Journal of Theoretical and Applied Information Technology
20th January 2015. Vol.71 No.2

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

A COMPARATIVE EVALUATION OF STATE-OF-THE-ART INTEGRATION TESTING TECHNIQUES OF COMPONENT-BASED SOFTWARE

[1]ABUBAKAR ELSAFI, [2]DAYANG N. A. JAWAWI, [3]ABDELZAHIR ABDELMABOUD, and [4]AWAD ALI

[1,2,3] Department of Software Engineering, Faculty of Computing, Universiti Teknologi Malaysia,

81310 UTM Skudai, Johor Bahru, Malaysia.

[4] Faculty of Computer Science and Information Technology, Kassala University, Sudan.

E-mail: [1]eaaabubakar2@live.utm.my, [2]dayang@utm.my, [3]areabdelzahir2@live.utm.my,

[4]awad.uofkassala@gmail.com

ABSTRACT

In the last few years, component-based software has gained widespread notice and acceptance as a method that facilitates the development of existing large, complex, and very critical systems by integrating prefabricated small pieces of software called components. Component integration becomes an essential stage in the component-based software development Lifecycle. Therefore, testing components after integration is an important activity. Due to the unavailability of source code of integrated components and due to the lack of component information or documentations, integration testing becomes more difficult and very complex task. In the literature, different techniques have been proposed with the purpose of facilitating the integration testing of component-based software. In this paper, we study, classify, and evaluate some of the existing integration testing techniques and make a comparison in order to help in develop new, better and more efficient and effective techniques for integration testing of component-based software systems.

Keywords: *Comparative Evaluation, Integration Testing, Component-Based Software, Component Testing, Critical Review*

1. INTRODUCTION

Nowadays, software applications have become increasingly larger, more complicated, distributed amongst network and very critical. On the other hand, time-to-market need to be decreased due to the competition amongst Software Company. Due to that, software engineers and developers are facing several challenges in developing software applications. To meet these challenges, software engineers and developers try to look for alternative approaches that facilitate the development of this kind of software applications. Consequently, they found that, today's large, complex and very critical software applications could be developed partially if not completely, by reuse of pre-built sub-systems that their operations have been previously tested as a part of successful applications. This approach is called Component-Based Software Engineering (CBSE) [1].

CBSE is a way used in developing software systems by integrating of pre-built software parts or components. Whereas old approaches that fail to develop today's complex software applications, attempt to develop software from scratch, one at a time [2]. Given this fact, the overall software management has becomes more complex and the results of that, the productivity has becomes low and the cost of development is becoming higher. In the literature, there are several definitions for software component has provided by many the researchers. However, the most widely accepted definition for software component is presented by Clemens Szyperski [3]. According to [3], "a software component is a unit of composition with contractually specified interfaces and explicit context dependencies only. A software component can be deployed independently and is subject to composition by third parties".

Many advantages have been obtaining from developing software applications using previously developed components. The most important advantages are; the efficiency of development increased, the product becomes more reliable, need

Journal of Theoretical and Applied Information Technology
20th January 2015. Vol.71 No.2

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

for maintenance is radically reduced, assemble systems rapidly with viewer resource, and hence the development costs will be reduced [4, 5]. In current software development efforts, more than 70% of all new complex software applications will closely depend on Component-Based Software (CBS) [6].

Despite of this several advantages and propensity of using software components in building large and complex software systems, CBS has introduced new problems in the field of software engineering and software testing as mentioned by many researchers [4, 7, 8]. Ye et al. [7] mentioned that, depending on software components for building software applications introduces new problems of testing and maintaining software systems. The evolvement of CBS, which produces systems by integrating prefabricated software, and the increased emphasis on software quality, highlight the need for an improved testing methodology. Also, the heterogeneous nature of components and deployment architectures introduce complexities in the integration process that must be analyzed and validated during a testing process [8]. Furthermore, the existing body of knowledge in this field tells us that there are problems in integrating the component [4]. This in turn will affect the quality and reliability of the software constructed by integrating components. Therefore, failure of testing CBS means a financial loss, increased expenses of software and hardware development, and worse than that, the loss of relationships with consumers, i.e. Ariane 5.

In the literature, there are several techniques for integration testing of CBS has been proposed by the researchers. This paper aims to study, classify, and analyze the various existing integration testing techniques of CBS, which can help researchers and practitioners to develop and build new, better and more efficient techniques.

The remainder of this paper is organized as follows. Section 2 presents the procedures for obtaining relevant study. Section 3 introduces the related works and background. Section 4 describes the existing CBS integration approaches and techniques. Section 5 provides the comparative evaluation. Section 5 discusses the results. Section 6 summarizes the current issues and future directions and finally, section 7 concludes the paper along with future work.

2. METHODOLOGY

In order to analyze various existing integration testing techniques available for CBS systems, available literature was extensively studied. This was done by applying various search techniques to sources like digital libraries of Science Direct, IEEE Xplore, Springer Link, ACM Digital Library, and other online sources such as Google scholar, DBLP, Inspec, and open access journals. During the study, various journal, book chapters, technical reports, and conference papers were referred from these sources. However, the complete review process has been described in Figure 1. The keywords that were used are given in Table 1.

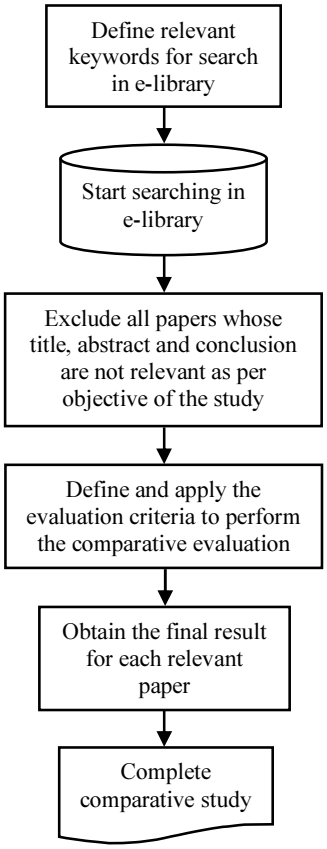

Figure 1: The Comparative Evaluation Procedure.

Table 1: Relevant Keywords

#	Keywords
1	integration testing
2	integration testing technique
3	component based software
4	component-based testing technique
5	component based software engineering

Journal of Theoretical and Applied Information Technology
20th January 2015. Vol.71 No.2

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

3. RELATED WORKS AND BACKGROUND

During the last decade, there have already been literature surveys and systematic review works done in this area. Therefore, several researchers have already reviewed the state-of-the-art with respect to testing CBS. Beydeda and Gruhn [9] have conducted a survey on the current approaches of component testing that explicitly tackle the problem of lack of information in development. Beydeda [10] also has provided an overview of the approaches to testing CBS. In his work, some of the drawbacks of these approaches were outlined. Rehman et al. [11] have provided a review on proposed techniques for component integration testing on the component users' side, and they highlighted some issues in software component testing. Shashank et al. [4] have presented a systematic literature review related to integration testing of CBSE. The aim of this systematic literature review is to investigate the state of the art in integration testing of CBS. The conducted literature review was based on 49 articles. The study covers articles that were published between 1995 and 2009.

However, all the above-mentioned studies aggregated the existing literature as normal literature survey without following any pre-defined criteria. To the best of the authors' knowledge, there is no any study in this area that considers the existing literature with respect to a set of pre-defined criteria.

3.1 CBS Testing Levels

Three basic kinds of testing are needed in Component-Based Software Development (CBSD) Lifecycle in order to detect and reveal errors [11]. These kinds are unit testing (component testing), integration testing (deployment testing), and system testing. However, Figure 2 and Table 2 illustrate and summarize these main levels of testing in CBS. Brief overview of these three main levels are given in this section.

The first type of testing in CBSD is unit testing or component testing. It is performed by component developer or the person who built a component. The main target of unit testing is to early detection of possible failures. Due to the availability of source code, the component developers can use white-box or black-box testing techniques to perform the testing process.

The second type of testing in CBSD is integration testing or deployment testing. It is mainly performed by the component user or by independent tester (third-party tester). Referring to IEEE, integration testing could be defined as "testing in which software components are combined and tested to evaluate the interaction between them" [11, 12]. Therefore, this kind of testing performed to evaluate the interaction between combined or multiple components. Due to the unavailability of source code of integrated components, a component user limited only to use black-box testing techniques to perform the testing process.

The last kind of testing in CBSD is system testing; which is also performed by the component user or independent tester (third-party tester) when all the various components are integrated and the entire system is ready to run. Since the source code of a components is also not available to the component user, the black-box testing techniques are the only choices for component users to perform system testing.

3.1.1 Motivation

The topic selected for this study is motivated by the following; integration of components is currently a major mode of software development [13]. Despite the fact that a component might go through a numerous of successful tests, unit testing still cannot guarantee the reliability and the behavior of components in a new environment after integration [12]. Thus, integration testing plays an important role and it becomes very essential step in the testing process in CBSD Lifecycle, which individual software components are assembled and verified as a group to attain a high level of quality and reliability. Approximately, in current software development efforts, 40% of software errors are discovered and revealed during integration testing [14]. Additionally, integration testing is more time consuming and very expensive part of testing process in CBS [15].

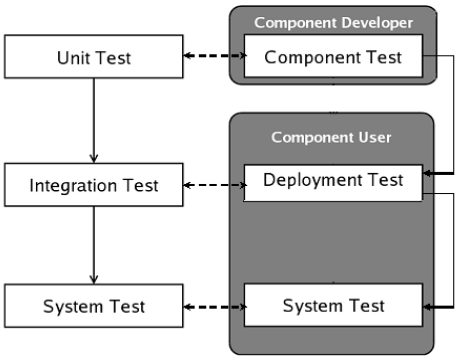

Figure 2: CBS Testing Level.

Journal of Theoretical and Applied Information Technology
20th January 2015. Vol.71 No.2

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

Table 2: Summary of Main Level of Testing in CBS.

Testing Level	Specification	Scope	Technique	Who does it?
Unit testing.	Low level design. Actual code structure.	Single component.	White-box or Black-box.	Component developer.
Integration testing.	Low level design. High level design.	Multiple components.	Black-box.	Component user or Independent tester.
System testing.	Requirement analysis.	Whole product in representative environment.	Black box.	Component user or Independent tester.

4. INTEGRATION TESTING OF CBS

When considering components integration testing, many issues and challenges could be identified [9, 16-19]. More precisely, the crucial challenges are missing of information (i.e., internal structure, specifications) of component and the unavailability of component source code. Due to that, the traditional techniques such as static analysis, program slicing, invariant detection, model extraction, validation and verification have become ineffective when one considers integration testing of CBS [20].

Consequently, several approaches has been proposed by researchers in recent years such as, Built-in testing approach, Metadata-based testing approach, Testable architecture approach, Self-testing approach and Certification strategy, aiming to facilitate integration testing of CBS. However, this section presents an overview of these approaches and the classification of the existing literature amongst these approaches.

4.1 Built-in Testing Approach

Built-in testing (BIT) is a way in which built-in tests are incorporated in the component's code with aims to facilitate component integration testing [9]. The idea of BIT previously was adopted in Object-Oriented programming [11]. Therefore, BIT is all strategies that add information in software component's implementation for facilitating integration testing or checking assertions at runtime to support self-testing.

Edwards [21] has proposed the use of wrappers to support the flow of information between component provider and component user. In this technique, the component provider attaches the information that can facilitate in CBS testing to the component and gives some wrappers that can interact with the component. So, wrappers can be used by component user to extract information from the component. Therefore, wrappers can be added or removed from software component by component user without the need to access the source code.

Wang et al. [22] provides a BIT technique for enhancing CBS maintainability. In this technique, built-in tests were added as extra member functions to the component's code. Therefore, the component user can make a decision whether to perform these tests or not. A software component can operate in two different modes in this technique. Precisely, "normal mode" or "test (maintenance) mode". In normal mode, tests methods are not executed, whereas in maintenance mode, the built-in tests methods are invoked through execution of the component. However, due to the added test methods, Wang's technique increases component size.

Atkinson et al. [23] has proposed the Component+ BIT (C+ BIT) technique in order to deal with Wang's problem. In this technique, test cases were separated from software component. The component developer constructs a BIT-component and a test-component. In a BIT-component, the component has built-in testing abilities. The test-component encloses test cases and interacts throughout its interfaces with the built-in testing capabilities of the BIT-component.

Momotko and Zalewska [24] have proposed a framework for testing the interaction of components at runtime. This framework is another example of C+ BIT technique. In this framework, two types of testing were proposed to test software component: contract testing and QoS testing.

Self-testable software component using Transaction Flow Model (TFM) proposed by Martins et al. [25] is another example to add extra information to software component intentionally to improve component testability by integrating testing resources into it. A tool called Concat was developed to support the proposed technique and to generate test cases.

Beydeda and Gruhn [26] have presented a Self-TEsting COTS Components (STECC) strategy. The idea of STECC is to supplements the test component with testing tools and analysis functionality. By doing this, either the information that the component user needs to generate test cases

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

can be generated on demand, or it can be augmented in the component.

Mahmood [8] has proposed a CBS integration testing framework to identify test criteria and to prioritize test cases based on complexity metrics.

4.2 Metadata-based Testing Approach

The idea of the Metadata-based testing approach is that a component provider attaches with the component some information (meta-methods, component metadata) that the component user can use for performing integration testing [27]. The meta-methods are the methods that used by component user to extract or compute information about the component while component metadata is information about the component itself.

Orso et al. [27] has proposed that all software engineering artifacts used in the development of software component is metadata and should be appended along with the component. Therefore, the component developer can present data-flow and control-flow graphs of the component that increase testability and understandability of the component. In addition, these provided graphs are useful in implementing coverage analysis during CBS integration testing.

Wu et al. [6] has suggested idea to attach a components' UML model as metadata. This UML models can be used to define context-dependent relationships between the components, which can be useful for CBS integration testing.

Belli and Budnik [28] have extended the work of [6] by augmenting a component with UML statecharts diagrams. The UML statecharts will be used to generate test cases with a help of model-based tools. Using these techniques, the component user can perform coverage-based execution of the model, to achieve greater reliability of the component. However, the component provider has to generate the model each time the component is modified.

Liu and Richardson [29] was introduced the concept of retro-components. A retro-component has a retrospector in it that maintains testing and dynamic execution history. It records the component developer tests and makes this testing information available to the component user. Retrospectors enhance the component such that the user can query the information provided and collect relevant information during their own testing activities.

Silva et al. [30] has presented approach covered by a CASE tool integrated in the development environment to support CBS integration testing aiming to reduce the lack of information between component developer and component user.

Naseer et al. [31] has presented an approach to use metadata technique for CBS black box testing and developed a tool which takes <.dll> component.

4.3 Testable Architecture Approach

In this approach, the component provider equips a specific testable architecture with software component that helps the component user to easily execute the test cases for integration testing.

Ye et al. [32] has proposed a test model for integration testing called Component Interaction graph (CIG). In this technique, test elements should be defined at first to construct a CIG. For each test element, test cases will be generated, and based on this test elements the test coverage criteria are defined.

Gao et al. [33] has introduced testable beans technique with aims to increase a component testability. Therefore, the component developer implements codes test cases and a specific interface for testing (test interface) in the term of clients.

Jabeen and Rehman [34] have provided a framework for testing object-oriented components. In this framework, the requirements of the component (test information) are inserted in what is called descriptors, and later the component provider, component user, and independent tester will use these descriptors to communicate test information between them. Therefore, the component developer should prepare and attach component descriptor into the component. The component's requirement should be specified by the component consumer in another descriptor called component requirement descriptor. At last, the test information will generated by independent tester using the information in the component descriptor and the component requirement descriptor.

Brohi and Jabeen [12] have extended the work of [34] by proposing a framework that enhances component testability to facilitate the integration testing process by defining a uniform information flow in the component Lifecycle.

4.4 Certification Strategy Approach

The philosophy behind this approach come from assumption a components can be certified before their reuse in CBS. This certification will increase the component user's trust [35]. However, to certify

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

a software component, three strategies have been introduced, third-party certification, developer certification, and user certification [11].

4.4.1 Third-party certification

Counsill [36] has suggested that a component must be certified by a third-party. In third-party certification, the quality of the component should be tested by independent third-party organization tests and provides the test results, along with the test environment, to the component consumer. Yu-Seung et al. [37] has proposed a framework for third-party certification which consists of three steps. The third-party provides guidelines to the component provider, then, the component provider generates a test package using these guidelines. Lastly, the third-party executes the test package and produces a test report. Some errors in a component were revealed during the evaluation of this framework, hence demonstrated its usefulness.

4.4.2 Developer certification

Morris et al. [38] has suggested that the component developers must perform component certification. The idea of this approach is that the component provider adds test cases along with their results (as a proof of their execution) to the component in order to avoid the cost problem related with third-party certification technique.

4.4.3 User certification

Voas [39] has suggested that black-box testing should be used by the component user to certify the component, in which test cases are generated from the interface specifications of the component and hence will help to address the above issue in component developer certification technique. Therefore, the component user may use fault-injection techniques in which faults are generated instead of testing the component with the correct inputs, to determine the reliability of the component.

To conclude, from Table 3 we can summarize and observe that each approach described in this section has some strength and suffers from some weakness.

Table 3: Strength and Weakness of the Existing Approaches.

Approach		Strength	Weakness
BIT approach		Increase component testability. Allow easy of maintenance.	Memory consumption problem. Static test cases.
Metadata-based testing approach		Increase component testability. Dynamic test case generation. No memory consumption problem. Generate test data.	Affect implementation transparency of the component. Don't handle heterogeneous components.
Testable architecture approach		Increase component testability. No memory consumption problem.	Affect implementation transparency of the component. Extra effort is demanded from developer during the maintenance time. Don't handle heterogeneous components.
Certification strategy approach	Third-party certification	Impartial testing. Increase the confidence in component services. Testing is unbiased as it's conducted by third party.	May be too costly for small organizations to afford.
	Developer certification	Test cases are available to user to re-execute. Facilitate users' understanding of component functionality.	Context-independent testing. Testing is biased by the component developer.
	User certification	Context-dependent testing. Increase reliability in the component services.	No test adequacy criteria for component reuse.

Journal of Theoretical and Applied Information Technology
20th January 2015. Vol.71 No.2

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

5. COMPARATIVE EVALUATION

In this section, we present a comparative evaluation of the most prominent techniques for integration testing of CBS that are discussed in the previous section with respect to a set of pre-defined criteria. At the first, the evaluation criteria will be identified and described and then the results and discussion of the results will be presented after applying the evaluation criteria into existing techniques.

5.1 The Evaluation Criteria

Similar to [40-42], we base our comparison on a set of pre-defined criteria. To this end, this section defines and explains briefly the evaluation criteria used in this study.

The evaluation criteria have been identified based on the discussion in the previous section. According to that, the evaluation criteria (as presented in Table 4) have used to perform the evaluation process, and the results of evaluating the existing techniques of integration testing of CBS are presented in Table 5. For simplicity, as shown in Table 4, the criteria are numbered like Q1 to represent first criteria, Q2 to represent second criteria and so on. The criteria used to evaluate the existing works in integration testing of CBS are described briefly as follow:

- ***Q1.*** This criterion deal with the test case definition. Component developer, component user, or independent tester can define test cases.

Table 4: The Criteria for Evaluating Existing Techniques.

#	Criteria
Q1	Who defines the test cases?
Q2	Who execute the test cases?
Q3	Is the technique required additional information for test derivation?
Q4	Is the technique required additional structure for test execution?
Q5	Does the test specification easily accessible?
Q6	Do the technique Handles heterogeneous components?
Q7	Does the technique have tool support for test case generation?
Q8	Does the technique have tool support for test data generation?

- ***Q2.*** This criterion deal with the test case execution. Component developer, component user, or independent tester can execute test cases.
- ***Q3.*** This criterion will help to check whether the integration testing technique requires additional information for test derivation or not.
- ***Q4.*** This criterion will help to examine whether the integration testing technique requires additional information for test execution or not.
- ***Q5.*** This criterion will help to examine whether the technique can easily exploit test information from components or not.
- ***Q6.*** This criterion will help to examine whether the integration testing technique has the ability to handle heterogeneous components or not.
- ***Q7.*** This criterion will help to examine whether the integration testing technique has a tool to support the test case generation or not.
- ***Q8.*** This criterion will help to examine whether the integration testing technique has a tool to support test data generation or not.

5.2 Comparative Evaluation Remarks and Discussion

Integration testing is an important activity in CBSD Lifecycle for developing reliable CBS. The aim of this study is to provide a classification and comparison of the most prominent techniques for integration testing of CBS, which were discussed in the previous section, with purpose to check how far the existing techniques can support the proposed technique. In that case, those techniques categorized into four main classes, namely BIT Approach, Metadata-based testing Approach, Testable Architecture Approach, and Certification Strategy Approach. We believe that, at this time it is not possible to claim that the presented classification is comprehensive.

An evaluation was performed on the existing techniques for integration testing of CBS based on eight identified criteria (questions) presented in Table 4. These eight criteria have been developed and used as evaluation parameters, to analyze the strengths and the weaknesses of the existing techniques of integration testing of CBS.

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

Table 5: Summary Results of the Comparative Evaluation.

Approach	Works	Evaluation Criteria							
		Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8
BIT approach	Edwards [21]	CD	CU	×	√	√	×	×	×
	Wang et al. [22]	CD	CD/ CU	×	√	√	×	×	×
	Atkinson et al. [23]	CD	CD/CU	×	√	×	√	×	×
	Martins et al. [25]	CD	CD/ CU	×	√	√	×	√	×
	Beydeda and Gruhn [26]	CD	CU	×	√	√	×	×	×
	Momotko and Zalewska [24]	CD	CD/ CU	×	√	×	√	×	×
	Mahmood [8]	CD	CD/ CU	×	√	√	×	√	×
Testable architecture approach	Ye et al. [32]	CU	CU	√	√	√	√	×	×
	Gao et al. [33]	CU	CU	×	√	√	√	×	×
	Jabeen and Rehman [34]	IT	IT/ CU	√	×	√	×	×	√
	Brohi and Jabeen [12]	CU	CU	√	×	√	×	×	√
Metadata-based testing approach	Wu et al. [6]	CU	CU	√	√	√	×	×	×
	Liu and Richardson [29]	CU	CU	√	√	√	×	√	√
	Silva et al. [30]	CU	CU	√	×	√	×	√	√
	Belli and Budnik [28]	CU	CU	√	√	√	×	√	√
	Naseer et al. [31]	CU	CU	√	×	√	×	√	√
Certification strategy approach	Yu-Seung et al. [37]	IT	IT	√	×	√	×	×	×
	Morris et al. [38]	CD	CD/ CU	√	×	√	×	√	×
	Voas [39]	CU	CU	×	×	√	×	×	×

Legend:
CD - Component developer. √ - Requirement satisfied.
CU - Component user. × - Requirement unsatisfied.
IT - Independent tester.

Based on that, the above evaluation criteria (as shown in Table 4) employed in existing works, and the results of the presented comparative evaluation of existing techniques of integration testing of CBS are shown in Table 5.

Generally, our results show that the majority of the existing techniques of integration testing of CBS demand additional information packaged with the component to facilitate the integration testing process, and/or additional structure for reliable use of component applications. Moreover, based on the result in Table 5, we conclude that each technique suffers from some drawbacks and have it is own limitations, but all techniques attempt to resolve the problems of missing information (internal structure) and the unavailability of component source code (implementation transparency), which is necessary for effectively utilizing the reuse benefit of components. Therefore, our research direction is to develop a new technique that tries to cover most, if not all of this drawback and limitations of the existing techniques, using the idea of learning and testing approach, which have become an essential strategy to solve many problems in the software engineering domain.

6. CURRENT ISSUES AND FUTURE DIRECTIONS

Based on the work discussed in previous sections, it is clear that many issues and challenges are left to be solved. We are summarizing some of the open research issues; those must be considered during the future work in order to facilitate the integration testing process.

The general problem that can be seen in a majority of existing works is the missing of component information (i.e., component source code), and lack of information exchanged between component developers and component users during the development of the component itself and during the development of CBS systems. Unfortunately, unavailable information limits the capacity of both component developers and component users to test candidate components efficiently during the integration testing. Researchers try to solve this

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

problem by using metadata, BIT and UML models, allowing reverse engineering and to access the component source code, thus affecting the implementation transparency of software components. Therefore, continues efforts and further works still required to tackle this problem.

Understanding component's behavior is another issue, especially when considering black-box components, due to the unavailability of their formal models, updated specifications or source code. Strategies or algorithms are needed to extract the behavior (state) model of integrated components. The extracted models will be used to detect faulty interactions (integration bugs) or compositional problems between integrated components during integration testing.

Distributed system issues. As a CBS is always built under a distributed operating environment, which will then come into all the issues of distributed systems, such as transaction controlling and deadlocks. These distributions related issues can only be detected during the integration phase, which then requires further testing effort. Moreover, CBS may even introduce versioning issues, which is caused by the coexistence of two different versions of a component in the system.

The last issue deal with analyzing and validating the component interaction. It is important to ensure that the components that are developed separately, work properly together. Therefore, several works are needed to handle the interactions between components during the integration testing phase.

7. CONCLUSION AND FUTURE WORK

In this study, most of the existing integration testing techniques of CBS was discussed and classified into four main approaches; namely BIT approach, metadata-based testing approach, testable architecture approach and certification strategy approach. The strengths and weaknesses of existing approaches were also highlighted. Furthermore, the techniques initially were evaluated based on eight pre-defined evaluation criteria. Nevertheless, this study is in the initial stage, and we do not claim that our classification and evaluation can be seen as exhaustive. Finally, some of the current issues for future work were highlighted. We conclude that there is a need for further research in the field of integration testing of CBS, and we hope this study will serve as an introductory review to those who are new to the subject.

As future work, this study would be extended to provide a comprehensive report, which contains a systematic literature review of state-of-the-art of integration testing of CBS. We hope the systematic literature review also will help to find out gaps and future directions in the area of integration testing of CBS. The final goal, based on the result of the comparative evaluation, these techniques will be extended to provide more comprehensive integration testing technique that addresses these limitations, and it will be validated using strength case studies.

ACKNOWLEDGEMENT

The authors would like to thank Universiti Teknologi Malaysia (UTM) for supporting this research. We also thank our Embedded & Real-Time Software Engineering Laboratory (EReTSEL) members, especially Mr. Muhammad Imran Babar, and the people of Software Engineering Research Group (SERG) at UTM for their continuous support.

REFERENCES:

[1] M. Abdellatief, A. B. M. Sultan, A. A. A. Ghani, and M. A. Jabar, "A mapping study to investigate component-based software system metrics," *Journal of Systems and Software,* vol. 86, pp. 587-603, 2013.

[2] A. E. Ali, "A comparative Study of Software Development Methodologies," M.Sc diss., Sudan University of Science and Technology, 2010.

[3] C. Szyperski, *Component Software: Beyond Object-Oriented Software*: Addison-Wesley, 1998.

[4] S. P. Shashank, P. Chakka, and D. V. Kumar, "A systematic literature survey of integration testing in component-based software engineering," in *Computer and Communication Technology (ICCCT), 2010 International Conference on*, 2010, pp. 562-568.

[5] J. Z. Gao, J. Tsao, and Y. W, *Testing and Quality Assurance for Component-Based Software*: Artech House, Inc., 2003.

[6] Y. Wu, M.-H. Chen, and J. Offutt, "UML-Based Integration Testing for Component-Based Software," in *COTS-Based Software Systems*. vol. 2580, H. Erdogmus and T. Weng, Eds., ed: Springer Berlin Heidelberg, 2003, pp. 251-260.

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

[7] W. Ye, P. Dai, and C. Mei-Hwa, "Techniques for testing component-based software," in *Engineering of Complex Computer Systems, 2001. Proceedings. Seventh IEEE International Conference on*, 2001, pp. 222-232.

[8] S. Mahmood, "Towards Component-Based System integration testing framework," *Lecture Notes in Engineering and Computer Science,* vol. 2, pp. 1231-1235, 2011.

[9] S. Beydeda and V. Gruhn, "State of the art in testing components," in *Quality Software, 2003. Proceedings. Third International Conference on*, 2003, pp. 146-153.

[10] S. Beydeda, "Research in testing COTS components - built-in testing approaches," in *Computer Systems and Applications, 2005. The 3rd ACS/IEEE International Conference on*, 2005, p. 101.

[11] M. J. U. Rehman, F. Jabeen, A. Bertolino, and A. Polini, "Testing software components for integration: A survey of issues and techniques," *Software Testing Verification and Reliability,* vol. 17, pp. 95-133, 2007.

[12] M. N. Brohi and F. Jabeen, "A Metadata-based Framework for Object-Oriented Component Testing," *International Journal of Computer Applications,* vol. 41, pp. 8-18, 2012.

[13] R. Groz, K. Li, and A. Petrenko, "Integration testing of communicating systems with unknown components," *annals of telecommunications - annales des télécommunications,* pp. 1-19, 2014.

[14] M. E. Khan, "Different Software Testing Levels for Detecting Errors," *International Journal of Software Engineering (IJSE),* vol. 2, 2011.

[15] G. Ning, S. Nakajima, and M. Pantel, "Hidden Markov model based automated fault localization for integration testing," in *Software Engineering and Service Science (ICSESS), 2013 4th IEEE International Conference on*, 2013, pp. 184-187.

[16] A. Bertolino and A. Polini, "A framework for component deployment testing," in *Software Engineering, 2003. Proceedings. 25th International Conference on*, 2003, pp. 221-231.

[17] P. D. L. Machado, J. C. A. Figueiredo, E. F. A. Lima, A. E. V. Barbosa, and H. S. Lima, "Component-based integration testing from UML interaction diagrams," in *Systems, Man and Cybernetics, 2007. ISIC. IEEE International Conference on*, 2007, pp. 2679-2686.

[18] U. A. Khan, I. A. Al-Bidewi, and K. Gupta, "Challenges in Component Based Software Engineering as the Technology of the Modern Era," *International Journal of Internet Computing (IJIC),* vol. 1, 2011.

[19] H. Reza and C. Liang, "Context-Based Testing of COTs Using Petri Nets," in *Information Technology: New Generations (ITNG), 2012 Ninth International Conference on*, 2012, pp. 572-577.

[20] M. Shahbaz and R. Groz, "Analysis and testing of black-box component-based systems by inferring partial models," *Software Testing, Verification and Reliability,* vol. 24, pp. 253-288, 2014.

[21] S. H. Edwards, "A framework for practical, automated black-box testing of component-based software," *Software Testing Verification and Reliability,* vol. 11, pp. 97-111, 2001.

[22] Y. Wang, G. King, and H. Wickburg, "A Method for Built-in Tests in Component-based Software Maintenance," presented at the Proceedings of the Third European Conference on Software Maintenance and Reengineering, 1999.

[23] C. Atkinson and H.-g. Groß, "Built-in contract testing in model-driven, component-based development," presented at the ICSR-7 Workshop on ComponentBased Development Processes, 2002.

[24] M. Momotko and L. Zalewska, "Component+ built-in testing a technology for testing software components," *Foundations of Computing and Decision Sciences,* vol. 29, pp. 133-148, 2004.

[25] E. Martins, C. M. Toyota, and R. L. Yanagawa, "Constructing self-testable software components," in *Dependable Systems and Networks, 2001. DSN 2001. International Conference on*, 2001, pp. 151-160.

[26] S. Beydeda and V. Gruhn, "Merging components and testing tools: the self-testing COTS components (STECC) strategy," in *Euromicro Conference, 2003. Proceedings. 29th*, 2003, pp. 107-114.

[27] A. Orso, H. Do, G. Rothermel, M. J. Harrold, and D. S. Rosenblum, "Using component metadata to regression test component-based software," *Software Testing, Verification and Reliability,* vol. 17, pp. 61-94, 2007.

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

[28] F. Belli and C. J. Budnik, "Towards self-testing of component-based software," in *Computer Software and Applications Conference, 2005. COMPSAC 2005. 29th Annual International*, 2005, pp. 205-210 Vol. 1.

[29] C. Liu and D. Richardson, "Software components with retrospectors," in *In International Workshop on the Role of Software Architecture in Testing and Analysis*, 1998.

[30] F. R. C. Silva, E. S. Almeida, and S. R. L. Meira, "An approach for component testing and its empirical validation," presented at the Proceedings of the 2009 ACM symposium on Applied Computing, Honolulu, Hawaii, 2009.

[31] F. Naseer, S. ur Rehman, and K. Hussain, "Using meta-data technique for component based black box testing," in *Emerging Technologies (ICET), 2010 6th International Conference on*, 2010, pp. 276-281.

[32] W. Ye, P. Dai, and C. Mei-Hwa, "Techniques for testing component-based software," in *Seventh IEEE International Proceedings Conference on Engineering of Complex Computer Systems, 2001*, 2001, pp. 222-232.

[33] J. Gao, K. Gupta, S. Gupta, and S. Shim, "On Building Testable Software Components COTS-Based Software Systems." vol. 2255, J. Dean and A. Gravel, Eds., ed: Springer Berlin / Heidelberg, 2002, pp. 108-121.

[34] F. Jabeen and M. Rehman, "A framework for object oriented component testing," in *Emerging Technologies, 2005. Proceedings of the IEEE Symposium on*, 2005, pp. 451-460.

[35] A. Alvaro, E. S. de Almeida, and S. R. de Lemos Meira, "Software component certification: a survey," in *Software Engineering and Advanced Applications, 2005. 31st EUROMICRO Conference on*, 2005, pp. 106-113.

[36] W. T. Councill, "Third-party testing and the quality of software components," *Software, IEEE,* vol. 16, pp. 55-57, 1999.

[37] M. Yu-Seung, O. Seung-Uk, B. Doo-Hwan, and K. Yong-Rae, "Framework for third party testing of component software," in *Software Engineering Conference, 2001. APSEC 2001. Eighth Asia-Pacific*, 2001, pp. 431-434.

[38] J. Morris, G. Lee, K. Parker, G. A. Bundell, and C. P. Lam, "Software component certification," *Computer,* vol. 34, pp. 30-36, 2001.

[39] J. M. Voas, "Certifying off-the-shelf software components," *Computer,* vol. 31, pp. 53-59, 1998.

[40] A. Abdelmaboud, D. N. A. Jawawi, I. Ghani, and A. Elsafi, "A Comparative Evaluation of State-of-the-Art Cloud Migration Optimization Approaches," in *Recent Advances on Soft Computing and Data Mining*. vol. 287, ed: Springer International Publishing, 2014, pp. 633-645.

[41] H. M. Rusli, M. Puteh, S. Ibrahim, and S. G. H. Tabatabaei, "A comparative evaluation of state-of-the-art web service composition testing approaches," presented at the Proceedings of the 6th International Workshop on Automation of Software Test, Waikiki, Honolulu, HI, USA, 2011.

[42] A. A. Mansor and W. M. Wan-Kadir, "A Comparative Evaluation of State-of-the-Art Approaches in the Design of an Adaptive Software System," in *International Conference on Software Technology and Engineering, 3rd (ICSTE 2011)*, 2011.