Asian Journal of Information Technology 15 (20): 4063-4070, 2016
ISSN: 1682-3915

Estimation of Software Quality based on Software Metrics Using Modified Cuckoo Search Optimization Algorithm

[1]V. Suganthi and [2]S. Duraisamy
[1]Department of MCA, Nehru Institute of Information Technology and Management, Coimbatore, India
[2]Department of Computer Applications, Sri Krishna College of Engineering and Technology, Coimbatore, India

Abstract: Software metrics is considered as the major criterion as it directly depends on the software quality. The intensity with which developed software remains efficient with its purpose, determines its quality. In recent years, the software development with the major consideration being its quality has become an important research area. The increased use of software system in various technologies has led to vast development in the field of delivering efficient software with higher quality and reliability. In order to develop better quality software, we have considered software metrics like popularity, reliability and computational cost. Initially the test cases are generated and then the software metrics are calculated as features. Those features are optimized using Modified Cuckoo Search algorithm (MCS). The optimization is done in order to obtain completely relevant software metrics that suits our development process. Finally, the optimized result will be evaluated by software quality measure. The proposed method uses the reliability for the quality measure. The reliability can help us in verifying the quality of the particular software that has been developed.

Key words: Software metrics, software quality, reliability, cost, popularity, modified cuckoo search

INTRODUCTION

Modern software systems become more and more large-scale, complex and uneasily controlled, resulting in high development cost, low productivity, unmanageable software quality and high risk move to new technology. Consequently, there is a growing demand of searching for a new, efficient and cost-effective software development paradigm (Cai *et al*., 2000). At the same time, estimating the quality of a software product during its development or pre-operation phases is a challenging problem, especially when there is very limited or no information regarding its existing quality is available (Chaudron *et al*., 2001). In software quality estimation problems, a software quality classification or software fault prediction model is typically trained or built using software measurements and fault data from a previous system release or similar software project developed by the given organization. These models are supervised learning in the sense that the training process is guided by the software quality measurement (i.e., the dependent variable or the fault-proneness label). The trained model is then used to predict the quality of the software modules in a software project under consideration. In this study, we are particularly interested in learning software quality estimation models using unsupervised learning methods in the absence of software quality measurements.

The component-based approach is the most recent approach and will probably mature over the years of the millennium. Component-Based Software Engineering (CBSE) has emerged as a technology for rapid assembly of flexible software systems. It combines the elements of software architecture, modular software design, software verification, configuration and development (Chaitanya and Ramesh, 2011; Sirobi and Parashar, 2013). CBSE denotes the disciplined practice of building software from pre-existing smaller products, generally called software components, in particular when this is done using standard or de-facto standard component models (Sagredo *et al*., 2010). The popularity of such models has increased greatly in the last decade, particularly in the development of desktop and server-side software, where the main expected benefits of CBSE are increased productivity and timeliness of software development projects (Luders *et al*., 2005; Mikaelian *et al*., 2005; Ratnaweera *et al*., 2004). The dependence on reliability

Corresponding Author: V. Suganthi, Department of MCA, Nehru Institute of Information Technology and Management, Coimbatore, India

and quality of mission-critical and high-assurance software systems are very essential. Unfortunately, high software reliability often involves prohibitive consumption of time and monetary resources for software development. Reliability improvement techniques may include more rigorous design and code reviews, more exhaustive testing and focused re-engineering of high-risk segments of a software system (Khan *et al*., 2014). Due to the fact that quality improvement processes are so time and resource consuming, cost-effective strategies are warranted and hence have been the target for the software reliability improvement community (Khoshgoftaar and Seliya, 2003; Mikaelian *et al*., 2005).

The evolution method used in the present study is Particle Swarm Optimizer (PSO). However, PSO has evolved in Data Mining (DM) in which it can reduce complexity and speed up the data mining process (Shukran *et al*., 2011; Mikaelian *et al*., 2005). Although, PSO shares many similarities with evolutionary computation techniques, the standard PSO does not use evolution operators such as crossover and mutation. PSO emulates the swarm behaviour of insects, animals herding, birds flocking and fish schooling where these swarms search for food in a collaborative manner. Each member in the swarm adapts its search patterns by learning from its own experience and other members' experiences (Kadirkamanathan *et al*., 2006). These phenomena are studied and mathematical models are constructed. In PSO, a member in the swarm, called a particle, represents a potential solution which is a point in the search space. The global optimum is regarded as the location of food. Each particle has a fitness value and a velocity to adjust its flying direction according to the best experiences of the swarm to search for the global optimum in the D-dimensional solution space (Joseph *et al*., 2011; Liang *et al*., 2016).

Literature review: Abdellatief *et al*. (2013) have proposed a systematic mapping study of several metrics measure the quality of CBSS and its components. The research had found 17 proposals that could be applied to evaluate CBSSs while 14 proposals could be applied to evaSSSSluate individual components in isolation. Various elements of the software components that were measured had been reviewed and discussed. In the present research the quality assessment of the primary studies had detected many limitations and had suggested guidelines for possibilities for improving and increasing the acceptance of metrics.

Yacoub *et al*. (2004) have proposed a reliability model and a reliability analysis technique for component-based software. The technique had been named as Scenario-Based Reliability Analysis (SBRA). In this research, researcher had used a scenario of component interactions to construct a probabilistic model named Component-Dependency Graph (CDG). Based on CDG, a reliability analysis algorithm was developed to analyze the reliability of the system as a function of the reliabilities of its architectural constituents. An extension of the proposed model and algorithm was also developed for distributed software systems.

Huang *et al*. (2006) have proposed a novel approach to recovering software architecture from component based systems at runtime and changing the runtime systems via manipulating the recovered software architecture. The recovered software architecture can accurately and thoroughly describe the actual states and behaviours of the runtime system. The study had also presented that, the recovered software architecture can be represented as multiple views so as to help different users to control the complexity from different concerns. Based on the reflective ability of the component framework, the recovered software architecture was up-to-date at any time and changes made on it will immediately lead to the corresponding changes in the runtime system. The proposed method presented in this study was demonstrated on PKUAS, a reflective J2EE (Java 2 Platform Enterprise Edition) application server and the performance was also evaluated.

Rathfelder *et al*. (2014) have proposed an approach enabling the modeling and performance prediction of event-based systems at the architecture level. In this research, the proposed approach integrates platform-specific performance influences of the underlying middleware while enabling the use of different existing analytical and simulation-based prediction techniques. The study had contributed: the development of a meta-model for event-based communication at the architecture level, a platform aware model-to-model transformation and a detailed evaluation of the applicability of the proposed approach based on two representative real-world case studies. The results had shown the effectiveness, practicability and accuracy of the proposed model and prediction approach.

Zschaler (2010) has presented a formal specification of timeliness properties of a component-based system as an example of a formal approach to specifying non-functional properties in this study. The specification was modular and allows reasoning about the properties of the composed system.

Pham and Defago (2013) have proposed a novel extension built upon the core model of a recent component-based reliability prediction approach to offer

an explicit and flexible definition of reliability-relevant behavioral aspects, (i.e., error detection and error handling) of FTMs and an efficient evaluation of their reliability impact in the dependence of the whole system architecture and usage profile. The proposed approach was validated in two case studies by modeling the reliability, conducting reliability predictions and sensitivity analyses and demonstrating its ability to support design decisions.

MATERAILS AND METHODS

Proposed method: Software metrics are suggested to calculate the software quality and presentation features quantitatively, encountered during the planning and implementation of software development. For the purpose of contrast, cost estimation, fault prediction and forecasting these can provide as measures of software products. Several researches have been performed on software metrics and their functions. To improve software with higher quality concern is the most important goal of our suggested method. We have considered improving software with the help of some of the software metrics like reliability and cost in order to progress improved quality software. The reliability and cost can assist us in validating the quality of the specific software that has been proposed. The software parameters are selected with the help of optimization process. The optimization is prepared in order to attain totally related software parameter that outfits our suggested development process. We have exploited Modified Cuckoo Search algorithm (MCS) in suggested method for optimization of software parameters. In Fig. 1 the overall process of the executed method is exposed. The block diagram of the suggested method is demonstrated in beneath

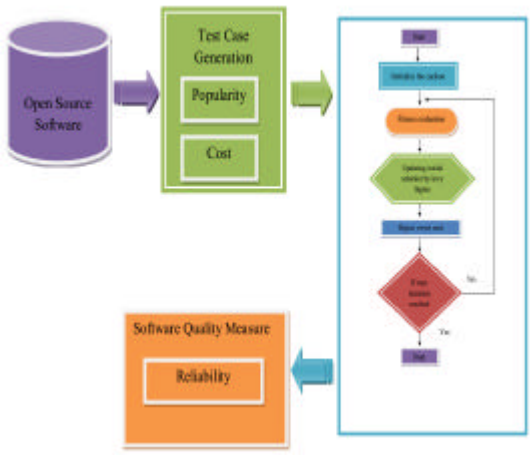

Fig. 1: Block diagram of proposed method

Open source software: For alteration or improvement by anybody open source software is software whose source code is accessible. The maintainability of the software along with reliability, constancy, difficulty and reusability has been a main problem when software is progressed and this participate the most important role in the functioning and a long lifetime of the progressed software.

Test case generation of proposed method: For the purpose of producing of test cases, open source software is specified to the test case generation in this section. Test cases are employed to test all feasible combinations in the application and as well it offers the user to simply replicate the steps that were assumed to expose a defect that is identified during test. Test cases can be charted directly and obtained from use cases. Test cases can as well be obtained from system requirements. Moreover, when the test cases are produced early, Software Engineers can frequently discover ambiguities and inconsistencies in the requirements specification and design documents. This will absolutely get down the cost of building the software systems as faults are eradicated early during the life cycle. Test case generation is a method where the test cases are produced not based on an algorithm but based on the ones statement of the application. Classes will be checked and different test inputs will be offered to make sure for the faults in the application. The generation of test cases is based on the popularity and cost. The result is fed to adapted cuckoo search algorithm for the assessment of software quality.

Popularity: To find the popularity of the open source software, initially we check all the function in a class. Then evaluate the popularity based on the function in each class (F):

$$F = \frac{\text{called by other fun} + (\text{call by this fun} - 1)}{\text{call by this fun}} \quad (1)$$

$$\text{Popularity} = \text{sum of function (F) in each class} \quad (2)$$

Consider an example of open source software; it is clearly explaining the popularity of the open source software. It is explained in Table 1.Here A and B are the two class, A1, A2, B1 and B2 are the function in a class. At first evaluate the F in each function in a class. Then find the popularity of each class:

$$F(A1) = 2 + (3-1)/3 = 2.6$$
$$F(A2) = 1 + (2-1)/2 = 1.5$$
$$F(B1) = 2 + (1-1)/1 = 2$$
$$F(B2) = 2 + (2-1)/2 = 2.5$$

Table 1: Example of popularity

Class A	Class B
A1	B1
{	{
B1, A2, B2	B2
}	}
A2	B2
{	{
B2, A1	A1, B1
}	}

Popularity of class A= 2.6+1.5 = 4.1

Popularity of class B = 2+2.5 = 4.5

Based on these procedures the popularity of open source software is calculated.

Cost: A calculation cost is particularly with the conformance or nonconformance of software product quality. The excellence of any software can rely mainly on the cost of the software. The cost estimation is made based on the implementation time necessary for the software to run. Rise in the time to check the software increases the difficulty of software. It is all the time viewed that allotted time for checking of software can go beyond its necessary schedule time. Automated testing tools are employed to hurry up the testing process. It does not merely hurry up the testing process however it raises the competence of the testing by certain extend. The software cost is estimated based on the time execution of software as well as the error value:

$$\text{Cost} = C_{mt} + C_a(1+r)f_r(t) + C_b\left[f_r(t_j) - (1+r)f_r(t)\right] + C_c\left(\int_0^t x(t)dt\right) \quad (3)$$

Where:

C_{mt} = Cost of adopting a new automated testing tools
r = Directly proportional to cost
C_a = Cost of correcting the error during the testing
C_b = Cost of correcting an error during process
C_c = Cost of testing per unit testing expenditure
$F_r(t)$ = Failure rate

Modified cuckoo search algorithm: Cuckoo search algorithm is a metaheuristic algorithm which was motivated by the breeding behavior of the cuckoos and relieves to execute. There are a number of nests in cuckoo search. Each egg indicates a solution and an egg of cuckoo indicates a novel solution. The novel and better solution is substituting the most terrible solution in the nest. The process of clustering is specified below:

Step 1

Initialization phase: The population (m_i where I = 1,2,... n) of host nest is commenced randomly.

Step 2

Generating new cuckoo phase: By means of levy flights a cuckoo is chosen at arbitrary and it produces novel solutions. Next the produced cuckoo is assessed by means of the objective function for finding out the quality of the solutions.

Step 3

Fitness evaluation phase: Assess the fitness function based on the equation and after that choose the best one:

$$P_{max} = \frac{P_S}{P_T} \quad (4)$$

$$\text{fitness} = \text{maximum popularity} = P_{max} \quad (5)$$

Where:

P_s = Selected population
P_T = Total population

Step 4

Updation Phase: Revise first the solution by levy flights in which cosine transform is employed. The excellence of the novel solution is assessed and a nest is chosen among randomly. If the excellence of novel solution in the chosen nest is better than the old solutions, it will be substituted by the novel solution (Cuckoo). Or else, the earlier solution is set aside as the best solution. The levy flights used for ordinary cuckoo search algorithm is:

$$m_i^* = m_i^{(t+1)} = m_i^{(t)} + \alpha \oplus \text{Levy}(n) \quad (6)$$

By altering the above equation, levy flight equation by employing the gauss distribution is shown in below:

$$\sigma_s = \sigma_0 \exp(-\mu\kappa) \quad (7)$$

Where:

σ_0, μ = Constants
K = Current generation

Step 5

Reject worst nest phase: The worst nests are thrown away in this part, based on their possibility values and novel ones are built. Later, based on their fitness function the best solutions are graded. Next the best solutions are recognized and spotted as optimal solutions.

Step 6; Stopping criterion phase: Until the maximum iteration accomplishes this process is replicated.

Fig. 2: Flowchart for modified cuckoo search

The optimized result will be examined for the measure of software quality. The specified process is obviously demonstrated in flowchart. It's shown in beneath (Fig. 2)

Software quality measure: Software quality measure is to estimate quality of the software. Here we used to calculate the software quality by reliability.

Reliability: In a particular environment, software reliability is described as the possibility of failure-free software operation for a given period of time. The excellence of software testing is openly associated to reliability growth. During the software designing the mistakes are mainly generated and these mistakes confirm to be the main reason for finding out the reliability of software. The amount of mistake in that software must be calculated approximately correctly and should be eliminated. The reliability can be calculated by estimating the testing effort of the specific software. The failure rate regarding the time of implementation can be computed and this presents as the reliability of that specific software at the implementation time. The software reliability is estimated based on the failure rate that is obtained from the software. The reliability of the software can be calculated during the expression specified below:

$$\text{reliability} = \frac{\text{failure rate } (f_r)}{\text{execution time}} \quad (8)$$

RESULTS AND DISCUSSION

As per our proposed method, we have used a Balloon tooltip open source software version 2.1. For the above software, we have estimated the fitness values using the optimization technique for test case optimization. The implementation is done in the working platform of JAVA. The computational cost and the reliability measure is also being measured and its average value is compared with that of the existing method. In this reseach at first we generate certain test case values and for these test cases the optimization process is carried which generate optimized output. The optimization helps in choosing the required test cases for generating our system. The fitness values for the proposed MCS algorithm as well as the existing PSO are then tabulated in correspondence to the different iterations. The fitness value for MCS is calculated using the below expression, Assess the fitness function based on the equation and after that choose the best one:

$$P_{max} = \frac{P_S}{P_T}$$

$$\text{fitness} = \text{maximum popularity} = P_{max}$$

Where:

P_s = Selected population
P_T = Total population

The fitness value for PSO is calculated using the below expression, Assess the fitness function based on the equation and after that choose the best one.

$$\text{fitness} = \frac{g_s}{P_s}$$

After fitness value generation, the reliability and the cost of are calculated for different iterations. Finally, the obtained reliability is compared with that of other existing method and the comparative graph is plotted. The reliability of the software can be measure dusing the expression given below:

Table 2: Fitness value for different iterations

No of Iterations	Fitness value using PSO	Fitness value using MCS
5	0.87	2.31
10	0.95	2.37
15	1.22	2.62
20	1.44	2.64
25	1.69	3.24

Table 3: Reliability and Computational Cost (CC) estimate in relation to execution time

No. of iterations	Reliability	CC
5	0.07634	20275.39
10	0.08078	21157.64
15	0.09553	24670.38
20	0.07786	21170.11
25	0.07079	19753.80

$$r(t) = f(t) / e_t \quad (10)$$

Where:

$f(t)$ = Failure rate at time (t)

e_t = execution time

The cost is calculated based on the below expression:

$$C_t = C_{0t} + C_1(1+k)f(t) + C_2\left[f\left(t_j\right) - (1+k)f(t)\right] + C_3\left(\int_0^t x(t)dt\right)$$

Where:

C_{0t} = Cost of adopting a new automated testing tools into testing phase

K = Directly proportional to cost as k increases cost also increases

C_t = Total cost of the software

C_1 = Cost of correcting the error during the testing,

C_2 = Cost of correcting an error during operation

C_3 = Cost of testing per unit testing expenditure

Table 2 shows the fitness values for our proposed Modified Cuckoo Search (MCS) and that of existing PSO method. The value shows that our proposed method has delivered better fitness value. The graphical representation of the fitness value for proposed and existing approach are shown in below Fig. 3. From the above graph it can be inferred that our proposed modified cuckoo search delivers better fitness values in terms of different iteration values when compared with that of PSO algorithm.

Table 3 shows the reliability and the cost value obtained from our proposed method for various iterations. The estimated reliability and the CC values are being tabulated and the graph is plotted. The graphical representation of reliability and cost for various iterations are shown in Fig. 3 and 4, respectively, Table 4 given below shows the reliability comparison of our proposed

Table 4: Comparison of reliability value of our proposed method with existing method

Method	Reliability
Proposed method	0.08026
Existing method (Joseph *et al*., 2011)	0.0015

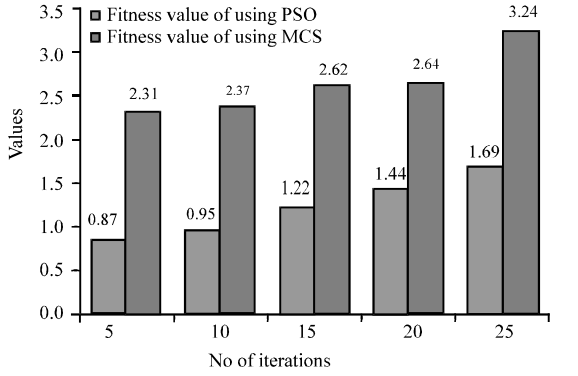

Fig. 3: Graphical representation of fitness values for MCS and PSO

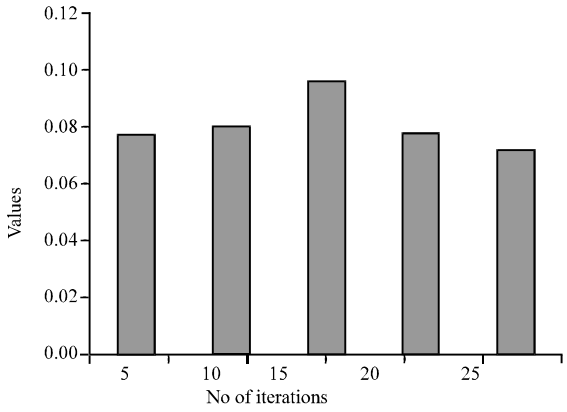

Fig. 4: Graphical representation of reliability versus iteration

methods with the existing method. The existing method is Model for Reliability Estimation of Software based Systems by Integrating Hardware and Software (Joseph *et al*. 2004). The major drawback that exists in the previous work is the reliability value which is much lesser. The reliability value is the major attribute that initiate the quality of particular software being designed. Hence it is considered as the major drawback in the system. Another problem in the existing work is regarding the test cases. The number of test cases is too high which makes the software design more complicated. The test cases should be chosen such that there will be those which can effectively support the software design with higher quality. Hence test case selection remains to be a major

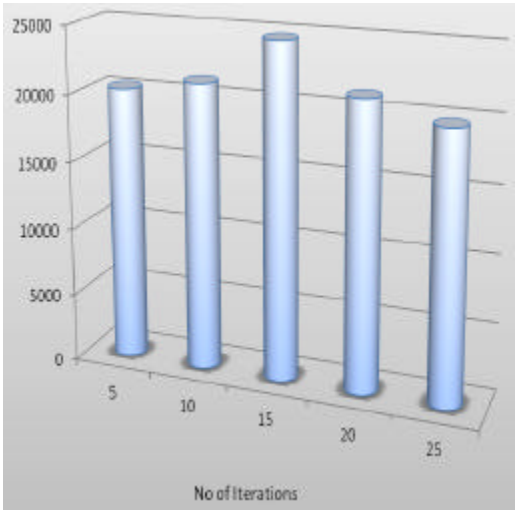

Fig. 5: Graphical representation of cost versus iteration

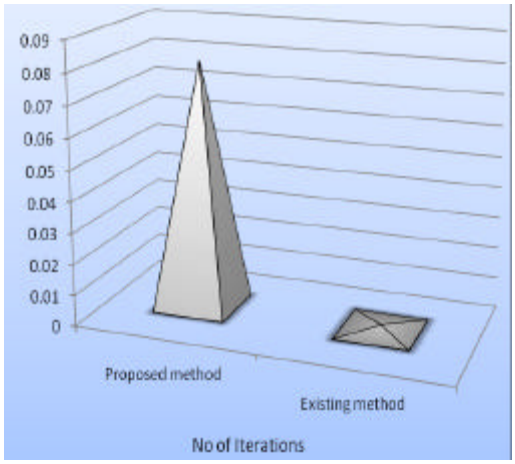

Fig. 6: Graphical representation of reliability values for existing and proposed methods

requirement in the software design process. Hence to solve these drawbacks we have proposed the optimization based software quality measurement technique that effectively increases the software quality by considering the reliability factor. The values are then plotted in graphical representation and is shown in Fig. 5 and 6. From the values it is clear that our proposed method delivers better reliability which further conclude that the software is of better quality.

CONCLUSION

In this study we have propose a method for Estimation of Software Quality based on Software Metrics. We have considered several software metrics like popularity, reliability and cost. The proposed method uses the popularity and cost for generating the test cases. Similarly the reliability of the software metrics used to evaluate the quality of the software. The method uses the Modified Cuckoo Search algorithm (MSC) for the optimization of software metrics. Then the optimized result will be evaluating the quality measure using the software metrics. The result shows that our proposed method achieves better quality of the software.

REFERENCES

Abdellatief, M., A.B.M. Sultan, A.A. Abdul Ghani and M.A. Jabar, 2013. A mapping study to investigate component-based software system metrics. J. Syst. Software, 86: 587-603.

Cai, X., M.R. Lyu, K.F. Wong and R. Ko, 2000. Component-based software engineering: Technologies, development frameworks and quality assurance schemes. Proceedings of the Seventh Asia-Pacific Conference on Software Engineering APSEC 2000, December 8-8, 2000, IEEE, Sha Tin, China, ISBN: 0-7695-0915-0, pp: 372-379.

Chaitanya, P.G. and K.V. Ramesh, 2011. Feasibility study on component based software architecture for large scale software systems. Int. J. Comput. Sci. Inf. Technol., 2: 968-972.

Chaudron, M.R.V., E.M. Eskenazi, A.V. Fioukov and D.K. Hammer, 2001. A framework for formal component-based software architecting. Proceedings of the Workshop on OOPSLA Specification and Verification of Component-Based Systems, October 14-14, 2001, Iowa State University, Ames, Iowa, pp: 73-80.

Huang, G., H. Mei and F.Q. Yang, 2006. Runtime recovery and manipulation of software architecture of component-based systems. Autom. Software Eng., 13: 257-281.

Joseph, S., P.V. Shouri and V.P. Jagathy Raj, 2011. A model for reliability estimation of software based systems by integrating hardware and software. Int. J. Comput. Appl., 1: 26-29.

Kadirkamanathan, V., K. Selvarajah and P.J. Fleming, 2006. Stability analysis of the particle dynamics in particle swarm optimizer. IEEE. Trans. Evol. Comput., 10: 245-255.

Khan, A., K. Khan, M. Amir and M.N.A. Khan, 2014. A component-based framework for software reusability. Int. J. Software Eng. Its Appl., 8: 13-24.

Khoshgoftaar, T.M. and N. Seliya, 2003. Analogy-based practical classification rules for software quality estimation. Empirical Software Eng., 8: 325-350.

Liang, J.J., A.K. Qin, P.N. Suganthan and S. Baskar, 2006. Comprehensive learning particle swarm optimizer for global optimization of multimodal functions. IEEE Tans. Evol. Comput., 10: 281-295.

Luders, F., I. Crnkovic and P. Runeson, 2005. Adopting a Component-Based Software Architecture for an Industrial Control System-A Case Study. In: Component-Based Software Development for Embedded Systems. Colin, A., C. Bunse, H.G.Gross and C. Peper (Eds.). Springer, Berlin, Germany, ISBN: 978-3-540-30644-3, pp: 232-248.

Mikaelian, T., B.C. Williams and M. Sachenbacher, 2005. Model-based monitoring and diagnosis of systems with software-extended behavior. Proceedings of the 20th National Conference on Artificial Intelligence, July 9-13, 2005, AAAI, Pittsburgh, Pennsylvania, pp: 327-333.

Pham, T.T. and X. Defago, 2013. Reliability prediction for component-based software systems with architectural-level fault tolerance mechanisms. Proceedings of the 2013 Eighth International Conference on Availability, Reliability and Security (ARES), September 2-6, 2013, IEEE, Nomi, Japan, ISBN: 978-0-7695-5008-4, pp: 11-20.

Rathfelder, C., B. Klatt, K. Sachs and S. Kounev, 2014. Modeling event-based communication in component-based software architectures for performance predictions. Software Syst. Model., 13: 1291-1317.

Ratnaweera, A., S.K. Halgamuge and H.C. Watson, 2004. Self-organizing hierarchical particle swarm optimizer with time-varying acceleration coefficients. IEEE Trans. Evolutionary Comput., 8: 240-255.

Sagredo, V., C. Becerra and G. Valdes, 2010. Empirical validation of component-based software systems generation and evaluation approaches. CLEI. Electron. J., 13: 1-13.

Shukran, M.A.M., Y.Y. Chung, W.C. Yeh, N. Wahid and A.M.A. Zaidi, 2011. Artificial bee colony based data mining algorithms for ation tasks. Mod. Appl. Sci., 5: 217-231.

Sirobi, N. and A. Parashar, 2013. Component based system and testing techniques. Int. J. Adv. Res. Comput. Commun. Eng., 2: 2378-2383.

Yacoub, S., B. Cukic and H.H. Ammar, 2004. A scenario-based reliability analysis approach for component-based software. IEEE Trans. Reliability, 53: 465-480.

Zschaler, S., 2010. Formal specification of non-functional properties of component-based software systems. Software Syst. Model., 9: 161-201