ISSN 2278-3091
Volume 8, No.2, March - April 2019
International Journal of Advanced Trends in Computer Science and Engineering
Available Online at http://www.warse.org/IJATCSE/static/pdf/file/ijatcse31822019.pdf
https://doi.org/10.30534/ijatcse/2019/31822019

Reliability of Component-Based Systems – A Review

Shivani Yadav[1], Bal Kishan[2]
Department of Computer Science and Applications, Maharshi Dayanand University, Rohtak-124001, India
[1]shivaniyadav17@gmail.com
[2]balkishan248@gmail.com

ABSTRACT

Component-based software development is a methodology to use and create software systems that consume and produce reusable software components. The components act as independent units that interact to form a functional system. This approach has the potential to overcome the overheads associated with monolithic software applications. The reliability of such a system is highly dynamic in real-time. Reliability of the software ensures that the program performs the specified function effectively and results in quality software. Through this paper an introduction of component based system with its reliability, performance factors and its advantages are provided. This paper also provides the related work carried out over the years by various researchers and the research gaps.

Key words: Reliability, software components, performance, component, reusability.

1. INTRODUCTION

Software systems have become such an important part of our day to day lives that it's easy to overlook their presence. They're present across every spectrum of our life, be it communication, transportation or something as simple as doing household chores with the help of home appliances. This seamless integration requires that the software be robust, reliable and functional at all times. The ability to conform to the required standards and perform the desired functions, determines the software quality. High software quality ensures reliability. The probability of operational performance without failure of a system within specified period of time is known as software reliability. As the proliferation of software continues, it becomes increasingly difficult to maintain software quality and reliability due to the extensive time and energy required. This calls for component based software engineering (CBSE), which also helps in reducing the development cost and effort required by reusing the components. Our next section introduces component-based system, its performance factors and reliability; Section III presents related work carried out over years by various researchers; Section IV presents the research gaps and at last Section V presents the conclusion and future work.

2. COMPONENT-BASED SYSTEM

CBSE is a branch of software engineering that focuses on separating concerns about extended facilities available in a particular software system. It relies on reusing independent components to define, implement and compose new composite systems. [1]
Components can create or consume incidents, and be used in event-driven structures. The component model works by identifying the elements in an interface and their definition.

1.1 Performance factors and advantages of component based systems

Performance of component-based system depends on the following factors [2]:

- **Component implementation:** A single component can be developed and deployed in multiple ways depending on the developer's implementation of the interface. While different components can result in the same functionality, the resources consumed by them, their execution times and dependencies on other components can vary.
- **Required services:** A component might require calls to different components for invoking services. The performance of the component then becomes contingent on the performance of the other components or services being called.
- **Deployment platform:** A software architect might deploy a component on a number of platforms as per requirements. This deployment platform can be a combination of several software and hardware platforms. Each of these combinations results in varied performance levels.
- **Usage profile:** The different input parameters to a component service determine the performance of a component. The execution times can changes the inputs from calls to required services are returned with results.
- **Resource contention**: Multiple components fight for resources as the program runs through different components. This results in processes waiting for limited resources, adding up to the execution times

The following advantages of the component-based system are [3]:

- **Reusability**: The primary motive of component-based development approach is to use already existing components and hence reduce the effort and cost. The required components are assembled to design a new application. Higher frequency of reuse ensures that the components are thoroughly checked for quality and increase the productivity of the system.

- **Maintainability:** In a component-based system, the software becomes easier to maintain as the functionality needs to implemented only once and can be updated easily by making changes at a single source.
- **Portability:** The ease with which a component can be transferred from one environment to another, for reuse, is defined as portability. Portability results in the higher frequency of component reuse.
- **Quality Enhancement:** Quality of components is improved as components are tested many times for reusability.
- **Low cost and time:** One component is used many times at many places which reduces production cost as well as time.
- Consistency increases
- Manages complexity in an effective manner or reduces complexity up to a certain level

2.1 Reliability of component-based systems

Reliability analysis approaches are broadly classified into three main models [4]:

1. State-based models

   This model takes into account the change of state as the control flows from one component to another. The assumption is that failure occurs due to individual components. Each component's behavior is taken to be independent of past behavior, also known as Markov behavior. Architectural models were proposed using Discrete Time Markov chain (DTMC) or semi-Markov processes. It can be represented in two ways:

   Hierarchical models: This model starts with the architecture model at the core and then including cover failure behavior to solve the architecture model and arrive at reliability.

   Composite model: The reliability of the application is predicted using a combination of software architecture and failure behavior into a composite model.

   State-based models analyses the reliability using various methods like mathematical formulas, algebraic method, algorithms and also, markov theory which includes discrete time markov model (DTMC), time continuous markov model (CTMC), semi markov model (SMP)

2. Path-based models

   An application can have multiple execution paths based on the input parameters and subsequent service requests. This model considers the different paths obtained by testing the components.
   Reliability of each path = product of component reliabilities along that path
   System reliability = average of path reliabilities over all paths of the software system

   Path-based model attains a series of the entire probable executable paths which are obtained by experiments, algorithms, simulation for evaluating system reliability.

3. Additive models

System reliability is calculated by focusing on the individual components failure data and mainly at the time of testing. In additive model, reliability is modeled using non-homogeneous poisson process (NHPP).

3. LITERATURE REVIEW

Some of the following approaches give information about the performance evaluation methods used for analysing component-based software systems:

Woodside et al. [5] came up with the concept of performance evaluation elements to a model of software systems. Software developers and designers create and store model elements like, model annotations, deployment models, infrastructure models, design refinements and communication models, to extend platform-independent models to platform-specific models.

A technique was proposed by Eskanazi et al. [6] to improve APPEAR to predict the performance of component-based software systems. The method involves simulating the full application model, keeping in mind the implementation of component services, predicting the performance of individual component processes, and creating flow charts for annotated control of component t services. However, it is difficult to repeat this approach because it does not include a model or tool to apply to other software systems.

Menasce et al. [7] suggested a method for negotiating performance properties at runtime with their clients by using QoS-aware software components. Using previously monitored performance properties, the components build a QN model which is then used to analyse their ability to fulfil the specified OoS service requirement.

The concept of parametric performance for software components was developed by Reussner et al. [8]. The performance of each component can be calculated if the performance of necessary services for the execution of that component is known. This design helps in determining the components to provide appropriate performance characteristics for the deployment environment at the design time.

Ostrand et al. [9] used negative binomial regression, with file size, programming language, status and the number of changes on file as metrics, to predict fault-prone modules. According to the accuracy parameter, a negative binomial regression model was found to be extremely useful.

Yang et al. [10] applied Self-Adaptation Learning Control Network-(FALCON)to a set of artificial data. FALCON, a specific type of neural network. The model enables measurement of many quality properties such as reliability, performance, and stability, the complexity of inheritance

trees, reuse and depth as input. However, research is still in its infancy because they have not used this approach with the actual data set.

Seliya et al. [11] used Expectation–Maximization (EM) technique on limited fault data to predict software faults.

Data sets used for training were JM1 data sets, whereas data sets used for testing were KC1, KC2 and KC3. Since, sufficient data was not available, an exact model could not be created and a model for only limited problem of fault data could be built. In the absence of fault data for components or the cost of operating the metrics collection can be very expensive. This necessitates the need for powerful classifiers to build accurate classification models which can use both unlabelled and labelled data.

The team evaluated the performance considering type I, type 2 and overall misclassification rate parameters. Then a prediction model was created after labelling the unlabelled data with EM technology and using all modules. Quality models of emerging markets presented acceptable results for the problem of semi-supervised fault prediction.

Koru et al. [12] used Mozilla open source projects to run their tree based models for identifying change-prone classes by classifying them under different labels using metrics. It was observed that 20% of classes changed.

Tomaszewski et al. [13] relied on a combination of expert judgement and univariate linear regression for predicting software faults. They used method and class metrics to study a couple of software systems developed in Ericsson. The accuracy of fault prediction was the parameter for performance evaluation. Eleven experts were invited to anticipate the fault-proneness of the components. It was observed that expert opinions were less accurate than statistical approaches and it became increasingly difficult for experts to anticipate faults in large data-sets. Even amongst the statistical approaches, class metrics based evaluation models were found to be more accurate than component metrics based evaluation models. The identification of fault-prone components by experts was quite subjective and varied for the most part and in cases where experts agreed on same components, the fault density values were often diverse.

Olague et al. [14] studied six versions of Rhino project using three software metrics suites to predict fault-proneness of object-oriented classes. The analysis was done using two techniques: univariate and multivariate binary logistic regression. Three metrics suite were used: Bansiya and Davis's quality metrics (QMOOD), Abreu's object-oriented metrics (MOOD) and Chidamber–Kemerer (CK) metrics suite. Model validation was done using accuracy as the parameter for model validation and the metrics' effects were examined using Spearman correlation. The best performance for fault prediction was observed with CK metrics suite, in particular the Weighted Methods Count and Response for Children metrics. Univariate binary logistics regression suggested that CK QMOOD metrics, CIS and NOM, were also useful. Multivariate binary logistic regression models were quite useful for iterative and agile software development processes.

Bibi et al. [15] applied Regression via Classification (RvC) to predict faults and estimated the number of software faults with a confidence interval. Results provided better regression error than the standard regression methods. The dataset used for analysis was Pekka dataset, provided by ISBSG (International Software Benchmarking Standards Group), and a Finland based commercial bank. Mean absolute error was used as the performance evaluation metric.

Turhan et al. [16] showed that independence assumption is not detrimental, by using PD, PF and balance parameters with principal component analysis (PCA) pre-processing when using Naive Bayes algorithm.

Chang et al. [17] discovered fault patterns using association rules and proposed a fault prediction model. The results were promising and the method can be used in causal analysis.

Tosun et al. [18] used three embedded software projects and conducted experiments on public datasets to validate Zimmermann and Nagappan's paper published. They found network measures to be effective indicators of fault-prone modules for large systems. PD, PF, and precision were used as the evaluation metrics.

Arisholma et al. [19] evaluated a legacy system project, based on Java, to determine fault-proneness using different models. Their findings showed that prediction accuracy was affected by the modeling technique in a limited way. Process metrics are a good measure for fault prediction and the best model can be determined using performance evaluation parameters. Further, they included cost-effectiveness as a measure for assessment of models. The best results were obtained when Adaboost was combined with C4.5 and evaluation techniques were run with default parameters.

Vikrant Kaulgud et al. [20] designed a metric that can be used for forecasting and early warnings, using data from component testing tools to derive in-process insights. Prioritization of component tests ensured that complex components received effort allocation in line with requirements. At the same time, exhaustive testing of the entire component set was ensured.

Nasib Singh Gill et al. [21] formulated a systematic approach for constructing testable components to increase test automation and convert a given component into a testable component. The approach helped in improving the overall quality of the system by making fault detection in the early phases of software development.

M. K. Pawar et al. [22] analyzed different integration testing approaches such as UML based, CIG (Component Interaction Graph), Component certification, Component Interaction testing, component metadata etc. and presented various issues associated with them.

Ahmad Nauman Ghazi et al. [23] compared three testing technologies for testing heterogeneous systems namely: Exploratory manual testing, combinatorial testing and search based testing. Exploratory manual testing was found to be the most frequently used technique for heterogeneous systems and search based techniques are least frequently used. This can be attributed to the fact that search based technique is a relatively new area and needs further research to utilize its full potential.

Prasenjit Banerjee et al. [24] proposed a quality evaluation framework, and a set of metrics, along with their theoretical validations for the conceptual level component based system model. The framework included parameters such as completeness, complexity, expressiveness and analysability for quality measurement. The software quality measurements were used to exhibit two-fold quality evaluation viewpoints: designer level and user level. The proposed framework was tested on the library management system.

4. RESEARCH GAPS

So far our literature survey shows that the authors are interested in model-based testing of component-based software. Soft computing paradigms which are based on meta-heuristic are very less exploited in the field of CBD. The combination of these two approaches will enhance the effectiveness of testing. Table 1 below enlists some important research publications in the field of CBD and testing along with their research gaps.

Table 1: Important Publications and Research Gaps

Publication Year	Title of Paper	Authors	Research Gaps/ Limitations
2018	"The Impact of Motivator and Demotivator Factors on Agile Software Development" [25]	Shahbaz Ahmed Khan Ghayyur, Salman Ahmed, Saeed Ullah, Waqar Ahmed	Needs to find motivator factors according to core agile practices
2017	"Professionals are not Superman: Failures beyond Motivation in Software Experiments" [26]	Oscar Dieste, Efra'in R. Fonseca C., Geovanny Raura, Priscila Rodriguez	Fails in casual relationship detection and empirical studies.
2017	"Motivators and Demotivators of Agile Software Development: Elicitation and Analysis" [27]	Shahbaz Ahmed Khan Ghayyur, Salman Ahmed, Adnan Naseem, Abdul Razzaq	Lacks in empirical analysis of motivators and demotivators
2016	"Deriving UML-based Specifications of Inter-Component Interactions from Runtime Tests" [28]	Thorsten Haendler, Stefan Sobernig, Mark Strembeck	Computationally expensive and requires runtime mapping
2016	"COSTOTest: A Tool for Building and Running Test Harness for Service-Based Component Models" [29]	Pascal André, Jean-Marie Mottu, Gerson Sunyé	Lacks generality and applicable to service oriented components only
2016	"Search-Based Cost-Effective Test Case Selection within a Time Budget: An Empirical Study" [30]	Dipesh Pradhan, Shuai Wang, Shaukat Ali	Can be improved by hybridizing with local search techniques.
2015	"A Generic Model-Based Methodology of Testing Techniques to Obtain High Quality Software" [31]	Khaled Almakadmeh, Fatima Abu-Zitoon	Not applied on component based softwares and soft computing techniques are not utilized.
2015	"Neuro-Fuzzy Model to Estimate & Optimize Quality and Performance of Component Based Software Engineering" [32]	Gaurav Kumar, Pradeep Kumar Bhatia	Lacks real life data and can be optimized for more number of quality attributes
2014	"A Novel Approach to Component-Based Software Testing" [33]	Lata Nautiyal, Dr. Neena Gupta, Dr. Sushil Chandra Dimri	Only theoretical basis, No implementation
2014	"Design Test Process in Component-Based Software Engineering: An Analysis of Requirements Scalability" [34]	Mariem Haoues, Asma Sellami, Hanêne Ben-Abdallah	Can be improved using soft computing along with model based techniques.

2013	"A Survey on Software Testing Techniques using Genetic Algorithm" [35]	Chayanika Sharma, Sangeeta Sabharwal, Ritu Sibal	GA can be hybridized with other soft computing techniques.
2012	"State-Model-Based Regression Test Reduction for Component-Based Software" [36]	Tamal Sen, Rajib Mall	Code should be auto-generated and applicable to regression testing
2012	"Automatic Generation of Test Models and Properties from UML Models with OCL Constraints" [37]	Miguel A. Francisco, Laura M. Castro	Requires tool support and test data repository. Not applied on integration testing.
2011	"Testing Component Based Software: What It has to do with Design and Component Selection" [38]	Shyam S. Pandeya, Anil K. Tripathi	Applicable on component selection level.
2011	"Towards Incremental Component Compatibility Testing" [39]	Ilchul Yoon, Alan Sussman, Atif Memon, Adam Porter	Needs knowledge of component dependencies
2010	"Automated Unit and Integration Testing for Component-based Software Systems" [40]	F. Saglietti, F. Pinte	Applicable to limited system size

Another method for improving the reliability of the system where only prediction is to be determined is using Reliability Block Diagram (RBD). It measures the important components of the system and the changes required [41]. For reducing failures in any software, one can also use component software engineering with maturity driven process. [42]

5. CONCLUSION AND FUTURE SCOPE

The aim of component-based software development process is to design and construct software systems using reusable components. Components are designed to produce or consume events and can be used for implementing event-driven architectures. The component model works by identifying the elements in an interface and their definition. Combination of model-based testing of component-based software and soft computing paradigms which are based on meta-heuristic are discussed in the field of CBD and the combination of these two approaches will enhance the effectiveness of testing. This paper provides an overview of component based software system, its performance factors and advantages. We have summarized some existing techniques, and after extensive literature survey, we realized that there are many research gaps which need improvement.

REFERENCES

[1] R. Alam, and M. U. Bokhari. **Assuring reliability of the software using component based software engineering,** *Journal of Basic and Applied Engineering Research*, vol. 1(7), pp. 55-60, 2014.

[2] S. Becker, and R. H. Reussner. **The impact of software component adaptation on quality of service properties**, *L'objet*, vol. 12 (1), pp. 105-125, 2006.
https://doi.org/10.3166/objet.12.1.105-125

[3] S. Becker, H. Koziolek, and R. Reussner. **The Palladio component model for model-driven performance prediction**, *Journal of Systems and Software*, vol. 82(1), pp. 3-22, 2009.
https://doi.org/10.1016/j.jss.2008.03.066

[4] K. Tyagi, and A. Sharma. **Reliability of component based systems – A critical survey**, *WSEAS Trans. on Computer*, vol. 2(11), pp. 45–54, 2012.

[5] M. Woodside, D. Petriu, and K. Siddiqui. **Performance-related completions for software specifications**, in *Proc. ICSE'02*, pp. 22-32, 2002.
https://doi.org/10.1145/581344.581346

[6] E. Eskenazi, A. Fioukov, and D. Hammer. **Performance prediction for component compositions**, in *Proc. CBSE'04*, in: LNCS, vol. (3054), pp. 28-293, 2004.
https://doi.org/10.1007/978-3-540-24774-6_25

[7] D. A. Menasce, V. A. F. Almeida, and L. W. Dowdy. ***Performance by Design***, Prentice Hall, 2004.

[8] R. H. Reussner, S. Becker, and V. Firus. **Component composition with parametric contracts**, in *Tagungsband der Net.Object Days*, pp. 155-169, 2004.

[9] T. J. Ostrand, E. J. Weyuker, and R. M. Bell. **Automating algorithms for the identification of fault-prone files**, in *Proc. ISSTA '07*, pp. 219–227, 2007.
https://doi.org/10.1145/1273463.1273493

[10] B. Yang, L. Yao, and H. Huang. **Early software quality prediction based on a fuzzy neural network model**, in Proc. *ICNC' 07, IEEE*, 2007, pp. 760–764.
https://doi.org/10.1109/ICNC.2007.347

[11] N. Seliya, and T. M. Khoshgoftaar. **Software quality estimation with limited fault data: A semi supervised learning perspective**, *Software Quality Journal*, vol. 15(3), pp. 327–344, 2007.
https://doi.org/10.1007/s11219-007-9013-8

[12] A. G. Koru, and H. Liu. **Identifying and characterizing change-prone classes in two large-scale open-source products**, *Journal of Systems and Software*, vol. 80(1), pp. 63–73, 2007.
https://doi.org/10.1016/j.jss.2006.05.017

[13] P. Tomaszewski, J. Håkansson, H. Grahn, and L. Lundberg. **Statistical models vs. expert estimation for fault prediction in modified code – An industrial case study**, *Journal of Systems and Software*, vol. 80(8), pp.1227–1238, 2007.
https://doi.org/10.1016/j.jss.2006.12.548

[14] H. M. Olague, L. H. Etzkorn, S. Gholston, and S. Quattlebaum. **Empirical validation of three software**

metrics suites to predict fault-proneness of object-oriented classes developed using highly iterative or agile software development processes, *IEEE Transactions on Software Engineering*, vol. 33(6), pp. 402–419, 2007.
https://doi.org/10.1109/TSE.2007.1015
[15] S. Bibi, G. Tsoumakas, I. Stamelos, and I. Vlahavas. **Regression via classification applied on software defect estimation**, *Expert Systems with Applicationl*, vol. 34(3), pp. 2091–2101, 2008.
https://doi.org/10.1016/j.eswa.2007.02.012
[16] B. Turhan, and A. Bener. **Analysis of Naive Bayes' assumptions on software fault data: An empirical study**, *Data & Knowledge Engineering*, vol. 68(2), pp. 278–290, 2009.
https://doi.org/10.1016/j.datak.2008.10.005
[17] C. Chang, C. Chu, and Y. Yeh. **Integrating in-process software defect prediction with association mining to discover defect pattern**, *Information and Software Technology*, vol. 51(2), pp. 375–384, 2009.
https://doi.org/10.1016/j.infsof.2008.04.008
[18] A. Tosun, B. Turhan, and A. Bener. **Validation of network measures as indicators of defective modules in software systems**, in *Proc PROMISE '09*, ISBN: 978-1-60558-634-2, pp. 1–9, 2009,
https://doi.org/10.1145/1540438.1540446
[19] E. Arisholma, L. C. Briand, and E. B. Johannessen. **A systematic and comprehensive investigation of methods to build and evaluate fault prediction models**, *Journal of Systems and Software*, vol. 83(1), pp. 2–17, 2010.
https://doi.org/10.1016/j.jss.2009.06.055
[20] V. Kaulgud, and V. S. Sharma. **Insights into component testing process**, in *Proc. WETSoM '11*, pp. 27-30, 2011.
https://doi.org/10.1145/1985374.1985382
[21] N. S. Gill, and P. Tomar. **New and innovative process to construct testable component with systematic approach**, *ACM SIGSOFT Software Engineering Notes*, vol. 36(1), pp. 1-4, 2011.
https://doi.org/10.1145/1921532.1921540
[22] M. K. Pawar, R. Patel, and N.S Chaudhari. **Survey of integrating testing for component-based system,** *International Journal of Computer Applications*, vol. 57(18), pp. 21-25, 2012.
[23] A. N. Ghazi, K. Petersen, and J. Börstler. **Heterogeneous Systems Testing Techniques: An Exploratory Survey**, *International Conference on Software Quality, Software Quality Days (SWQD), Part of the Lecture Notes in Business Information Processing book series*, 2015, pp. 67-85.
https://doi.org/10.1007/978-3-319-13251-8_5
[24] P. Banerjee, and A. Sarkar. **Quality evaluation framework for component based software**, in *Proc. ICTCS 16*, 2016 Article no. 17.
https://doi.org/10.1145/2905055.2905223
[25] S. A. K. Ghayyur, S. Ahmed and S. Ullah. **The impact of motivator and demotivator factors on agile software development,** *International Journal of Advanced Computer Science and Applications*, vol. 9(7), pp. 80-93, 2018.
https://doi.org/10.14569/IJACSA.2018.090712
[26] O. Dieste, E. R. Fonseca C.G. Raura, and P. Rodriguez. **Professionals are not superman: failures beyond motivation in software experiments**, in *IEEE/ACM 5th International Workshop on Conducting Empirical Studies in Industry (CESI)*, 2017, pp. 27-32.
https://doi.org/10.1109/CESI.2017.8
[27] S. A. K. Ghayyur, S. Ahmed, A. Naseem, and A. Razzaq. **Motivators and demotivators of agile software development: elicitation and analysis**, *International Journal of Advanced Computer Science and Applications*, vol. 8(12), pp. 304-314, 2017.
https://doi.org/10.14569/IJACSA.2017.081239
[28] T. Haendler, S. Sobernig, and M. Strembeck. **Deriving uml-based specifications of inter-component interactions from runtime tests**, in *Proc. SAC'16*, 2016, pp. 1573-1575.
https://doi.org/10.1145/2851613.2851981
[29] P. André, J. Mottu, and G. Sunyé. **Costotest: a tool for building and running test harness for service-based component models (demo)**, in *Proc. ISSTA'16*, 2016, *ACM*, pp. 437-440.
https://doi.org/10.1145/2931037.2948704
[30] D. Pradhan, S. Wang, S. Ali, and T. Yue. **Search-based cost-effective test case selection within a time budget: an empirical study**, in *Proc. GECCO'16*, 2016, *Association for Computing Machinery(ACM)*, pp. 1085-1092.
https://doi.org/10.1145/2908812.2908850
[31] K. Almakadmeh, and F. Abu-Zitoon. **A generic model-based methodology of testing techniques to obtain high quality software**, In *Proc. IPAC'15*, 2015, ACM, Article no. 49.
https://doi.org/10.1145/2816839.2816903
[32] G. Kumar, and P. K. Bhatia. **Neuro-fuzzy model to estimate & optimize quality and performance of component based software engineering**, *Newsletter ACM SIGSOFT Software Engineering Notes*, vol. 40(2), pp. 1-6, 2015.
https://doi.org/10.1145/2735399.2735410
[33] L. Nautiyal, N. Gupta, and S. Chandra Dimri. **A novel approach to component-based software testing**, *Newsletter ACM SIGSOFT Software Engineering Notes*, vol. 39(6), pp. 1-4, 2014.
https://doi.org/10.1145/2674632.2674640
[34] M. Haoues, A. Sellami, and H. Ben-Abdallah. **Design test process in component-based software engineering: an analysis of requirements scalability**, in *Proc. WETSoM'14*, 2014, pp. 48-54.
https://doi.org/10.1145/2593868.2593877
[35] C. Sharma, S. Sabharwal, and R. Sibal. **A survey on software testing techniques using genetic algorithm**, *International Journal of Computer Science Issues*, vol. 10(1), pp. 381-393, 2013.
[36] T. Sen, and R. Mall. **State-model-based regression test reduction for component-based software**, *International Scholarly Research Network, ISRN Software Engineering*, pp. 1-9, 2012.
https://doi.org/10.5402/2012/561502
[37] M. A. Francisco, and L. M. Castro. **Automatic generation of test models and properties from uml models with ocl constraints**, *in Proc OCL*'12, 2012, pp. 49-54.
https://doi.org/10.1145/2428516.2428525
[38] S. S. Pandeya, and A. K. Tripathi. B**esting component-based software: what it has to do with design and component selection**, *Journal of Software Engineering and Applications*, vol. 4(1), pp. 37-47, 2011.
https://doi.org/10.4236/jsea.2011.41005

[39] I. Yoon, A. Sussman, A. Memon, and A. Porter. **Towards incremental component compatibility testing**, in *Proc. CBSE'11*, 2011, pp. 119-128. https://doi.org/10.1145/2000229.2000247

[40] F. Saglietti, and F. Pinte. **Automated unit and integration testing for component-based software systems**, in *Proc. S&D4RCES'10*, 2010, Article no. 5. https://doi.org/10.1145/1868433.1868440

[41] D. Iudean, A. Cretu, R. Munteanu, R. Moga, N. Stroia, D. Moga, and L. Vladareanu, **Reliability Approach of a Compressor System using Reliability Block Diagrams**, *International Journal of Advanced Trends in Computer Science and Engineering*, vol. 8, no. 1.1, pp. 133-138,2019.

[42] A. Shaikh, **The Role of Maturity Driven Software Process Improvement in an Industry**, *International Journal of Advanced Trends in Computer Science and Engineering*, vol. 8, no. 1.1, pp. 344-350, 2019. https://doi.org/10.30534/ijatcse/2019/6081.12019