ABSTRACT

In Component-Based Software Engineering (CBSE), software systems are mainly constructed with reusable components, such as third-party components and in-house built components. Component Based Software Development (CBSD) is used for making the software applications quickly and rapidly. In Component Based Development (CBD), the software product is built by gathering different components of existing software from different vendors. This process reduces cost and time of the software product. But for a tester, many difficulties arise in testing phase because the tester has a limited access to source code of reusable component of the product. This concept is known as Black-Box Testing (BBT) of software components because Black box testing is used where source code of the component is not available. The additional information with the components can be used to facilitate testing. This paper has its focus on testing of an application using Finite Automata-based testing which covers two types of testing, viz. NFA-based testing and DFA-based testing. The working of the application is explained with the help of UML diagrams.

Keywords: Component Based Software Development (CBSD), Unified Modeling Language (UML), NFA, DFA, Software Testing.
This paper is organized as follows. Firstly, it gives an introduction to the Component-Based Software Development process. Section 2 contains the related research papers used for the survey process. Section 3 concludes the paper and the future work is present in the last section.

2. Literature Survey

In the past decades, there were number of publications that discussed the concepts of component testing. This section contains a review of these works and relates them with the research work presented in this paper.

In the year 1996, Rothermel et. al. [1] proposed a paper to state that regression testing is an expensive maintenance activity that aims at revealing that code is not badly affected by the changes. Regression test selection techniques reuse the tests from the existing test suite for testing a modified program. Many kinds of regression test selection techniques had been proposed, however, it was difficult to evaluate and compare these techniques because they all had different goals. This paper outlined the issues relevant to regression test selection techniques, and these issues are used as the basis for a framework within which it evaluated the techniques. It is used to evaluate existing regression test selection techniques. The evaluation revealed the strengths and weaknesses of existing techniques, and highlighted some problems that future work in this area should address.

In the year 1998, Kropp et. al. [2] presented a paper which stated that the designers of Mission-critical system have to use a commercial off-the-shelf approach for reducing the costs and shorten the development time, even if COTS software components may not be designed specifically for robust operation. Automated Testing accesses component’s robustness without sacrificing the advantages of COTS approach. This paper described the Ballista method for scalable, portable and automated robustness testing of component interfaces. An object-oriented approach based on the parameter data types rather than component functionality eliminated the need for functionality-specific test scaffolding. A full-scale implementation that automatically tested the robustness of 233 O.S (operating system) software components had been ported to ten POSIX systems. 42% to 63% of the components that were tested had robustness problems. The normalized failure rate ranged from 10% to 23% of the tests conducted. Robustness Testing was used by the developers for measuring and improving the robustness, or even by consumers for comparing the robustness of COTS component libraries.

In the year 2001, Jean et. al. [3] proposed a paper in which regression testing was applied to the modified software for providing the confidence that the parts that are changed are behaving as they are supposed to behave and that the parts that are not changed are not badly affected by the modifications. In order to reduce the cost of regression testing, the various test cases were selected from the test suite that was used for testing the original version of the software and this process was called regression-test-selection. A safe regression-test-selection algorithm selected every test case in the test suite that revealed a fault in the modified software. Safe regression-test-selection technique was based on the use of a suitable representation, handled the features of the Java language. Unlike the other safe regression-test-selection techniques, this technique also handled incomplete programs. The technique was thus safely applied in case of Java software that used external libraries of components; the analysis of the external code was not required for the technique to select test cases for such software. The paper also described that regression-test-selection algorithm was effective in reducing the size of the test suite.

In the year 2001, Orso et. al. [4] presented a paper in which it was stated that the Component-based software technologies are viewed essential for creating the future software systems. This paper also stated that the use of externally provided components have some serious drawbacks for a wide range of software engineering activities, often because lack of information available about the components. In the previous years, Alessandro Orso et. al. proposed the use of component meta-contents (additional data and methods provided with a component) to support software engineering tasks. The author also presented two new meta-content based techniques that addressed the problem of regression test selection for component-based applications (a code-based approach and a specification-based approach). After illustrating these two techniques, this paper included a case study that applied the code-based technique to a real component-based system. The study of this system had shown that on
average, 26% of the overall testing effort was saved over seven releases and that too with maximum savings of 99% over one version.

In the year 2001, Yoon et al. [5] presented a paper which proposed a new testing technique for component composition in Enterprise Java Beans (EJBs). This paper also defined the components made by a current developer as white-box components and components made by another developer as black-box components. Software from Component-Based Software Development (CBSD) consists of black-box components and white-box components and composition errors results from the interaction between black-box components and white-box components or the interaction between two white-box components. The technique proposed in this paper tested these composition errors. This paper had selected test cases by injecting a fault into specific parts of the white-box component. The specific parts defined in this paper lead to the high effectiveness of this technique. This paper evaluated that the effectiveness through an experiment and a theorem. In addition, the paper provided an example in Enterprise Java Beans (EJB).

In the year 2001, Beydeda et al. [6] presented a paper which stated that the basic idea behind Component-Based Development (CBD) is to use the existing components for building new software. The resulting software often has features that complicate testing, for example, the feature of absence of component’s source code. This paper proposed an approach for testing, which explicitly took into account testing relevant features of component-based software and thus allowed more rigorous testing. The basic constituent of the approach was graphical representation combining-black and white-box information from specification and implementation, respectively. This graphical representation was then be used for test case identification on well-known structural techniques.

In the year 2001, Seung Ma et al. [7] presented a paper which stated that the qualities of constituent components have to be assured in order to ensure that a component-based software system run effectively and properly. Third-party certification was a safe and good approach that buyers can trust while dealing with the component software. The third-party certification methods at that time not supported proper functional testing, which was the most important factor while selected the component. This paper suggested a framework of third-party component testing of functionality which satisfied the given constraint i.e. evaluation of large number of components within a short period of time in a very cost-effective way. This framework identified the metadata that a component developer should provide to the third-party tester and defined the process for third-party testing using these metadata. To evaluate this framework, the author conducted several experiment with Component Test Manager (CTM), prototype tool for EJB component testing. The result had shown that the metadata were positively necessary and the participation of a component developer was the key to the third-party testing.

In the year 2001, Ye Wu et al. [8] presented a paper which stated that the Component-Based Software Engineering (CBSE) is increasingly being adopted for software development. Although much work have been proposed for building component-based systems, techniques for testing component-based systems have not been developed. This paper presented a test model that depicted a generic infrastructure of component-based systems and suggested the key test elements. The test model was realized by using a Component Interaction Graph (CIG) in which the dependence relationships and the interactions among the components were illustrated. By utilizing the CIG technique, the author proposed a family of test adequacy criteria which allowed optimization of the balancing among schedule, budget and quality requirements typically needed in software development. The method proposed was efficient and effective, as demonstrated by the promising results obtained from a case study.

In the year 2003, Bertolino et al. [9] presented a paper which stated that the Component-based development is the emerging paradigm in software production, though several challenges still slow down its full taking up. In particular, the “component trust problem” referred to how adequate guarantees and documentation about a component’s behaviours to be transferred from the component developer to its potential users. The capability to test a component deployed within the target application environment helped to establish the compliance of a candidate component to the customer’s expectations and certainly contributed to “increase trust”. To this purpose,
this paper proposed the CDT framework for Component Deployment Testing (CDT). CDT provided the customer with techniques to early specify a deployment test suite and an environment for running and reusing the specified tests on any component implementation. The framework was also being used to deliver the component developer’s test suite and to later re-execute it. The central feature of CDT was the complete decoupling between the specification tests and the component implementation.

In the year 2003, Lee et. al. [10] presented a paper which stated that identifying a reusable component and its functionality is important to heighten the productivity of component-based software. A component repository is required for storing and maintaining reusable software components. This paper described the component repository for the reuse of software component in component-based software development. This component repository provided a function for component reuse process such as specification viewing, adapting, testing and deploying. Through the component repository with these functions, components were efficiently reused in component reuse process.

In the year 2003, Beydeda et. al. [11] presented a paper which stated that the development of a software system from existing components surely have various benefits, but also entails a series of problems. One type of problem is caused by a limited exchange of information between the developer and the user of a component. A limited exchange and thereby a lack of information have various consequences, among them the requirement to test a component prior to its integration into a software system. A lack of information not only makes test prior to integration necessary, it also complicates these tasks. This paper proposed a new strategy to testing components and making components testable. The basic idea of the strategy was to merge components and testing tools in order to make components capable of testing their own methods. Such components allowed their thorough testing without disclosing detailed information, such as source code. This strategy fulfilled the needs of both the developer and the user of a component.

In the year 2003, Ye Wu et. al. [12] presented a paper which stated that component-based software engineering is increasingly being adopted for software development. This approach relies on using reusable components as the building blocks for constructing software. On the one hand, this helps to improve software quality and productivity. On the other hand, it necessitates frequent maintenance activities. The cost of maintenance for conventional software accounts for as much as two-thirds of the total cost, and it is likely to be more for component-based software. This paper presented a UML-based technique that attempted to help resolve difficulties introduced by the implementation transparent characteristics of component-based software systems. This technique was also useful for other maintenance activities. For corrective maintenance activities, the technique started with UML diagrams that represented changes to a component, and used them to support regression testing. To accommodate this approach for perfective maintenance activities, more challenges were encountered. This paper provided a UML-based framework to evaluate the similarities of the old and new components, and corresponding retesting strategies were provided.

In the year 2004, Heckel et. al. [13] presented a paper which stated that the Component-based technology increases reuse and productivity, but high-quality component-based systems are often difficult to implement. Component developers do not know the systems where the components are to be used, while software engineers developed new systems with limited knowledge on available components. The paper proposed a new technique that generated, at the time of component development, integration test cases from the specification of the behaviour expected from other components of the system. The specification was provided by executable graph transformation rules that were visualized by a UML-based notation. Test cases were executed at an early stage to validate the integration of the component with the expected behaviour of the system, and then were re-executed with concrete components at deployment time. The technique presented in this paper supported both the component developer, who early tested the integration of the components with the system, and the software engineers, who tested the concrete components at deployment time, simply by reusing existing test cases.
In the year 2005, Beydeda [14] presented a paper. This paper stated that the use of components in the development of large software systems have various benefits. Their testing, however, is still an open issue in software engineering. The user of a component is often faced with the problem that information necessary for testing purpose is not available. This lack of information distinguishes the testing of components from other software entities known from non-component-based development. This article gave an overview of the approaches to testing components. Besides the overview, this article also outlined some of the limitations of these approaches. Testing components was still an open problem.

In the year 2005, Belli et. al. [15] presented a paper which stated that the conventional test methods that are widely accepted are not necessarily adequate for testing of Component-Based Software (CBS). As a consequence, also conventional test tools cause similar problems for the test automation of CBS based on their Graphical User Interfaces (GUI), because for any level of user-focused testing domain knowledge and knowledge about the implementation of the CBS are essential to run the tests. The component manufacturer, on the other hand, is usually not willing to deliver the code to protect his/her commercial interest. For solving this conflict, this paper introduced a framework for the automation of user-oriented component testing that significantly reduced the test costs. The concept was based on black-box testing techniques and utilized the common features of commercial capture/replay test tools.

In the year 2005, Gao et. al. [16] presented a paper which stated that the components are the major building blocks for component-based systems, developing high quality components become very critical for component-based software engineering. To generate high quality components, the author had paid attention to component testability to ensure that the reusable components not only were tested by component vendors, but were also easily validated by component users. Therefore, component testability analysis, verification and measurement became very important research topic in testing components and component-based systems. This paper discussed the component testability in a quantifiable approach based on a component testability analysis model. Engineers used this model to verify and measure component testability during a component development process. Based on this testability model, the paper discussed testability verification and proposed a pentagram model for testability measurement.

In the year 2005, Gao et. al. [17] presented a paper stating that constructing component-based software using reusable components is becoming a widely used approach. Since the quality of a component-based system is highly dependent on the quality of its components, component quality validation becomes very critical to both component vendors and users. Effectively validating component quality needs adequate test models and testing coverage criteria. This paper proposed an adequate test model and test coverage criteria for component validation. The paper discussed a dynamic approach to analyze component test coverage based on the proposed test model and test coverage criteria. The major contribution of this paper was its dynamic test coverage analysis solution to monitor API-based component validation and reuse. The paper reported the recent development efforts of a component test coverage analysis tool, and presented an application example.

In the year 2005, Jiang et. al. [18] presented a paper which stated that testing plays an important role in the maintenance of component-based software development. Test adequacy for component testing is one of the hardest issues for component testing. To tackle this problem, it is a natural idea to apply mutation testing, which is a fault-based testing method used for test adequacy, for component contracts, whose aim is to improve the testability of the component. Though powerful, mutation testing is usually very computation-expensive, as many mutants need to be produced and executed in mutation testing. In this paper, the author proposed a contract-based mutation testing technique for testing components. Based on the discordance between contracts and specification, this approach employed a set of high level contract mutation operators. The experimental results had shown that these operators greatly reduced the number of mutants compared with traditional mutation operators. At the same time, the contract-based mutation using these contract mutation operators provided almost the same ability as that of using traditional mutation operators. Moreover, effective test suite was produced to reduce the maintenance effort.
In the year 2005, Yao et. al. [19] presented a paper which stated that Component-Based Software Engineering (CBSE) is an influential trend in software engineering. Adopting component-based technique, a system was constructed by synthesis of various distributed components. Based on the CORBA (Component Object Request Broker Architecture) architecture and Java technology, this paper provided an environment to allow a client-side software component to define tests for a black-box component published on the server-side. This technique simplified test execution, test results check and report, and supported test reuse and test automation. The paper revealed a practical approach to test software components by enhancing software component testability and test re-usability. The incremental testing framework introduced in this paper was helpful in saving time, energy and cost required for testing distributed components and for enhancing software quality. A testing supporting tool was implemented to facilitate distributed component testing based on CORBA.

In the year 2005, Mao et. al. [20] presented a paper which stated that the component-based software have been widely used in various application domains and become a fairly popular software form. However, due to lack of information about the externally-developed components, system testers (i.e. component users) generally not perform effective testing (especially regression testing) on their component-based systems. Component users do not know the details about change in component, so they are not able to select the proper test cases to retest the modified system. In this paper, the author presented an improved regression testing method based on the enhanced change information of component version to test the software system containing some modified components. It was a collaborative testing method that needed the joint particulars of component developer and user. Component developers calculate the change information from labeller method call graph and provide it to component users via XML files. Component users use this change information and their instrumentation records together to pick out test cases for next-round testing. In addition, the paper employed preliminary on some medium scale systems, the experimental results had shown that the regression testing method was fairly feasible and cost-effective in practice.

In the year 2005, Vincenzi et. al. [21] presented a paper which stated that software testing is a crucial activity in the software development process, only recently has more sound and consistent testing tools become available for testing Java programs and their components. These tools support mostly functional and control-flow based coverage criteria. This paper explored control-flow and data-flow based coverage criteria to support the testing of Java programs and/or components. It also described a testing tool, named JaBUTi, which was used by both the component developer and the component user to test Java-based components and/or systems. To achieve this goal, the tool worked at the byte code level such that no source code was required during the testing activity. It illustrated these ideas and concepts with an example extracted from the literature.

In the year 2006, Brenner et. al. [22] presented a paper which stated that the component and service-based technologies plays a central role in many aspects of enterprise computing. However, although the technologies used to define, implement and assemble components have improved significantly over recent years, techniques for verifying systems created from them have changed very little. The correctness and reliability of component-based systems are usually checked using the traditional testing techniques that are in use before components and services become widespread, and the associated costs and overheads remain high. This paper presented an approach which addressed this problem by making the system verification process more component-oriented. Based on the notion of built-in tests - tests that were packaged and distributed with prefabricated, off-the shelf components – the approach and supporting infrastructure helped to automate some of the testing process, thereby significantly reduced system testing effort. After providing an introduction to the principles behind component-based verification, and explaining the main features of the approach, the author had shown by means of a small example how it reduced system verification effort.

In the year 2006, Gao et. al. [23] presented a paper which stated that in Component-Based Software Engineering (CBSE), software systems are mainly constructed based on reusable components, such as third-party components and in-house built components. Hence, system quality depends on the quality of the involved components. Any change of a component must be re-tested at the unit-level and re-integrated to form component-based application...
systems. Although a number of recently published papers addressed regression testing and maintenance of component-based systems, very few papers discussed how to identify component changes and impacts at the unit-level and found out the reusable test cases in a component’s test suite to support its evolution. This paper focused on component API-based changes and impacts and proposed a systematic retest method for software components based on a component API-based test model. The proposed method had been implemented in a component test tool, known as COMP Test. It was used to automatically identify component-based API changes and impacts, as well as reusable test cases in a component test suite. The paper also reported this tool and its application results.

In the year 2007, Mariani et. al. [24] presented a paper which stated that the software engineers frequently update COTS components integrated in component-based systems and often chose among many candidates produced by different vendors. This paper tackled both the problems of quickly identifying components that were syntactically compatible with the interface specifications, but badly integrated in target systems and the problem of automatically generating regression test suites. The technique proposed in this paper on behavioural models that represented component interactions and were automatically generated while executing the original test suites on previous versions of target systems.

In the year 2007, Mao [25] presented a paper which stated some of the specialities of component, such as high evolvability, implementation transparent and limited access support that brings a great challenge for testing the systems built by externally-provided components, especially for regression testing. Built-in test design is a fairly effective way to improve component’s testability. This paper presented a built-in regression testing method to validate the change and its impact of component-based software, which needed the mutual collaboration between the component developers and component users. Through employing preliminary experiments on some medium scale systems, this regression testing method based on built-in test design was proved to be feasible and at statement level, it needed less exchange information (i.e. meta-data) and test scripts, so it was more cost-effective.

In the year 2007, Hamlet [26] proposed a paper which stated that several program-analysis schemes make unusual use of information derived from execution sampling. From finite test sets these techniques derive measures with wider meaning, which are then exploited in novel ways. The test information uses beyond its actual limitations is called a test-based specification. The idea is quite different from the usual software specification, which is a priori statement of what a program should do. Rather, a test-based specification is an empirical approximation to what a program actually does. The great virtue in analysis using test-based specifications is that (in contrast to the usual software specifications) test-based analysis is decidable and automatic. A test-based specification system for Component-Based Software Development (CBSD) has been implemented over the past five years, along with an extension of fundamental testing theory to precisely describe its properties. The CBSD tools provide an ideal context for experiments to study test-based specification, provide insights into sub-domain testing, composition of test results, and especially the role that program persistent state plays in testing and analysis. This paper described the CBSD theory and tools, listed the insights gained and suggested new ways to think about and practice testing using test-based specifications.

In the year 2007, Liangli et. al. [27] presented a paper which stated that a software component must be tested every time it is reused, to guarantee the quality of both the component itself and the system in which it is to be integrated. So how to increase testability of component has become a key technology in the software engineering community. This paper introduced a method to increase the component testability. Firstly the author analyzed the meanings of component testability and the effective ways to increase testability. Then the author gave some definitions coupling testing criterion. And the author further gave the definitions of DU-I (Definition-Use Information) and OP-Vs (Observation-Point Values). Based on these, the paper introduced a definition-use table, which included DU-I and OP-Vs item, to help component testers understanding and observing the component better. Then a framework of testable component based on DU-table was given. These facilities provided ways to detect errors, to observe state variables by observation-points based monitor mechanism. And the paper adopted coupling-based testing using information DU-table provided. Lastly, it applied the method to the application software developed before and generated some test cases. And the method was compared with Orso method and

**Karambir, Damini Verma**

*International Journal of Advances in Computing and Information Technology* 481
Kan method using the same example, presented the comparison results. The relevant results illustrated the validity of the method, effectively generated test cases and killed more mutants.

In the year 2007, Hou et. al. [28] presented a paper which stated that Regression Testing plays an important role in software maintenance, usually relies on test adequacy criteria to select and prioritize test cases. However, with the wide use and reuse of back-box components, such as reusable class libraries and COTS components, it is challenging to establish test adequacy criteria for testing software systems built on components whose source code is not available. Without source code or detailed documents, the misunderstanding between the system integrators and component providers become a main factor of causing faults in component-based software. This paper applied mutation on interface contracts, which described the rights and obligations between component users and providers, to simulate the faults that might occur in this way of software development. The mutation adequacy score for killing the mutants of interface contracts served as a test adequacy criterion. It performed an experimental study on three subject systems to evaluate the proposed approach together with four other existing criteria. The experimental results had shown that this adequacy criterion was helpful for both selecting good-quality test cases and scheduling test cases in an order of exposing faults quickly in regression testing of component-based software.

In the year 2008, Zheng et. al. [29] presented a paper which presented a new contract-based Software Component Testing (SCT) technique, Test by Contract (TbC), which extended the design by contract concept to the SCT domain and leveraged it with UML-based testing at the modelling level to design model-level test contracts for UML-based SCT. This paper introduced a new concept of contract for testability as the principal TbC goal and developed a set of important contract-oriented concepts (e.g. test contract, effectual contract scope, and internal/external test contract), and useful test criteria for effective model-based testability improvement. A practical step-wise TbC working process was developed to show how to put the TbC technique into practice for contract-based testing activities to undertake UML-based SCT with a case study.

In the year 2008, Bin et. al. [30] presented a paper which stated that Component-based software developing raise software developing efficiency and quality effectively. Component users configure tested component to obtain part of functions. Because of unavailable source code, errors of component configuration are difficult to be tested. This paper tested component configuration by using mutation testing, proposed the framework of component configuration testing based on mutation, increased the quality of component configuration testing based on mutation testing and the degree of automation.

In the year 2008, Liangming et. al. [31] presented a paper which stated that the key factor of component-based software development is component composition technology. Test-Driven Development (TDD) was a software development methodology for achieving high reliability. The combination of these two technologies helped in locating the component mismatch quite early and in reducing the cost of system testing. According to the component’s interaction graph, this paper analyzed the processes of Testing Based Components Composition (TBCC) and introduced the method as well as proposed an algorithm. By applying testing ideas to component composition, it helped to control the composition process and laid foundation for high quality component software.

In the year 2010, Jiang et. al. [32] presented paper which stated that the Component-Based Software Development (CBSD) gained popularity in previous years. In this way of software development, software components, which are typically black-box components, are intensively reused to construct new systems. The existing component models describe the characteristics of component from different aspects. However, there is not certain information about component testing. This paper proposed an extended component model that supported the component testing and reuse. A model for component specification and composition was established through extending the semantic specification, the protocol specification and the testing specification in the component interface. Then the paper gave a model for component implementation based on Web Services. Finally, it discussed the validation of model and it supported the development, testing and reuse of component.
In the year 2010, Jiang et. al. [33] presented a paper which stated that the adequate testing of black-box components is an important basis before they are to be reused in the approach of Component Based Software Development. The test-data generation and test adequacy ensuring are difficult issues for the unavailability of source code of black-box components. In this paper, an extended component interface specification model was proposed to support the component understanding, testing and reuse. Then the function of different kinds of specification elements in testing was defined. Based on the syntactic and semantic specifications, the proposed test-data generation method produced test suite meeting a certain mutation score, which was viewed as a kind of effective test adequacy criterion. Finally, some experiments were carried and the results had shown that the different kinds of specification supported the testing of black-box components.

In the year 2010, Naseer et. al. [34] presented a paper which stated that the Component Based Software Development is used for making the software applications quickly and rapidly. In Component Based Development, the software product is built by gathering different components of existing software from different vendors. By using this process, cost and time of the software product was reduced. But in testing phase there aroused many difficulties for a tester because the tester had a very limited access to the source code of the reusable component of the product. The component meta-data was used to attach additional information with the components to facilitate testing. Black-box testing was used where code of the component was not available. Usually, a component had a hidden interface and a tester was not able to input the values in it unless its interface was not completed. In this paper, the issues in component based testing using meta-data approach for black-box testing was discussed when component’s interface not available. It also presented the methodology how meta-data was used in for black-box testing. It had also proposed an application (tool) which was used in automated for black-box testing of a Dynamic Link Library (DLL) component.

In the year 2010, Shashank et. al. [35] presented a paper which stated that Component Based Software Engineering (CBSE) have emerged as an approach that offer rapid development of system using fewer resources and effort. The core idea of reuse and cutting down the development costs is achieved if the components offer reliable services. Thus, integration components and testing became an important phases in CBSE. Integration of components is a very important activity. This involves understanding, communication and coordination between the components. Developers are not provided with sufficient information on these components. As a result of this, understanding data flow while integrating these components is a challenge. Component based software facilitates development of complex systems by allowing integration of reusable components. Testing components is a challenging area of research. There had been troubles integrating the components. This had in turn affected the quality and reliability of the software. This research aimed at finding the existing integration testing and challenges in CBSE.

In the year 2010, Binbin et. al. [36] proposed a paper which stated that when a component is integrated into Component-Based System (CBS), it creates the dependency relationship with other components. This paper presented an approach of test sequence generation for integration testing of component software based on these dependencies. Definitions of Component Dependency Graph (CDG) and component dependability were given. According to these definitions, the concept of Component Dependency Matrix (CDM) was presented to describe the dependency relations between components in CBS. It developed test coverage criteria based on dependency for component-based integration testing. In terms of the coverage criteria, it proposed an algorithm of test sequence generation for integration testing of CBS according to CDM. The experiment results had shown that the test sequences generated by this approach reflected the dependency relationship better than other methods.

In the year 2010, Bin et. al. [37] presented a paper which stated that the Component Based Development allow one to build software from existing components and promise to improve software reuse and reduced costs. For critical applications, the user of a component ensures that it fits the requirements of the application. To achieve this, testing is a well-suited means when the source code of the application is not available. Robustness testing is a testing methodology to detect the vulnerabilities of a component under unexpected inputs or in a stressful environment. As components failed differently in different states, the author used a state machine based approach...
to robustness testing. Initially a set of paths was generated to cover transitions of the state machine, and it was used by the test cases to bring the component into a specific control state. Then methods called with invalid inputs were fed to the component in different stated to test the robustness. By traversing the paths, the test cases covered more states and transitions compared to stateless API testing. This approach was applied to several components, including open source software and compared the results with existing approach.

In the year 2011, Shang et. al. [38] presented a paper which stated that Component Based Software Development is an effective and efficient approach to improve productivity and quality of software development. So far, almost all the technologies of software components focused on component description and operation. But the software architecture which took the component as the basic unit had a great progress and provided a software development process of the top-down by describing its structure and features. In order to improve the efficiency and quality of Management Information System (MIS), the reusable component technology had been introduced in MIS. Through the description of the structure and features, the reusable components were used in the structure and features of MIS in the development process such as demand analysis, system design, system implementation, system testing and system maintenance. The result had shown that supporting and implementing the assembly of MIS under the platform of component running had improved the stability, quality and development efficiency of the existing MIS.

In the year 2011, Mohanty et. al. [39] presented a paper which stated that regression testing is the process of testing a modified system using the old test suite. As the test suite size is large, system retesting consumes large amount of time and computing resources. This issue of retesting of software systems is handled using a good test case prioritization technique. A prioritization technique schedules the test cases for execution so that the test cases with higher priority are executed before lower priority. The objective of test case prioritization is to detect fault as early as possible so that the debuggers can begin their work earlier. This paper proposed a new prioritization technique to prioritize the test cases to perform regression testing for Component Based Software System (CBSS). The components and the state changes for a component based software system were represented by UML state chart diagrams which were then converted into Component Interaction Graph (CIG) to describe the inter-relation among components. The prioritization algorithm took this CIG as input along with the old test cases and generated a prioritized test suite taking into account total number of state changes and total number of database access, both direct and indirect, encountered due to each test case. This algorithm was found to be very effective in maximizing the objective function and minimizing the cost of system retesting when applied to few Java projects.

In the year 2011, Tao et. al. [40] presented a paper which stated that a Component-based software system consists of various components, such as third-party components and in-house built components. Component changes frequently occur in software maintenance, which refer to regression testing. When changes are made to a component, the component is affected and the changes bring impacts on the entire system. Related existing research not addresses the issue of systematic regression testing of component-based software, especially at system level. This paper proposed a systematic regression testing method from components to system, including analysis of change, impact and test suite refreshment. The paper also reported a case study based on a realistic component-based software system using a state-based testing practice, which had shown that the approach was feasible and effective.

In the year 2011, Cibulski et. al. [41] presented a paper in which it was stated that Test-Driven Development (TDD) is characterized by repeated execution of a test suite that enables developers to change code with confidence. However, running an entire test suite after every small code change is not always cost effective. Therefore, Regression Test Selection (RTS) techniques play vital role for TDD. The most challenging thing for TDD is the task of selecting a small subset of tests that are most likely to detect a regression fault in a given small and localized code change. The authors presented cost-bounded RTS techniques based on both dynamic program analysis and natural-language analysis. The authors implemented this techniques in a tool called Test Rank, and evaluated its effectiveness on two open-source projects. It was shown that using these techniques, developers
could accelerate their development cycle, while maintaining a high bug detection rate, whether actually following TDD, or in any methodology that combines testing during development.

In the year 2011, Hsu et al. [42] proposed a paper which stated the fact that with the growing size and complexity of software applications, traditional software reliability methods are insufficient to analyze inter-component interactions of modular software systems. The numbers of test cases are extremely large for this application; therefore, it is hard to extensively test each software component given resource limitations. This paper proposed an adaptive framework of incorporating path testing into reliability estimation for modular software systems. Three estimated methods based on common program structures, namely, sequence, branch, and loop structures, were proposed to calculate the path reliability.

3. Conclusion

Today, most of the software systems are developed by using the existing code or the available components. This concept of using existing code is called Reusability. Reusability is achieved by performing some interfacing between different software components. The software reusability is presented either in terms of some code or in terms of component objects.

This paper presents a new approach for testing the component-based software systems. The studies show that testing of component-based software systems is necessary yet expensive. The testing of component is required for checking the reliability of complete system. Finite Automata-based testing is an easy way to test an application. The testing is based on the requirement of the system. If the system requirements are not met, then it shows failure otherwise success. The testing technique is quite simple, effective, and less complex and it does thorough testing of an application, but the process is time consuming because every single path (change of state) need to be tested solely which increases its time complexity.

4. Future Scope

In future, the testing technique can be improvised to reduce the time complexity. It can be done by testing integrated paths (group of paths) instead of testing every single path, which might prove little complex but less time consuming.

References

A Literature Review on Component Testing


