Journal of Software Engineering and Applications, 2014, 7, 934-942
Published Online October 2014 in SciRes. http://www.scirp.org/journal/jsea
http://dx.doi.org/10.4236/jsea.2014.711083

Significant Factors for Reliability Estimation of Component Based Software Systems

Kirti Tyagi[1], Arun Sharma[2]

[1]Ajay Kumar Garg Engineering College, Ghaziabad, India
[2]Department of Information Technology, Indira Gandhi Delhi Technical University for Women (IGDTUW), Kashmere Gate, Delhi
Email: Kirti.twins@gmail.com, Arunsharma2303@gmail.com

Received 29 August 2014; revised 20 September 2014; accepted 15 October 2014

Abstract

Software reliability is defined as the probability of the failure-free operation of a software system for a specified period of time in a specified environment. Traditional approaches for software reliability analysis are black box approaches. These approaches use the software as a whole. At present, main emphasis of software is on reuse, hence component based software applications came into existence. Black box models are not appropriate for these applications. This paper introduces some significant factors for reliability estimation of Component Based Software Applications. Reliability of Component Based Software Application depends upon these factors. This paper also gives the definition of factors and explains its relation with reliability of software application.

Keywords

Failure Rate, Reliability, Component Based Systems, Flexibility, Operational Profile

1. Introduction

This paper is a review of articles and research work that have undergone in past three decade (1980-2014) for estimating CBSS reliability. The data are taken from most prestigious journals and website covering area of CBSS. On the basis of above literature survey we find some important factors for reliability estimation of CBSS. These factors play an important role in estimating reliability of CBSS. Some of these factors are application-related and others are component-related. Year wise statistics is shown in **Figure 1**.

It is easy to do the measurement in other engineering field, but not in software engineering. Though annoying, the chase of measuring software reliability has never ceased. Till now, we don't have any good way for measuring software reliability. Since, we don't have good understanding of the nature of software. Measuring software

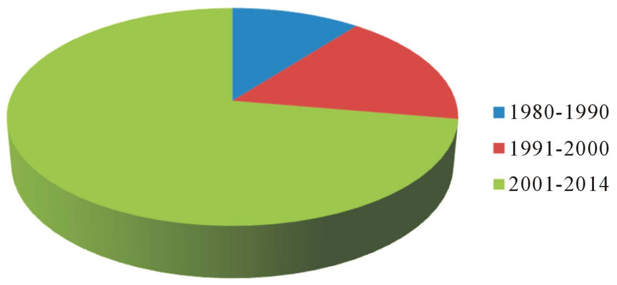


Figure 1. Statistics of literature review.

reliability has always been a tough job. Still, there is no precise definition to what aspects are associated with software reliability. It is difficult to find a suitable way for measuring software reliability, and other aspects which are associated with software reliability. Even, till now, we don't have a specific definition for the most common product metrics, *i.e.* software size.

It is alluring to measure something associated with reliability to reveal the characteristics, if we can't quantify reliability directly. The practices for measuring software reliability can be divided into four categories:

1) Project Management Metrics;
2) Product Metrics;
3) Fault and Failure Metrics;
4) Process Metrics.

Reliability is defined as the probability of successfully completing a given system task. The main aim of software reliability engineering is to model the failure behaviour of software systems to estimate and project reliability. Software reliability estimate serve many purposes such as allows software users to be selective about the software they purchase by considering the advertised software reliability. Reliability measurements are also useful in supporting management of the software development process. However, at this time, the possible benefits of estimating software reliability are not widely acknowledged in the software community. Hence, estimation of software reliability is both necessary and beneficial.

The software evolution has distinct phases or layers of growth. These layers are built up one by one over the last ten decades, with each layer come up with the improvement over the previous one and fulfilling the need of the time. Software evolution which begins with understanding the concept of 1 and 0 (*i.e.* bit) gives rise to machine language, followed by assembly language, procedure-oriented, object oriented, component oriented to service oriented (**Figure 2**).

2. Literature Review

The major difference between software and other engineering artefacts is that software is pure design. Software unreliability is always the result of design faults, which in turn arise from human failures. Hardware systems do fail through design and manufacturing defects more often than is desirable. CBS reliability greatly depends upon the interaction among components. Interaction promotes dependencies. Higher dependency leads to a complex system, hence reliability estimation will be difficult.

Goseva and Trivedi [1] presented a framework for software reliability modelling based on Markov renewal process which naturally introduced dependence among successive software runs. The presented approach enabled the phenomena of failure clustering to be specifically characterized and also its impacts on software reliability to be analysed. Furthermore, it also provided base for a more consistent and flexible model formulation and solution. The Markov renewal model presented can be related to the existing software reliability growth models, means, most of them can be derived as special cases under the assumption of failure independence. Markov renewal model formulation had several advantages, both theoretical and practical. It provided flexible and more consistent modelling of software reliability since the model was constructed in two stages. First, the outcomes of successive software runs were considered to construct the model in discrete time. Then, the execution times of the software runs was considered to build a model in continuous time. The model was adaptable to both dependent and independent sequences of software runs since the model naturally introduced dependence among successive software runs, that is, failure correlation. Considering the independence among software runs

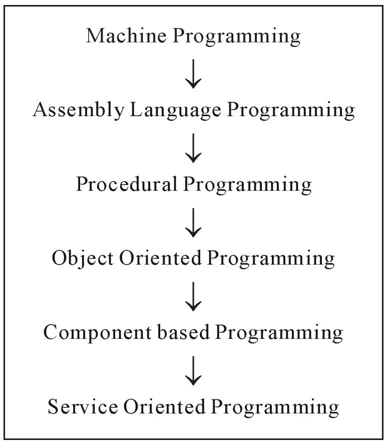

Machine Programming
Assembly Language Programming
Procedural Programming
Object Oriented Programming
Component based Programming
Service Oriented Programming

Figure 2. Technology evolutions.

is a special case of the proposed modelling framework. The model was applicable to different phases of the software life cycle since the proposed modelling approach was applicable for testing (debugging) phase, as well as for validation phase and operational phase

Hamlet *et al*. [2] presented a foundational theory for reliability of software system based on components. The theory has described how component developers could design and test their components to produce measurements which can later be used for calculating composite system reliability by system designers without implementation and testing of the designed system. The theory has described how to make component measurements that were independent of operational profile, and how to calculate system reliability incorporating the overall system-level operational profile. In principle, the fundamental problem of assessing a component, *i.e.* a component developer could not know how the component will be used and so could not certify it for an arbitrary use; but if the component consumer must validate each component prior to its use, Component-Based Development loses much of its appeal has been resolved by the theory. This dilemma was resolved if the component developer does the certification and give the results in such a manner that the component buyer can factor in the usage information later without repeating the certification. The theory addressed the vital technical issues inherent in certifying components to be released for later use in an arbitrary system.

According to Lo *et al*. [3], the parameters in the software reliability models are usually directly obtained from the field failure data. Since, the systems have dynamic properties and the failure data is insufficient, it is really tough to determine the values of the parameters accurately. To deal with this problem, sensitivity analysis has been basically used at this stage. Sensitivity analysis has provided a way to analyze the impact of different parameters. In order to assess the reliability of Component-Based Software, author proposed a new approach to analyze the reliability of the system, based on the reliabilities of the individual components and the architecture of the system. Furthermore, author had also presented the sensitivity analysis on the reliability of component-based software in order to determine which of the components affects the reliability of the system most.

Gayen and Misra [4] proposed an innovative approach to predict the upper and lower bound on the reliability of the COTS Component-Based software application. On the basis of the execution scenario analysis for the COTS Component-Based Software System, a distinctive methodology was formulated. The proposed algorithm for the prediction of upper bound was an improvement over Dolbec and Shephard [5] model for reliability estimation of Component-Based Software. The drawback of his model was that it was execution path independent and component interfacing time was not taken into account. Therefore, it was unable to predict the upper bound on reliability, as the upper bound on reliability obtained using Dolbec and Shephard [5] model was much less than the value obtained in the proposed approach. The proposed approach was applicable under any processing environment be it batch or parallel processing in a uniprocessor or a multiprocessor system. Since, in this approach one was mainly concerned with the component usage ratio and not with the clock time for the execution of the component. Hence, it was assumed that the developer was an ideal developer who codes correctly (as the reliability of the interfacing code was considered to be unity) interfacing the COTS components without any error (which may later on cause failure of the application system) to produce the product which may not always be in reality.

Fan Zhang *et al*. [6] proposed a model based on a CDG. In this model, an operational profile of a system is given, and the model can be used to check whether reliability changes when the operational profile changes. Assuming that control flow transits from component *i* to component *j*, component *j*'s reliability is calculated as

$$T_{ij} \times \left(R_{ij} \times W_{ij}\right),$$

where

T_{ij} = the transition probability from component *i* to component *j*,

R_{ij} = the reliability vector for each subdomain of component *j*, and

W_{ij} = the weight vector for each subdomain of component *j* for the transition from component *i* to *j*.

Dong *et al*. [7] proposed a method for CBSS reliability estimation in which component relationships are analyzed and solved using a Markov model. This technique extends the scope of Markov models. A limitation of this model is that it assumes that all component reliabilities and transition probabilities are given, but in practice this is not always true.

Huag *et al*. [8] proposed a technique based on algebra which provides a framework that can be implemented on Maude for describing syntax and predicting reliability.

Goswami and Acharya [9] proposed an approach to CBSS reliability analysis which takes into consideration the system's component usage ratio, calculated through mathematical formulas. Due to the flexibility of the component usage ratio, this approach may be used for real-time applications. This approach quantified overall software system reliability based on the individual component reliabilities which are combined together to form the system. Moreover, the components which have longer execution time contribute more towards overall system reliability. Operational profile of a component was used for the calculation of component usage ratio, which was a very different function from the traditional use of operational profiles in software reliability engineering.

Overall system reliability was summation of terms weighted by path propagation probability where each term was a summation of individual component reliability weighted by their respective Component Usage Ratio for that particular assembly of components of a given execution path. This approach lends itself to implementation for calculating reliability of COTS Component-Based Software System when reliability of individual components and their usage structures were known in advance.

Si *et al*. [10] proposed a framework for estimating reliability through a component composition mechanism. The approach proposes five basic component composition mechanisms and techniques for their reliability estimation. After calculating the reliability for each composition, a procedure estimates the overall application reliability based on the component composition mechanisms and component utilization frequencies. It is possible to recognize additional composition mechanisms.

Hsu *et al*. [11] proposed an adaptive reliability estimation technique using path testing for complex component-based systems. Three methods are proposed for estimating path reliability, namely, sequence, and branch and loop structures. The proposed path reliability can then be used for estimating the reliability of the overall application. This approach can provide a promising estimation of software reliability when testing information is available. A sensitivity analysis is also performed to determine the effect of each node on the system's reliability.

Wang and Huang [12] proposed an approach for reliability estimation based on rewrite logic (RABRL). This method considers systems whose specification is given with an operational profile. Maude's rewrite technique is used to estimate the reliability. This technique statistically analyzes an application's execution process and uses this to approximately estimate the transition probabilities between components and the expected number of visits to components. However, this approach has some limitations: First, it can only be applied to simple CBSs, and second, it does not consider failure dependencies between components.

Shukla *et al*. [13] presented a systematic method which provided a step-by-step procedure for developing operational profiles for software components. The method has used both usage data and intended usage assumptions to discover a usage distribution, usage structure and characteristics of parameters. The usage distribution defined the probabilities of the operations. The method developed a usage structure using guidelines from the information gathered in the first step, and then modelled the usage structure using state charts. The usage quantification process produced a usage distribution. The parameter analysis process defined parameter characteristics. To demonstrate the method, author applied it to the Symbol Table and tree components of the PGMGEN testing tool. The method was intended to be part of a more general framework for the reliability assessment of software components. The framework has included support for test case execution and output evaluation.

Lance Fiondella *et al*. [14] proposed an approach based on correlated component failures (COCOF). In this paper an efficient approach to access the reliability of a software application, considering the component reliabilities, correlation and application architecture is proposed. This proposed approach is based on an algorithm that transforms a Multivariate Burnoulli distribution (MVB) into a joint distribution of the component outcomes.

The effectiveness of the approach was demonstrated through two experiments, because of its efficiency, the approach is also suitable for analyzing the sensitivity of the application reliability to component to correlation parameters. This approach is simple and efficient. The approach may be applied to large systems to identify correlations that impede system reliability.

Hai Hu *et al*. [15] proposed a software reliability estimation method using modified adaptive testing (MAT) for reliability of CBS. In this paper, a set of extended metrics based on the Nelson's software reliability model account for information gained from a user's point of view regarding the severity of the observed failures. In the approach, the use of test history information allows the resulting test process to be adaptive in the selection of tests under the limited test budget. This approach can enhance the software reliability estimation testing by guiding test case selection process by providing more descriptive and accurate results.

Marko Palviainen *et al*. [16] proposed a technique for reliability estimation, prediction and measuring of CBS. The proposed approach explains that reliability is a key driver for safety critical systems such as health care systems and traffic controllers. This paper gives software reliability evaluation during design and implementation phases. The contribution of this approach lies in the integrating the component level reliability. There is a need for a systematic approach that facilitates software developers to both construct reliable component based software systems and ensure that the software architecture, selected components and constructed software system meet the reliability goals. The approach integrates heuristic reliability estimation, model base reliability prediction and model based reliability measuring activities at component level and reliability prediction activities at system level to support the incremental and iterative development of reliable CBSS. This proposed approach is not applicable for distributed systems.

Soft computing techniques are emerging techniques nowadays. These techniques are comprises of information processing and biological systems. Human type information processing involves both logical and intuitive information processing. Conventional computer systems are good for logical processing but their capability for intuitive information is far behind that of human beings.

The use of intelligent neural networks and hybrid technique in place of traditional statistical techniques has shown a remarkable improvement in the prediction of software reliability in the recent years. Among the intelligent and the statistical techniques it is not easy to identify the best one since their performance varies with the change in data.

Researches also proposed many soft computing technique based approaches for estimating reliability of CBSS. Some of them are discussed as follows:

Lo [17] proposed a software reliability estimation model based on a support vector machine (SVM) and a genetic algorithm (GA). This model assumes that recent failure data alone are sufficient for estimating reliability. Reliability estimation parameters for the SVM are determined by the GA. This model is less dependent on failure data than are other models.

Dimov and Sasikumar [18] proposed a fuzzy reliability model for CBSSs, based on fuzzy logic and possibility theory. A mathematical fuzzy model based on necessity and possibility is proposed to predict the reliability of a CBSS. Like many other models, this model does not require component failure data because it is based on uncertainty. However, a mechanism is necessary to model the propagation of failure between components and failure behavior.

Singh *et al*. [19] presented a Bayesian reliability estimation model using a unified modelling language (UML) technique for reliability prediction and assessment. The technique provides reliability analysis at the design level, *i.e.*, before the system development and integration level. Because this approach is based on UML diagrams, reliability can be predicted in the early design phase. This approach is scalable because all calculations are done by an automated tool. The approach has one limitation: If new paths are taken into account, the reliability prediction algorithm considers this to be a new system, so that separate operational profiles are generated.

It is evident from review of literature review that many models have been proposed for estimating reliability of CBSS. Some of them are architectural model and some are mathematical models. Although all the available models estimate reliability of CBSS up to a great extent, but in our view, reliability is a real world phenomenon many real time issues are associated with it. Hence some soft computing technique based approaches are required to estimate reliability of CBSS.

3. Significant Factors for Reliability Estimation of CBSS

Table 1 shows the significant factors for reliability estimation, which we have find through the literature survey.

3.1. Reusability

Reusability is one of the most basic concepts of component-based development (CBD). As the name suggests, reusability refers to how frequently a component is used in different applications. In case of Component Based Development, software reuse refers to the utilization of a software component C with in a product P, where the original motivation for constructing C was other than for use in P. in other words, reuse is the process of adapting a generalized component to various contexts of use. The idea of reusing software embodies several advantages. It improves productivity, maintainability and quality of software. A reusable component can be seen as a box. These boxes are defined as:

Black box reuse: In black box reuse the re user sees the interface, not the implementation of the component. The interface contains public methods, user documentation, requirements and restrictions of the component. If a programmer were to change the code of black box component, compiling and linking the component would propagate the change to the applications that reuse the component.

Glass box reuse: In glass box reuse the inside of the box can be seen as well as the outside, but it is not possible to touch inside. This solution has an advantage when compared to black box reuse, as re user can understand the box and its use better.

White Box reuse: In white box reuse it is possible to see and change the inside of the box as well as its interface. A white box can share its internal structure or implementation with another box through inheritance or delegation.

3.1.1. Criterion for Selection

We selected reusability as a factor for estimating component reliability because it is believed that components that have been used in many applications are highly reliable. Hence the reliability of a component is directly proportional to its reusability:

$$\text{Component reliability } \alpha \text{ reusability} \tag{1}$$

3.1.2. Empirical Evaluation

We selected components from the site www.ejbeans.com; reusability measures for these components are given on the site.

3.2. Operational Profile

An operational profile (OP), which is a quantitative characterization of how software will be used, is essential in

Table 1. Significant factors.

Component Related Parameters	Application Related Parameters
Reusability	Failure Rate
Operational Profile	Application Complexity
Portability	Component Dependency
Functionality	Flexibility
Failure Probability of Each Component	Faults
Error Propagation Probability of Each Component	Repair Rates
	Mean Time before Failure
	Security

any software reliability engineering application. An operational profile is a complete set of operations along with the probabilities of their occurrence. In a component-based system, the operational profile is usually specified for components that directly interact with the users. Components, which only interact with other components, receive the operational profile in a transformed way. Inputs on the provided interfaces of a component are transformed along the control flow down to the required interfaces. Thus, the provided interfaces of subsequent components connected with the required interfaces receive a different operational profile than the first component. The transformations form a chain through the complete architecture of components until the required interfaces of components only execute functions of the operating system or middleware.

3.2.1. Criterion for Selection

We selected the operational profile as a factor for estimating reliability because reliability may vary for different operational profiles.

3.2.2. Empirical Evaluation

The operational profile for each component is the input set for the component.

3.3. Portability

This factor is defined as the ability of a component to be transferred from on environment to another with little modifications, if required. The component should be easily and quickly portable to specified new environments if and when necessary, with minimized porting costs and schedules. In Component Based Development, it is a very important factor as a component may be used and reused in various different environments. Therefore the specification of the component should be platform independent. Some components are platform independent that are highly portable. The components that are available on the site www.jars.com are highly portable because they do not require any change in the environment whenever they are used in different applications. Whereas some of the components required changes whenever used from one application to another. So the portability of the component is among one of the most important factors for reliability of Component Based Software System. Thus in estimating CBSS reliability we can take portability as a highly important factor. If a component is easily portable then the reliability is high.

3.4. Functionality

Functionality of a component depends upon the number of functions and their properties in these functions. It means that the component should provide the functions and services as per the requirement when used under the specified condition. Pre-existing components with or minimum changes will allow low cost, faster delivery of end product. If the functionality is high then the reliability is low.

3.5. Failure Rate

Number of failures per unit time is known as failure rate, e.g., failures per hour or failures per day or failures per year. It is denoted by λ.

$$\text{Failure Rate } \alpha 1/\text{Reliability} \tag{2}$$

Failure rate is inversely proportional to reliability of software application.

3.6. Application Complexity

The complexity of any CBS application can be defined in terms of the number of components in that application and their interconnectivity. An application that has more components is said to be more complex and hence less reliable. Therefore, the complexity of an application is inversely proportional to its reliability:

$$\text{Application Complexity } \alpha 1/\text{reliability} \tag{3}$$

3.7. Component Dependency

In a CBS, various components are connected to one another to form a larger application. The components are dependent on each other to perform their function, that is, the output of one component may serve as an input for another component. This dependency plays an important role in estimating the reliability of the overall application. If components are highly dependent on one another, the reliability of the system will be low.

There are several mechanisms for characterizing dependencies among components. For example, an adjacency matrix representation of dependencies indicates that dependencies exist among components. However, this representation does not explain the interactions between components. These interactions play an important role in relation to the dependencies of components and therefore their reliability, and so we needed a different way to represent the interactions.

3.8. Flexibility

Ease of making changes required by changes in the operating environment. Flexibility is inversely proportional to the reliability.

$$\text{Reliability } \alpha 1/\text{Flexibility} \tag{4}$$

3.9. Repair Rate

Repair rate is expressed in terms of number of repairs per unit time, e.g., repairs per hour or repairs per day. It is denoted by μ.

3.10. Fault

A fault can be defined as a defect in software which causes a failure in it. It is also referred as a "bug". Example of software fault is incorrect statement of a program. A single fault can cause multiple failures in software. A fault is a property of program rather than property of its execution or behaviour.

3.11. Mean Time before Failure

Mean time before failures (MTBF) is the statistical average time before failures of the system.

3.12. Security

The primary goals of software security are the preservation of the confidentiality, integrity, and availability of the information assets and resources that the software creates, stores, processes, or transmit, including the executing programs themselves. In other words, users of secure software have a reasonable expectation that their data is protected from unauthorized access or modification and that their data and applications remain available and stable. Clearly some applications have a need for a much higher degree of assurance than others. Security constraints impact a project in two ways. Functional security requirements increase the functional size of the software system being developed and need to be treated in the same way as all other functional requirements being met by COTS components or home-grown code. Non-functional security requirements to attain a specific level of security assurance require additional processes, documentation, testing, and verifications. COTS components are typically black box products developed by third parties. Using them in your enterprise's information system can introduce significant security and reliability risks. If your organization uses the Internet, for example, COTS components can leak internal information across a globally connected network.

The component may access unauthorized resources or services.

The component may access a resource in an unauthorized way. This can cause another component to fail. For example, if a broken Web server writes to an HTML file it should only read, it can overwrite a home page.

The component may abuse authorized privileges. For example, attackers can commandeer a component running with root ("super user") privilege and take complete control of the system.

It refers how the component is able to control the unauthorized access to its provided services. If the component is highly secure then the reliability of component will be high.

4. Conclusion

This paper identifies some significant factors for reliability estimation of CBSS. On the basis of these factors reliability can be estimated by using any soft computing technique. We have taken some expert advice to find most significant factors out of these significant factors. There may be some other factors also but at this stage by our literature review we are able to find out these factors. This paper may be an insight for estimating reliability of CBSS.

References

[1] Goseva, K. and Trivedi, K.S. (2000) Failure Correlation in Software Reliability Models. *IEEE Transactions on Reliability*, **49**, 37-48. http://dx.doi.org/10.1109/24.855535

[2] Hamlet, D., Mason, D. and Woit, D. (2001) Theory of Software Reliability Based on Components. 23*rd International Conference on Software Engineering* (*ICSE*), Portland State University, 12-19 May 2001, 361-370.

[3] Lo, J.H., Huang, C.Y, Kuo, S.Y. and Lyu, M.R. (2003) Sensitivity Analysis of Software Reliability for Component-Based Software Applications. *Proceedings of the* 27*th International Computer Software and Applications Conference* (*COMPSAC* 2003), Dallas, 3-6 November 2003, 500-505.

[4] Gayen, T. and Misra, R.B. (2008) Reliability Bounds Prediction of COTS Component Based Software Application. *IJCSNS International Journal of Computer Science and Network Security*, **8**, 219-228.

[5] Dolbec, J. and Shepard, T. (1995) A Component Based Software Reliability Model. *Proceedings of the* 1995 *Conference of the Centre for Advanced Studies on Collaborative Research*, Toronto, 7-9 November 1995, 19.

[6] Zhang, F., Zhou, X., Dong, Y. and Chen, J. (2009) Consider of Fault Propagation in Architecture-Based Software Reliability Analysis. *International Conference Computer System and Application*, Rabat, 10-13 May 2009, 783-786.

[7] Wang, D. and Huang, N. (2008) Reliability Analysis of Component Based Software Based on Rewrite Logic. 12*th IEEE International Workshop on Future Trends of Distributed Computing Systems*, 21-23 October 2008, 126-132.

[8] Huang, N., Wang, D. and Jia, X.G. (2008) Fast Abstract: An Algebra-Based Reliability Prediction Approach for Composite Web Services.19*th International Symposium on Software Reliability Engineering*, Mysuru, 16-19 November 2009, 285-286.

[9] Gayen, T. and Misra, R.B. (2008) Reliability Bounds Prediction of COTS Component Based Software Application. *IJCSNS International Journal of Computer Science and Network Security*, **8**, 219-228.

[10] Si, Y.J., Yang, X.H., Wang, X.Y., Huang, C. and Kavs, A.J. (2011) An Architecture-Based Reliability Estimation Framework through Component Composition Mechanisms. 2*nd International Conference on Computer Engineering and Technology*, Chengdu, 16-18 April 2010, 165-170.

[11] Hsu, C.-J. and Huang, C.-Y. (2011) An Adaptive Reliability Analysis Using Path Testing for Complex Component Based Software Systems. *IEEE Transaction on Reliability*, **60**, 158-170. http://dx.doi.org/10.1109/TR.2011.2104490

[12] Wang, D. and Huang, N. (2008) Reliability Analysis of Component Based Software Based on Rewrite Logic. 12*th IEEE International Workshop on Future Trends of Distributed Computing Systems*, 126-132.

[13] Shukla, R., Carrington, D. and Strooper, P. (2004) Systematic Operational Profile Development for Software Components. *Proceedings of the* 11*th Asia-Pacific Software Engineering Conference* (*APSEC*'04), Busan, 30 November-3 December 2004, 528-537.

[14] Fiondella, L., Rajasekaran, S. and Gokhale, S. (2013) Efficient Software Reliability Analysis with Correlated Component Failures. *IEEE Transaction on Reliability*, **62**, 244-255.

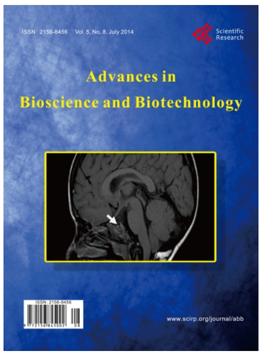
[15] Hu, H., Jiang, C.H., Cai, K.Y., Wong, W.E. and Mathur, A.P. (2013) Enhancing Software Reliability Estimates Using Modified Adaptive Testing. *Information and Software Technology*, **55**, 288-300.

[16] Palviainen, M., Evesti, A. and Ovaska, E. (2011) The Reliability Estimation, Prediction and Measuring of Component-Based Software. *Journal of System and Software*, **84**, 1054-1070. http://dx.doi.org/10.1016/j.jss.2011.01.048

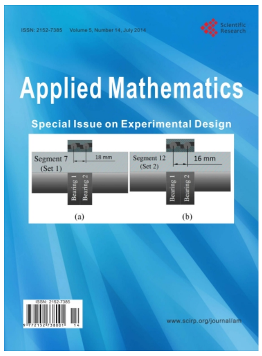
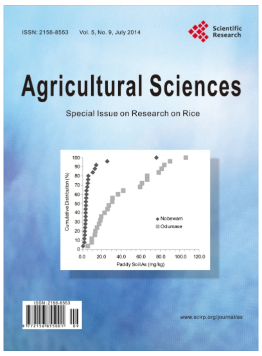
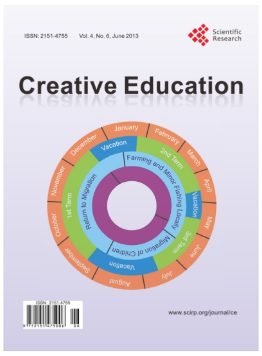
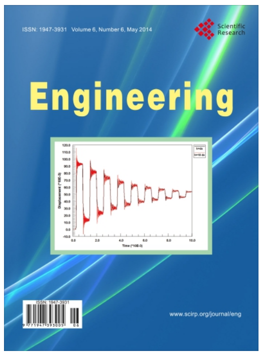
[17] Lo, J.-H. (2010) Early Software Reliability Prediction Based on Support Vector Machines with Genetic Algorithms. *International Conference on Industrial Electronics and Application*, 2221-2226.

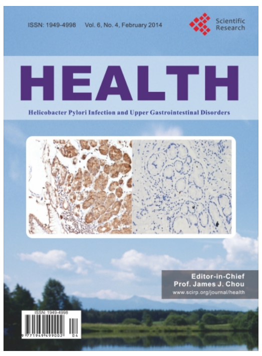
[18] Dimov, A. and Punnekkat, S. (2010) Fuzzy Reliability Model for Component-Based Software Systems. 36*th EUROMICRO Conference on Software Engineering and Advanced Applications*, Sofia, 1-3 September 2010, 39-46.

[19] Singh, H., Cortellessa, V., Cukic, B., Gunel, E. and Bharadwaj, V. (2001) A Bayesian Approach to Reliability Prediction and Assessment of Component Based Systems. 12*th IEEE International*. *Symposium on Software Reliability Engineering*, Hong Kong, 27-30 November 2001, 12-21. http://dx.doi.org/10.1109/ISSRE.2001.989454

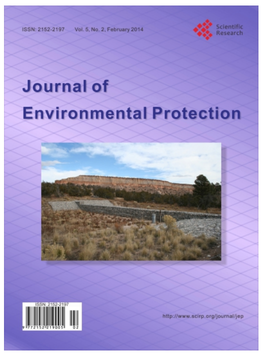
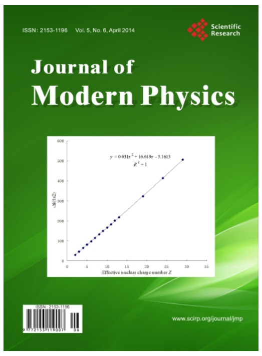

Advances in Bioscience and Biotechnology

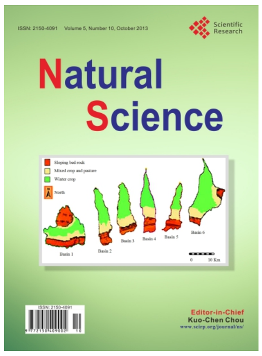
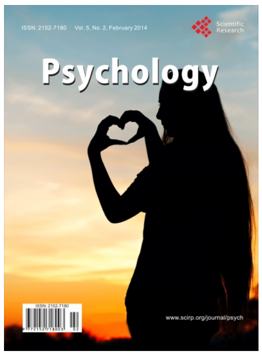
American Journal of Analytical Chemistry

American Journal of Plant Sciences

Applied Mathematics

Agricultural Sciences

Creative Education

Engineering

Food and Nutrition Sciences

HEALTH

Journal of Computer and Communications

Journal of Cancer Therapy

Journal of Environmental Protection

Journal of Modern Physics

Modern Economy

Natural Science

Psychology