International Journal of Multimedia and Ubiquitous Engineering
Vol.9, No.8 (2014), pp.75-82
http://dx.doi.org/10.14257/ijmue.2014.9.8.07

Reliability Analysis of COTS-based Software System

Zhang Nan[1] and Wei Jiamin[2]

[1]*Harbin University of Commerce,harbin,China*
[2] *China Mobile Group Design Institute Co., Ltd.*
Email:zhangnan@hrb.edu.cn, weijiamin@cmdi.chinamobile.com

Abstract

With more emphasis on reuse of software applications, the commercial-off-the-shelf (COTS) based software systems have emerged. As a result, reliability analysis of COTS-based software system has gained prominence. The focus of this paper is to provide an overview for the state of the art of COTS-based software system reliability analysis. In this paper, we first describe the definition of COTS and reliability. Then, we discuss approach and structure of COTS-based software system reliability.

Keywords: *COTS-based, Software reliability, State-based, Path-based.*

1. Introduction

With the development of plug-and-play reusable software, component-based software development (CBSD) has generated tremendous interest in the last decade [1]. In particular, some software vendors have commenced to successfully sell and license COTS components, which have led to the concept of the COTS-based software system [2]. These COTS-based software systems move organizations from application development to application assembly. Current, constructing an application involves the use of prefabricated pieces, perhaps developed at different times, by different people, and possibly with different uses in mind. The ultimate goal, once again, is to be able to reduce development budgets, times, and efforts, while improving the flexibility, reliability, and reusability of the final application due to the reuse of software components already tested and validated [3]. Undoubtedly, a COTS-based software system is one of the leading directions in the software development.

Reliability is usually defined to be the probability of execution without failure for some specified interval of natural units or time [4]. Reliability is a fundamental attribute of software systems, which is also an important index to scale systems' quality. Focusing on safety, reliability analysis aims at the quantification of the probability of failure of the system and its protective barriers. Reliability of tradition software system is based on the non-homogeneous Poisson process in which a software system was considered as black-box. However, the reliability of COTS-based software system considers the architecture of software system namely while-box. The study on the reliability of COTS-based software systems is currently one of the active fields attracting much attention from researchers and practitioners, which is focused on connectivity reliability [5]. With the growing emphasis on using of COTS, there is a need for modeling approaches that are capable of considering the architecture of the software and estimating the reliability. However, there are some questions no answer.

Rest of this paper is organized as follows Section 2 gives the some definitions of COTS. Section 3 describes the reliability. Section 4 provides the approach for COTS-based system reliability analysis. In Section 5, structural analysis of reliability model is given. Paper is concluded with a summary and the description for future work in Section 6.

ISSN: 1975-0080 IJMUE

2. COTS

What is a COTS? First of all, we need to define what we understand by a COST. The term COTS is very generic; it can refer to very broad and levels of software, *e.g.*, software that provides a specific functionality or a tool used to generate code. COTS may be one of the most diversely defined terms in current software development.

Vigder's definition [6]: the COTS component software product is sold in many copies with minimal changes; customer access to source code as well as internal documentation is usually unavailable; customers have no control over specification, schedule and evolution; limitations, performance, or resource consumption, may never have been collected.

Oberndorf's definition [7]: The main characteristics of COTS are: it exists a priori; it is available to the general public and it can be bought (or leased or licensed).

Software Engineering Institute's definition [8]: a COTS component is defined as sold, leased, or licensed to the general public; offered by a vendor trying to profit from it; supported and evolved by the vendor, who retains the intellectual property rights; available in multiple, identical copies; and used without source code modification.

Basili and Boehm's definition [9]: COTS software has the following characteristics: the buyer has no access to the source code; the vendor controls its development and it has a nontrivial installed base (that is, more than one customer; more than a few copies).

Due to market strategies and further variability, there are no widely agreed on standards in the COTS products market. Thus, the definition of COTS products is not a single unified.

3. Reliability

Reliability is usually defined as the probability that a component, or a system, will deliver its intended functionality and quality for a specified period of time, and under specified conditions, given that the system was functioning properly at the start of this time period [10]. Since, software reliability focus on how software is used, software usage information is an important part of reliability evaluation. This includes information on the environment in which software used, as well as the actual frequency of usage of different functions that the software system offers. The usage is quantified through operational profiles. Next, we will give the $R(t)$ mathematical expression.

Let X and $F(t)$ be time to failure of a system and distribution function of system lifetime, respectively. According the definition of reliability, we have

$$R(t) = P(X > t) = 1 - F(t)$$

The software reliability can be measured in many ways. A metric that is commonly used to describe software reliability is failure intensity. Failure intensity is defined as the number of failures experienced per unit time period, which can be computed for all experienced failures, or for some specified category of failures of a given type or severity. Failure intensity is a good measure for reflecting the user perspective of software quality. The relationship between reliability and failure intensity is:

$$R = \exp(-\lambda t)$$

where λ is failure intensity.

In general, mean time to failure (MTTF) is well approximated by the inverse of the failure intensity. Following, we will give the MTTF mathematical expression. According the definition of MTTF, we get

$$MTTF = E[X] = \int_0^\infty tf(t)dt = \int_0^\infty R(t)dt$$

where $f(t)$ is density function of system lifetime.

To increase the reliability as fast as possible, testing is done by finding faults as quickly as possible. The reliability of COTS components is done by black-box testing. Black-box testing only requires a functional description of the program and is done only considering the input/output description of COTS components, nothing about the implementation of the COTS is assumed to be known. Generally, the reliability of COTS can be given by vendor.

4. Approach of COTS-based System Reliability Analysis

Since a COTS-based software system is composed of many COTS components, the COTS-based software system reliability is evaluated through the relationship between the COTS-based components and the reliability of each individual COTS-based component. This method involves the use of specific frequencies of each component. Therefore, it is no need that a system error and system failure rate is proportional to the number of assumptions. There are two approaches that are used to analysis the reliability of COTS-based software system, i.e., path-based approach and state-based approach [11].

4.1. Path-based Approach

The path-based approach considered a sequence of components executed along each path and computed the path reliability by multiplying their reliabilities. Then, the software system reliability is estimated by averaging path reliabilities over all paths [12]. The approach that used to find the possible execution paths of the program is by either experimentally, testing or algorithmically.

4.2. State-based Approach

The state-based approach used the control flow graph (CFG) to describe the architecture of software system. The CFG indicates the system structure, branches in program code, and decision points, thus representing the interaction between components and possible execution paths [13]. The state based approach assumes that the transfer of control between components has been considered as Markov process [11, 13], which means that current components behavior at any given time is independent of the past behavior. In state based approach, software system architecture has been modeled as a continuous time Markov chain (CTMC), discrete time Markov chain (DTMC), or semi-Markov process (SMP).

The state-based approach can be further classified as irreducible Markov chain and absorbing Markov chain [11]. The class of irreducible approach is well suited for continuously operating software applications, such as in real time control systems, where it is either difficult to determine what constitutes a run or there may be very large number of such runs if it is assumed that each cycle consists a run. The class of absorbing approach indicates applications that operate on demand for which software runs that correspond to terminating execution can be clearly identified.

According to the solving methods, the state-based approach can be classified into composite and hierarchical [13]. Table 1 shows the classified of solving methods of stated-based. The composite method combines the architecture of the software with the failure behavior into a composite model, which is solved to predict reliability of the application. The hierarchical method is to solve first the architectural model and then to

superimpose the failure behavior on the solution of the architectural model in order to predict reliability.

Table 1. Classified of Solving Methods of Stated-based

Model	Solving
irreducible CTMC	composite or hierarchical
absorbing CTMC	composite or hierarchical
irreducible DTMC	composite or hierarchical
absorbing DTMC	composite or hierarchical
irreducible SMP	composite or hierarchical
absorbing SMP	composite or hierarchical

4.3. The Comparison of Path-based and State-based Approach

The path-based approach assumes each component is independent of each other, which is an experimental method. Thus, the accuracy of software system reliability is heavily dependent on the initial conditions of experimental process. If the input data of software system is not comprehensive and full, it may interpose final test results. Moreover, the path based approach provides only an approximate estimate for application reliability. If the majority of path exists in the presence of loops and the number of cycles is very large, the final reliability of the system may be lower than the actual circumstances, particularly in an infinite loop system, which is difficult to handle.

The state based approach takes into account the state of the infinite loop system. The state based approach depends on the level of information available and considers the different types of failure modes. Thus, it can be used to the entire development lifecycle, especially testing stage. In the state based approach, the time-varying failure intensity function represents the component failure behavior. Furthermore, hierarchical approach is conducive to sensitivity analysis and predictive analysis of software development.

5. Structural Analysis of Reliability Model

5.1. Common Model

This main purpose of the following discussion is to focus attention on the common structure in which the existing COTS-based software system reliability models have been developed. We classified the structural into four categories namely scope, approach, architecture and model.

Krishnamurthy and Mathur model [14] involves computing the path reliability estimates based on the sequence of software component executed for each test run, averaging them over all test runs to get estimates reliability of software system.

Scope: COTS-based software system.

Approach: the path-based approach.

Architecture: in the simulation, testing and experimental of the program behavior, a sequence of components along different paths is observed by the component traces collected. R_{tc} is multiply of each component reliability of software systems.

Model: the Reliability of software system is given by

$$R = \frac{\sum_{\forall tc \in TS} R_{tc}}{|TS|}$$

where TS is a test case.

Littlewood model [15] is the general component-based software system model.

Scope: COTS-based software system

Approach: irreducible SMP of the state-based approach

Architecture: software architecture of continuously running application can be described by an irreducible SMP. The model assumed the each component failed independently and a finite number of components of software system. The software considered the transfer of control between components is described by the probability p_{ij}. The failure rate of each component is a Poisson process with parameter λ_i.

Model: the failure rate is given by

$$\lambda_s = \sum_i \frac{\pi_i \sum_j p_{ij} m_{ij}}{\sum_i \pi_i \sum_j p_{ij} m_{ij}} \lambda_i + \sum_{i,j} \frac{\pi_i p_{ij}}{\sum_i \pi_i \sum_j p_{ij} m_{ij}} v_{ij}$$

where p_{ij} is the probability of program transits from component i to component j, m_{ij} is finite mean, π_i is the probability of steady state, λ_i is constant failure rate of each component, and v_{ij} is the failure probability of component i calls component j.

Cheung model [16] considered the component's utilization and their reliabilities.

Scope: COTS-based software system.

Approach: absorbing DTMC of the state-based approach.

Architecture: Cheung model assumed the software flow graph of a terminating application has a single entry and a single exit. The transfer of control among components can be described by an absorbing DTMC with a transition probability matrix. The component fail independently and the transition probability is R_n.

Model: the Reliability of software system is given by

$$R = S(1,n)R_n$$

where $S(1,n)$ is probability from N_1 to N_n.

Laprie model [17] is a case of Littlewood model.

Scope: COTS-based software system.

Approach: CTMC of the state-based approach.

Architecture: the transfer of control between components is described by a CTMC. λ_i is constant failure rate of each component. The term $\pi_i\lambda_i$ can be considered as the equivalent failure rate of component i.

Model: the failure rate of system is given by

$$\lambda_s = \sum_{i=}^{n} \pi_i \lambda_i$$

where π_i is the probability of steady state.

Gokhale model [18] combined the architecture of software system and the reliability of components by testing the software system application using the regression test suite.

Scope: COTS-based software system.

Approach: absorbing DTMC of the state-based approach.

Architecture: the software system terminating application is described by an absorbing DTMC. The transition probabilities between component is p_{ij}. The failure behavior of each component of software system application is assumed by enhanced non-homogeneous Poisson process model. Time-dependent failure intensity $\lambda_i(t)$ is computed by block coverage measurements.

Model: the Reliability of software system is given by

$$R = \prod_{i=1}^{n} \exp\left(- \int_0^{\left(q_i + \sum_{j=1}^{n} V_j p_{ji}\right)\tau_i} \lambda_i(t)dt \right)$$

where V_j is the expected number of visits to components j.

Kubat model [19] assumed the case of terminating software applications components designed for K different tasks.

Scope: COTS-based software system.

Approach: DTMC and SMP of the state-based approach.

Architecture: the transitions between components of software system follow a DTMC such that with probability $q_i(k)$ task k will call component i and with probability $p_{ij}(k)$ task k will call component j. The probability λ_i is constant failure rate of each component. The each task is an SMP.

Model: the failure rate of software system is given by

$$\lambda_s = \sum_{k=1}^{K} r_i [1 - \prod_{i=1}^{n} [R_i(k)]^{\sum_{j=1}^{n} V_j(k) p_{ji}(k) + q_i(k)}]$$

where r_i is the arrival rate of task i. $V_j(k)$ is the expected number of visits in component i by task k.

5.2. Example

In this section, we demonstrate the potential of a COTS-based software system through an example. As the study of reliability analysis of the COTS-based software system is in the early stage, the data of COTS-based software system is not available from the vendor. We only use the application reported in literate [16] as the example for illustration. This application has been used extensively to illustrate architecture-based reliability in the recent past. The architecture of the application is shown in Figure 1.

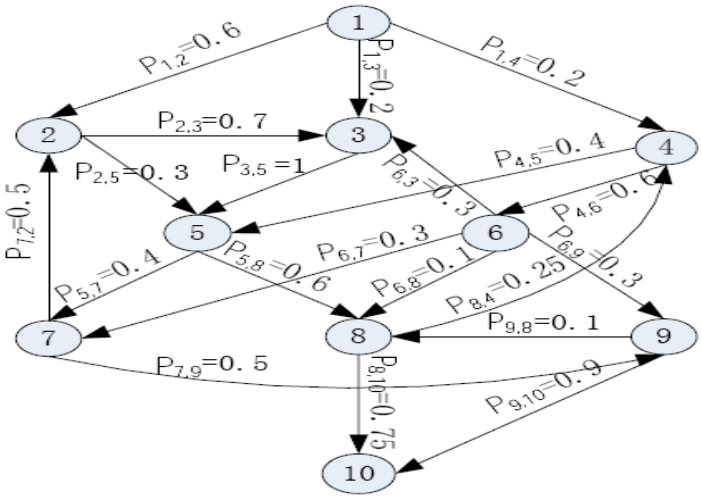

Figure 1. Architecture of the Application

To simplify computation, let the reliability of the components be constants as follows [16]: R1=0.999, R2=0.980, R3=0.990, R4=0.970, R5=0.950, R6=0.995, R7=0.985, R8=0.950, R9=0.975, R10=0.985. According to the Cheung model [16], we can compute the reliability of software system. The reliability of the program is, therefore, 0.8299.

6. Conclusions

This paper provided and overview of the state-of-the-art research in the area of COTS-based software systems reliability analysis. First, various definitions for COTS and reliability are presented. Then, the paper presents reliability analysis for COTS-based software applications. The benefit of COTS-based software application is evident in the software system. However, many questions related to the COTS-based software systems are still unanswered. In further, to evaluation the overall application reliability existing work take some important considerations are reliability of COTS component, COTS

components' operational profile and operational profiles of COTS-based software systems.

References

[1] F. Q. Yang, H. Mei and K. Q. Li, "Software Reuse and Software Component Technology", CHINESE Journal of Electronics, vol. 27, no. 2**, (1999).**
[2] D. Carney and F. Long, "What Do You Mean by COTS?", IEEE Software, vol. 17, no. 2, **(2000).**
[3] M. Morisio, C. B. Seaman and V. R. Basili, "COTS-based Software Development: Processes and Open Issues", Journal of Systems and Software, vol. 61, no. 3, **(2002).**
[4] J. D. Musa, "Software Reliability Engineering, More Reliable Software Faster and Cheaper", New York: McGrwa-Hill, **(2004).**
[5] N. F. Schneidewind, "Methods for Assessing COTS Reliability, Maintainability, and Availability", Proceedings of the Software Maintenance, **(1998)** November 16-20, CA, USA.
[6] M. Vigder, M. Gentleman and J. Dean, "COTS Software Integration: State of the Art", Technical Report NRC No. 39190, **(1996).**
[7] P. Oberndorf, "COTS and Open Systems", http://splc.sei.cmu.edu/library/assets/cotsopensystems.pdf, **(1998).**
[8] L. Brownsword, T. Oberndorf and C. Sledge, "Developing New Processes for COTS Based Systems", IEEE Software, vol. 17, no. 4, **(2000).**
[9] V. Basili and B. Boehm, "COTS-Based Systems Top 10 List", IEEE Computer, vol. 34, no. 5, **(2001).**
[10] S. L. Ho, M. Xie and T. N. Goh, "A Study of the Connectionist Models for Software Reliability Prediction", Computer and Mathematics with Applications, vol. 46, no. 7, **(2003).**
[11] S. S. Gokhale, "Architecture-Based Software Reliability Analysis: Overview and Limitation", IEEE Transactions on Dependable and Secure Computing, vol. 4, no. 1, **(2007).**
[12] C. J. Hsu and C. Y. Huang, "An Adaptive Reliability Analysis using Path Testing for Complex Component-based Software Systems", IEEE Transactions on Reliability, vol. 60, no. 1, **(2011).**
[13] S. S. Gokhale and R. T. L. Michael, "A Simulation Approach to Structure-based Software Reliability Analysis", IEEE Transactions on Software Engineering, vol. 31, no. 8, **(2005).**
[14] S. Krishnamurthy and A. P. Mathur, "On the Estimation of Reliability of a Software System using Reliabilities of its Components", Proceedings of the Eighth International Symposium on Software Reliability Engineering, **(1997) November** 2-5, IN, USA.
[15] S. S. Gokhale, "Architecture-based Software Reliability Analysis Overview and Limitations", IEEE Transactions on dependable and secure computing, vol. 4, no. 1, **(2007).**
[16] W. L. Wang, P. Dai and M. H. Chen, "Architecture-based Software Reliability Model", Journal of Systems and Software, vol. 2, no. 79, **(2006).**
[17] J. C. Laprie and K. Kanoun, "X-Ware Reliability and Availability Modeling", IEEE Transactions on Software Engineer., vol. 15, no. 5, **(1992).**
[18] S. S. Gokhale, "An Analytical Approach to Architecture based Software Reliability Prediction", Proceedings of the Third International Computer Performance and Dependability Symposium, **(1998)** September 7-9, NC, USA.
[19] K. G. Popstojanova and K. S. Trivedi, "Architecture-based Approach to Reliability Assessment of Software Systems", Performance Evaluation, vol. 8, no. 45, **(2001).**

