLUSION

During the entire software development life cycle, testing is an essential part. Equivalence class partitioning testing along with decision-table based testing and boundary-value testing make up the foundation of functional testing.

The future work is to propose an innovative approach in terms of test case generation procedure. And studying the complexity of test cases generation process can also be included in future scope of the paper.

10. References
