A Review on Component-Based Software Engineering and Testing

1Bhupender Yadav, 2Anshu yadav
1,2Assistant Professor, GITM, Gurgaon, Haryana
Email: 1byadav81@gmail.com, 2venaro1@gmail.com

Abstract: In the early 90s by the introduction of Microsoft’s Component Object Model (COM), Component Based Software Engineering (CBSE) has evolved as a popular software development methodology. Today Component Based Software Engineering (CBSE) is more generalized approach for software development. To ensure the quality of component based system testing is necessary. The objective of component-based software engineering is to efficiently and effectively develop high-quality software products. Under ideal situations, with minimal testing efforts, integration of reliable software components should produce high-quality software. In this paper some testing techniques are discussed that software engineers use in CBSE Models.

General Terms: Component-Based Software Engineering (CBSE), Commercial of the Shelf components (COTS), Component-based Software (CBS), Component-Based Development (CBD).

Keywords: CBSE, COTS, CBS and CBD.

I. INTRODUCTION

The building blocks for any software applications are software components is a software building block. A component-based system is an assembly of software components, which follows an engineering process that conforms to component-based software engineering principles. Different programming languages can be used to write these components, those can be executed in various operational platforms, and when on the Web, they can be distributed across vast geographic distances. Some of the components are developed by any third party, whose source code is usually not available to developers and some in-house. By allowing the integration of reusable components from different vendors the CBS characteristics facilitate development of quite complex systems in a way that it doesn’t affect the desired resulting system and by providing flexibility for dynamic software evolution. At the same time, they introduce new problems in testing and maintaining component-based software systems as well. [1-4]

II. PROPERTIES OF SOFTWARE COMPONENTS IN CBSE

A software component in CBSE must have the following basic properties:

• Identity:

The identity of each individual software components whose quality determines component is a key feature in its effectiveness of the applications. In Component-Based development environment and targeted Software Engineering (CBSE), reusable components deployment environment and it must be are mainly used to construct software systems. A uniquely identifiable. A large scale of component reuse is impossible without this feature. Current component technology ensures this by using a well-defined (or standard) naming scheme. CORBA, EJB, and Microsoft DCOM are typical examples.

• Modularity and encapsulation: Software components result from partitioning of a software system by focusing on system modularity. A set of closely related data elements is encapsulated by each of the component and then to perform a specific task it implements coherent functional logic.

2.1 SOFTWARE DEVELOPMENT CHALLENGES

Nowadays, there is an enormous expansion in the use of software in business, industry, administration and research. Software is no longer marginal in technical systems but has now become a central factor in many fields. System features based on software are becoming the most important factor in competing on the market, for example in car industry, the service sector etc. Increasing numbers of software users are non-experts. These trends place new demands on software. The most important features of software are usability, robustness, simple installation and integration. The demand for the integration of different areas has increased as a consequence of the wider area of software utilization.

2.1.1 Long-term management of component-based systems

The problem of system evolution becomes significantly more complex as component-based systems include sub-systems and components with independent lifecycles. There are many questions of different types: technical issues can a system be updated technically by replacing the components? Administrative and organizational issues which components can be updated, which components should be or must be updated? Legal issues who is responsible for a system failure, the producer of the system or the producer of the component? etc. CBSE is a new approach and there is little experience as yet of the maintainability of such systems. There is a risk that many such systems will be troublesome to maintain.
2.1.2 Development models

Although existing development models demonstrate powerful technologies, they have many ambiguous characteristics, they are incomplete and difficult to use.

2.1.3 Component configurations

Complex systems are made as many components which, in turn, include other components. Compositions of components will be treated as components in many cases. As soon as to begin to work with complex structures, the problems involved with structure configuration emerge.

2.1.4 Composition predictability

Even if it is assumed that all the relevant attributes of components can be specified, it is not known how these attributes determine the corresponding attributes of the systems of which they are composed. The ideal approach, to derive system attributes from component attributes is still a subject of research. A question remains — “Is such derivation at all possible? Or shouldn’t we concentrate on the measurement of the attributes of component composites?” [5].

2.1.5 Requirements management and component selection

Requirements management is a complex process. A problem with requirements management is that requirements in general are contradictory, incomplete and also not precise. The main objective in in-house development is to implement a system which will satisfy the requirements as far as possible within a specified framework of different constraints. In component-based development, the fundamental approach is reuse of existing components.

III. COMPONENT BASED SOFTWARE TESTING

White-box testing. Black-box testing, which is also known as functional testing, is the testing that ignores the internal mechanism of a system or component and focuses solely on the outputs generated in response to selected inputs and execution conditions. White-box testing, also known -as structural testing or glass-box testing, is “testing that takes into account the internal mechanism of a system or component.

IV. DIFFICULTIES IN ADEQUATE TESTING AND MAINTENANCE FOR COMPONENT-BASED SOFTWARE

Some major factors that affect our testing and maintenance activities are explained as:

4.1 Code availability

For COTS components source code is very often unavailable and interaction among the components is not that simple and fast. Lack of source code causes many problems. First of all, all White-box based techniques are not available and only Black-box techniques can be used. But sometime even black-box based testing techniques are difficult to adopt due to lack of knowledge that must be known for the software system.

4.2 Performance and reliability analysis

When testing component-based software, correctness is among the issues that often draws attention. Component-based software features heavily affects the performance. The other features such as reliability and maintenance are also reexamined beside the performance feature. Generally, how quality analysis results are passed from developer to user is key issue in testing and maintaining component-based software.

4.3 Adequacy

Adequacy is included in the list of toughest challenges in testing component-based software. If we are not provided with source code then to identify unified adequacy criteria is difficult.

4.4. Maintenance

The reusability of components as building blocks for software development helps in improving software quality and productivity. When a component in component-based software is modified or upgraded, a maintenance activity occurs [6]. Due to many of the characteristics of component-based software, difficulties can be encountered when traditional maintenance approaches are applied. The cost of maintenance phase for conventional software as two-thirds of the total cost and it can be still more for maintaining CBS.

V. REGRESSION TESTING FOR COMPONENT-BASED SOFTWARE

Regression testing is the process of testing changes to computer programs to make sure that the older programming still works with the new changes. Regression testing is a normal part of the program development process. Regression testing plays an integral role in software maintenance; without proper regression testing we are reluctant to release modified software. Regression testing attempts to revalidate the old functionality inherited from the old version and users have to maintain their systems according to the changes that the component providers have made. Component providers have full control of their components, and can therefore use traditional approaches to maintain their components [7,8].

5.1 Need of Component Based Regression Testing

The purpose of regression testing is to selectively retest the software after certain modifications to ensure that they have not caused unintended effects on unchanged parts and changed parts of the software behave as intended [9]. Therefore, regression testing focuses on identification of changes so that those unchanged parts
are already tested should not be tested again to reduce cost, and only changed parts corresponding to those changes should be tested. The objective of regression testing include not only involves selective retesting, but also enhancing the confidence of the clients that the software product can be changed according to their requirements [10].

VI. CONCLUSION

Testing is one of the important phases of software development. Component-based software technology is a new trend of constructing software system. Due to many excellent specialties of component and the immense benefits from using component to build software systems, component-based software has been widely used in various application domains and becomes a fairly popular software form in recent years In general there is a lack of tools, method and strategies that covers the integration testing problem as a whole: from defining the integration order to test case selection. It can be concluded to overcome the component-based software testing challenges use of proper methods, tools and strategies is key factor. And an approach to component and their interactions modeling using UML interaction diagram, component and their interface are specified by using UML diagram [11].

VII. FUTURE SCOPE

Future of Component-Based Software Engineering It is obvious that CBD and CBSE are in the very first phase of their lives. CBD is recognized as a new, powerful approach that will, if not revolutionized, at least significantly change the development of software and software use in general. The components and component-based services will be widely used by non-programmers for building their applications. Component assembly will develop tools for building such applications. Automatic component update over the Internet will be a standard means of application improvement already present in many applications.

REFERENCES


