Volume 2, No. 2, May-June 2011
ISSN No. 0976-5697
International Journal of Advanced Research in Computer Science
RESEARCH PAPER
Available Online at www.ijarcs.info

Techniques for Developing Testable Component-Based Software: Similarities, Differences and Classification

Shyam S. Pandeya*
Department of Computer Engineering.
IT, BHU, Varanasi, India
pandeyashyam@gmail.com

Anil K. Tripathi
Department of Computer Engineering.
IT, BHU, Varanasi, India
anilkt@bhu.ac.in

Abstract: Works on testability of components or component-based software have proposed several techniques for increasing testability of component-based software systems. This work aims at reviewing these techniques for understanding their similarities and differences. It classifies the techniques in accordance with the nature of problems in component-based software testability. This helps in evaluating proposed techniques as per their contribution in solving the concerned problems. Further, this makes their relative efficiency explicit, and, lets us have an overview of major issues being taken up by previous works on component-based software testability. This has been used to arrive at current research gaps and questions in component-based software testability.
Keywords: Component-based software, COTS component, Testing, Testability.

I. INTRODUCTION

Component-based software systems are increasingly being used in almost all walks of human endeavour [7]. Failures of software systems in safety critical and mission critical systems can lead to the unprecedented loss of business, money and lives [2]. These problems require better testing and quality assurance techniques. Testability of a software system is an effective and viable technique of reducing the testing cost, and, increasing testing effectiveness [7]. Testability is not only the indicator of testing effectiveness, but, also, a measurable indicator of quality of a software development process [7]. Testing involves test-case generation, test-case execution and test evaluation. All the aspects of software development that ease these activities directly or indirectly make a software system more testable.

This work aims at reviewing testability techniques for component-based software in order to understand their similarities and differences. It classifies the techniques in accordance with the nature of problems in CBS testability. This makes their relative efficiency explicit and lets us have an overview of major issues in component-based software testability. This helps in arriving at current research gaps in component-based software testability. Section 2 gives an overview of component-based software testability techniques. In section 3, we categorise the existing testability techniques according to the theme of the problems taken up in these works. Section 4 discusses the studies so as to bring forth the newer research gaps. Next, in section 5, we draw conclusions. Section 6 is for references.

II. TESTABILITY TECHNIQUES FOR CBS SYSTEMS

Researchers have been concerned with testability in order to make software testing more efficient. CBS development aims at maximising reuse of COTS and in-house components. Widespread use of COTS and in-house components makes testability more crucial and relevant to software development. Making components testable can greatly increase the rate of successful reuse of COTS components. This explains why most of the techniques addressing CBS testability are concerned with component testability. Software testability encompasses all the aspects that ease software testing, from quality of its specification, design, code, and tests, to availability of test support [3]. Freedman (1991) investigated the meaning of testability and introduced the ideas of domain testability, observable extension, and, controllable extensions [1]. He proposed a metric model for measuring controllability and obsevability.

The study experimentally demonstrated that a component with domain testable specification can be built and tested in less time than that which is not having domain testable specification. Study stresses that testable component should not have the hidden inputs and outputs. Further, its output range should be equal to the set of outputs obtained by executing the inputs from domain of allowed inputs. Martins *etal.* (2001) presented an approach for construction and use of self-testable object-oriented components that are unique class [3]. According to the study, a testable component consists of the class under test, built-in-test capability, and, a test specification. Tool supporting implementation of assertions relative to class invariants and class methods pre and post conditions, test driver generation, test retrieval, and, test history creation and maintenance was presented in the study. Gao *etal.* (2002) introduced the concept of testable bean [21]. They showed a technique of constructing testable component based on testable architecture. Vincent and King (2002) was concerned with built-in-test support required at runtime [4].

Nguyen *etal.* (2002) proposed a code based analysis method of C or Ada components [5]. They measured controllability and observability based on information loss of modules. Based on above measures one can control and observe a module by choosing the flow with greater observability and controllability values. Collet *etal.* (2004) identified tracks for built-in-test concept for components [8]. Study proposed the structure of built-in testable components to be made of specification (contracts), component implementation, and test cases. In this model, confrontation of test-cases, contracts, and, specification increases testability of components. Gao and Shih (2005) proposed a qualitative model for component testability. It is comprised of five factors and sub-factors affecting the testability of a component. Study proposed a pentagram model for measuring component testability. Gross *etal.* (2005) proposed the use of models for generating built-in-test for automation and effort reduction [20]. Beydeda (2006) proposed an approach of self-testability which integrates

STECC method and metamorphic testing [11]. It encompasses test case generation and test evaluation.

Brenner *etal.* (2006) considered the need of run-time testing [12]. He stressed that due to the evolutionary nature of CBS systems built-in test support for accomplishing run-time testing has to be a concern for reducing testing effort. Liangli *etal.* (2006) identified the dependencies between components of software systems [10][13]. These dependencies can be supplied to users of component in form of metadata for generating test-cases when integrating components. Mao (2007) proposed the use of aspect-oriented programming for incorporating invariants, traceability and pre-condition for methods [14]. This way of modulating testability concerns helps in efficient evolution and maintenance of CBS systems. Liangli *etal.* (2007) analysed the meaning and ways to increase component testability [15]. Work proposed a way incorporating DU table and observation points based monitoring mechanism for generating test-Case and enhancing observability respectively [15]. Mao *etal.* (2007) considered the problem of regression testing of COTS component [19]. According to the work vendor can let user know the affected methods due to the changes made to the component. This information can be used to test affected methods using built-in-test support. Kanstern (2008) tried to find out the testability support required at architectural level based on interviews of two companies [16]. Aim was to understand the techniques used by companies for test automation, controllability and observability at architecture level. Study considered test implementation, control of messaging, simulation strategies and test functionality to understand the testability and test automation strategies being taken up. Gonzalez *etal.* (2009) introduced the qualitative model of run-time testability. The work introduced a technique for measurement of run-time testability. Gill and Tomar (2011) proposed a process to construct testable component [22].

III. PROBLEMS ADDRESSED BY CBS TESTABILITY

In order the understand CBS testability, we will try to concentrate on the problems of concern of various testability techniques. This gives a ground for categorising the proposed testability techniques. Further, this lets us understand the contribution of different techniques more explicitly. This helps in understanding the relative efficiency of different techniques, and, further, the current research gaps or questions.

What are the problems which are being considered under the purview of CBS testability? Below, we classify the testability techniques as per nature of the addressed problems. This would let us evaluate testability techniques in a realistic manner. Classification is not disjoint; rather, it has to do with major problems that have been considered by researchers. Following are the major problems that have been considered under purview of CBS testability.

A. Modelling Component testability

These studies considered the aspects related to the form of a testable component and the way testability can be analysed and measured. Freedman (1991) showed that it is essential to recognise hidden inputs and outputs [1]. Further, he stressed on the fact that we should specify output domain so that it matches the set of outputs caused by input values. This is because if we don't know what are inputs and outputs, then, it would be impossible to evaluate the result of test case execution. It may not be possible to specify outputs according to exact set of outputs obtained by executing the set of inputs in every case. We should be able to analyse the testability on the basis of its code and internal structure. This is important because certain decision regarding testability can be made only after the analysis of its structure or code. A work entitled "Testability Analysis for Software Components" aims at testability analysis of C or Ada component to recognise those control flows which can be used to test a part of code more easily [5]. A qualitative model incorporating the factors that affect component testability was proposed by Gao and Shih (2005) in a work entitled "A Component Testability Model for Verification and Measurement" [9]. This work collected the intuitive factors which affect testability of a component.

B. Facilitating User-Oriented Component Testing

Validating the COTS components as per an application context is a key step in successfully developing the CBS systems [18]. Many testability techniques are concerned with this issue. Major problem is to let users test COTS components as per their application context [11]. A work entitled "Constructing Self-Testable Software Components" [3] aims at providing built in test capability incorporating generation of test cases, implementation of assertions relative to class invariants and class methods pre and post conditions, test driver generation, test retrieval, test history creation and maintenance. Work entitled "Contract-based Testing: from Objects to components" is another built-in testability approach [8]. It proposes the structure of built-in testable component to be made of specification (contracts), component implementation, and test cases. It is assumed that confrontation of test-cases, contracts, and, specification increases the testability of a component. Work entitled "Self-Metamorphic testing Components" presents an approach to self-testability [11]. Unlike other approaches of self-testability, which assume a test case set and assumes means of test execution; this approach encompasses test case generation and test evaluation. It integrates STECC [11] method and metamorphic testing [11] to accomplish the specified features. Next, a work tries to capture the dependency between two components [10]. It devises a method of providing dependency information in form of metadata which can be used for generating test cases when integrating components. Work entitled "Construct Metadata Model on Coupling Information to Increase the Testability of Component-based Software" aims at facilitating testing by supplying user metadata for generating test cases based on definition-use criteria [15]. It also aims at increasing observability based observation points [15]. Work entitled "AOP – based testability Improvement for Component-based software" aims at providing technique for checking invariants of components and collecting pre-condition of a method execution [14]. Aspect oriented programming has been devised for accomplishing above tasks. This helps in modulating the separate concerns, and, facilitates maintenance and evolution of software systems.

C. Test Support and Automation at Architecture Level

A work entitled "A study on Design for testability in Component-Based Embedded Software" is concerned with test support at architecture level [16]. It identified that techniques for control of messaging, simulation of stubs and deployment environment, testing support in form of built-in

test, trace support and support for ad-hoc testing support are required at architectural level.

D. Deployment and run-time test support

Run-time testing is a viable option for integrating components that cannot be tested during traditional integration time testing [12] [17]. Performing tests during deployment or in-service time introduces interference problems, such as undesired side effect in the state of a system or the outside of the system. Major issue with run-time testability is devising models for analysing interference, and, come up with built-in test support and test infrastructure support to deal with the interference [12][17]. A qualitative model was proposed by the work entitled "A Model for the Measurement of the Runtime Testability of Component-based Systems" [17].

IV. DISCUSSIONS

Testability techniques for software systems aim at devising methods and guidelines, setting standards, and analysing artefacts so as to make testing easy. Many testability techniques have been proposed in connection with issues that can make testing easier. We will discuss the various challenges and problems falling in the categories presented in previous section. Next, we will discuss the problems in measuring component testability.

A. Challenges and Problems in Modelling Component Testability

What is a testable component? Jerry Gao has identified factors affecting component testability. According to him component testability depends on understadability, observability, component traceability and test support capability of a component. He further represented these attributes in terms of lower level factors. It is required to define these attributes using operational definitions. By operational, we mean in a way so that ambiguities of terms are removed. For example, from term Document readability [9], it is not clear, what are the symptoms of a readable document? How one can decide what is readable? Does it vary from document to document? Another challenge is to provide well defined guide lines to incorporate the above mentioned attributes. For example if one is interested in incorporating controllability, then, what is required to be done? What are steps that have to be followed? What analysis technique is required? What activities can be automated? Further, one needs to understand how exactly these factors make a component testable? Current issues concerning component testability is to provide the operational definition of each of the factors affecting testability. These operational underpinning of attributes not only make the deeper understanding of relevant concepts but also lets one define efficient metric models for the attributes of concern. Further, one needs to find out guidelines, analysis techniques and automation required to incorporate these attributes in software components.

B. Challenges and Problems in Facilitating User-Oriented Testing

User-oriented testing has to be accomplished without access to source code. This means that user needs to stick to some form of specification based testing. If user wants code-based testing, then, vendor of component needs to supply those test cases along with the component. Further, component should facilitate test execution and evaluation.

This is supported by self-testable component [3]. A self-testable component requires Built-in-test capabilities, test case set or some specification for creating test case set and infrastructure supporting self-testability [3]. Problem with this approach is that user can not choose test-criteria at his discretion. It is prefixed by the vendor of a component. An approach which integrates STECC method and metamorphic testing allows user to choose control flow based criteria. It uses BINTEST [6] approach for test-case generation [11].

One problem with this approach is that test cases generated in this context do not necessarily include expected results [11]. Next, a limitation of this approach is that it does not allow non-control flow based criteria. Further, execution performed during test-case generation can be computationally intensive, and, therefore, cannot be trivially neglected in every case. Another point is that metamorphic testing involves using metamorphic relation from domain knowledge. It has its own challenges. How to collect and specify these relations, and, inculcate in the code? Since satisfying these relations doesn't mean that a test result is positive, it needs to be evaluated for its benefits. In another work, addition of contracts based on specification of a component has been devised as a way to increase the observability [8]. This has some immediate challenges.

What are the guidelines to extract contracts from specification? How to add contracts when a failure is observed? How to partially activate contracts at run-time? Some studies propose metadata based technique for facilitating integration testing [10] [13] [15]. These techniques have to be evaluated with respect to integration testing techniques based on other form of specifications, and, relative efficiencies have to be understood. This is because they essentially demonstrate criteria based on metadata. Maintenance and evolution are important concerns of CBS systems. It is required to communicate the modifications made by a vendor of a component to the users of the component. Vendor also needs to provide built-in-test support for conducting the testing of only some methods based on the modifications. This is very similar to the original problem of facilitating the testing of a component based on test criteria chosen by a user at his discretion. In this case, some extra support for identifying the test cases for only some methods (which are supposed to be tested) are required.

Aspect – oriented programming has been shown to be applicable for checking invariants that a component should obey, and, to collect pre-conditions of a method execution [14]. What are the guidelines to use and collect aspects? Which tools can be used for these tasks? How to inculcate maintenance support in a systematic way? These questions should be answered in order to apply the above mentioned technique.

C. Challenges and Problems for Test and Automation Support at Architecture Level

Testability is an important concern at architectural level [16]. Test implementation, control of messaging, simulation of stubs and execution environment and built-in test support from components are challenges at architecture level [16]. What are effective ways to accomplish the above mentioned tasks? How can we support the testing of non-functional requirements at architectural level? It has been argued that

COTS components must be verified early in the software development life cycle for support of functional and non-functional requirements [18]. Do we need to implement and test a CBS system or any system in certain order? How can we prioritise testing order of functionalities of CBS systems? This can play a big role in reducing testing effort, and, can make testing more effective. How can we analyse the requirements of a system for prioritised implementation and testing?

D. Challenges and Problems for Run-Time Testing

Challenges in this category comprised of understanding and separating the requirements according to type of affect they produce on a running system. Next, each of the requirements has to be analysed for the possible test support required. In order to meet these challenges, we need to develop models and techniques to understand the way the components interact with each other, and, the affect functionalities to be tested have on functionalities of other components. Once it is understood, it is required to provide the test support to deal with the problem of interference [17].

E. Challenges and problems in measuring component testability

Since, testability may be related to almost all the activities of the software development life cycle, a testability technique needs to be evaluated in terms goodness of solution of the practical problems it helps in solving so as to make testing easier. This implies that we cannot compare testability gain due to a specification standard with that of testability gain resulting from coding standard. This is because we do not have so much refined understanding of testability gain obtained by following these two standards. Both techniques help in making testing easier in different ways. Though, gain due to one may be far more important than that of other. It is more important to come up with taxonomy of the issues and problems which exist in component/CBS testing, and, refine and compare those techniques which are local to an issue or problem. Comparing techniques across issues might not be meaningful, as, testability measure in one case might be measuring very different attribute than in another case. Let us understand it by an example. We can have a readable document which will definitely make testing easier. But it cannot be compared with a observability measure. Since, observability measures the extent to which values and states of a component or class can be observed. Both are measuring completely different attributes. It is not reasonable to compare these measures. Further, we cannot come up with single measure of testability by combining these two measures by multiplication, division or some other combination of arithmetic operations. This can be done only if it is know that readability and observability are affecting testability in well understood way.

V. CONCLUSIONS

Major issues in component and CBS testability are modelling component testability, providing support for user-oriented testing, test support and automation at architectural level, run-time testing support and measurement of testability. Modelling component testability rests on providing operational definition of intuitive factors. This helps in devising better methods and comparison of two or more techniques proposed for same goal. Support for user oriented testing mostly resides on the extent to which user is able to choose testing criteria at his discretion. This problem of facilitating code based testing is a prime concern for user-oriented test support. Other concerns are incorporating contracts, assertions, pre and post conditions, traceability support and metamorphic relations to enhance the capability of a component in detecting its own faults, and, facilitate maintenance of CBS (and components). A systematic method incorporating well defined guidelines and tool support is required for these activities. Problem with these solutions to partial oracle is that these may be required to be activated as per needs of application as it might be computationally intensive to check these at run-time. This needs some form of analysis which can be used to activate these checks partially as per needs. Another issue is facilitating users to select test cases only for methods which have been modified by vendors. This support for regression testing is important as component can be changed often due to the faults and performance reasons.

Test implementation, control of messaging, simulation of stubs and execution environment and built-in test support from components are challenges at architecture level. Well defined guidelines for controlling and observing component state and behaviour at different levels are required to effectively test and locate faults. Run-time testability is concerned with models and techniques to effectively understand the interference points and provide test support for them. Measurement of testability is important to better understand the contribution of different factors. But, first we need to provide operational definitions of different attributes. Next, there effects on testability have to be understood. We cannot compare the measures of two attributes whose effects on testability are not well understood.

VI. REFERENCES

[1] R. S. Freedman, “Testability of Software Components,” IEEE Trans. Softw. Eng., vol. 17, June 1991, pp. 553-564, doi=10.1109/32.87281.

[2] E. J. Weyuker, “Testing Component-Based Software: A Cautionary Tale,” IEEE Softw., vol. 15, September 1998, pp. 54-59, doi=10.1109/52.714817

[3] E. Martins, C. M. Toyota, and R. L. Yanagawa, 2001. “Constructing Self-Testable Software Components,” Proc. 2001 International Conference Dependable Systems and Networks (formerly: FTCS) (DSN '01), IEEE Computer Society, 2001, pp. 151-160.

[4] J. Vincent, G. King, P. Lay, and J. Kinghorn, “Principles of Built-In-Test for Run-Time-Testability in Component-Based Software Systems,” Software Quality Control, vol. 10, September 2002, 115-133, doi=10.1023/A:1020571806877.

[5] T. B. Nguyen, M. Delaunay, and C. Robach, “Testability Analysis for Software Components,” Proc. International Conference Software Maintenance (ICSM'02) (ICSM '02), IEEE Computer Society, 2002, pp. 422-

[6] S. Beydeda and V. Gruhn, “BINTEST - Binary Search-based Test Case Generation,” Proc. 27th Annual International Conference Computer Software and Applications (COMPSAC '03), IEEE Computer Society, 2003, pp. 28-.

[7] J. Z. Gao, J. Tsao, Y. Wu, and T. H.-S. Jacob. Testing and Quality Assurance for Component-Based Software. Artech House, Inc., Norwood, MA, USA, 2003.

[8] P. Collet, D. Deveaux, and R. Rousseau, "Contract-based testing: from objects to components," Proc. First International Workshop Testability Assessment, Nov. 2004, pp. 5-14, doi:10.1109/IWOTA.2004.1428408

[9] J. Gao and M.-C. Shih, "A component testability model for verification and measurement," Proc. 29th annual international conference Computer software and applications conference (COMPSAC-W'05). IEEE Computer Society, 2005, pp. 211-218.

[10] L. Ma, H. Wang, and Y. Lu, "The Design of Dependency Relationships Matrix to improve the testability of Component-based Software," Proc. Sixth International Conference Quality Software (QSIC '06), IEEE Computer Society, 2006, pp. 93-98, doi=10.1109/QSIC.2006.64

[11] S. Beydeda, "Self-Metamorphic-Testing Components," Proc. 30th Annual International Conference Computer Software and Applications (COMPSAC '06), IEEE Computer Society, 2006, pp. 265-272, doi=10.1109/COMPSAC.2006.161

[12] D. Brenner, C. Atkinson, R. Malaka, M. Merdes, B. Paech, and D.Suliman, "Reducing verification effort in component-based software engineering through built-in testing," Proc. 10th international IEEE conference Enterprise Distributed Object Computing, July 2007, pp. 151-162, DOI=10.1007/s10796-007-9029-4.

[13] M. Liangli, W. Houxiang, and L. Yongjie, 2006. "Using Component Metadata based on Dependency Relationships Matrix to improve the Testability of Component-based Software," Proc. 1st International Conference Digital Information Management, 2006, pp. 13-18.

[14] C. Mao, "AOP-based Testability Improvement for Component-based Software," Proc. 31st Annual International Conference Computer Software and Applications (COMPSAC '07), IEEE Computer Society, 2007, pp. 547-552. DOI=10.1109/COMPSAC. 2007.76

[15] M. Liangli, W. Houxiang, and L. Yongjie, "Construct Metadata Model based on Coupling Information to Increase the Testability of Component-based Software," International Conference Computer Systems and Applications, IEEE, 2007, pp. 24-31

[16] T. Kanstr\&\#233;n, "A Study on Design for Testability in Component-Based Embedded Software," Proc. Sixth International Conference Software Engineering Research, Management and Applications (SERA '08), IEEE Computer Society, 2008, pp. 31-38, DOI=10.1109/SERA.2008.11

[17] A. Gonz\&\#225;lez, \&\#201;. Piel, and H.-G. Gross. 2009. A Model for the Measurement of the Runtime Testability of Component-Based Systems. Proc. IEEE International Conference Software Testing, Verification, and Validation Workshops (ICSTW '09). IEEE Computer Society, 2009, pp. 19-28. DOI=10.1109/ICSTW.2009.9.

[18] S. S. Pandeya and A. K. Tripathi, "Testing Componnt-Based Software: What It has to do with Design and Component Selection," Journal of Software Engineering and Applications, vol. 4, Jan. 2011, pp. 37-47, doi: 10.4236/jsea.2011.41005

[19] C. Mao, Y. Lu, and J. Zhang, "Regression testing for component-based software via built-in test design," Proc. ACM symposium Applied computing (SAC '07), ACM, 2007, pp. 1416-1421 doi=10.1145/1244002. 1244307

[20] H.-G. Gross, I. Schieferdecker, and George Din, 2005. "Model-Based Built-In Tests," Electron. Notes Theor. Comput. Sci., vol. 111, January 2005, pp. 161-182. doi=10.1016/j.entcs.2004.12.001

[21] J. Z. Gao, K. K. Gupta, S. Gupta, and S. S. Y. Shim. 2002. "On Building Testable Software Components," Proc. First International Conference COTS-Based Software Systems (ICCBSS '02), Springer-Verlag, London, UK, 2002, pp. 108-121.

[22] N. S. Gill and P. Tomar, "New and innovative process to construct testable component with systematic approach," SIGSOFT Softw. Eng. Notes, vol. 36 January 2011, pp. 1-4. doi=10.1145/1921532.1921540

