Simulation techniques for component-based software reliability modeling with project application

Ruohao Huang, Michael Lyu, Karama Kanoun

Simulation Techniques for Component-Based Software Reliability Modeling with Project Application

Ruohao Huang , Michael R. Lyu
Dept. of Computer Science and Engineering
The Chinese University of Hong Kong

Karama Kanoun
LAAS/CNRS
Toulouse, France

Abstract

In this paper, we consider to combine analytical models with simulation techniques for software reliability measurements. We have implemented a set of failure-rate-based simulation techniques which can capture the characteristic of software process and structure in a way that permits us to obtain quantified results for software reliability measures. We address two methods to take into account the functional dependency and error correlation among components, so that we can treat a software system as a combination of interdependent components. This offers a more appropriate approach for analyzing reliability measurements of component-based software systems. The results from a project application indicate that the incorporation of simulation techniques into analytical models has the advantages of accurate analyses, early predictions, and comprehensive evaluations for software reliability engineering.

Keywords: Software Reliability Engineering, Analytical Model, Simulation, Component-based software, Project Application

1. Introduction

Most existing analytical methods to obtain reliability measures for software systems are based on the Markovian models [1,2], and they rely on the assumption on exponential failure time distribution. The Markovian models are subject to the problem of intractably large state space. Methods have been proposed to model reliability growth of components which can not be accounted for by the conventional analytical methods [3, 4], but they are also facing the state space explosion problem. Discrete event simulation, on the other hand, offers an attractive alterative to analytical models as it can capture a detailed system structure when performing software reliability analysis. Some simulation methods have been proposed [6], and a detailed description of simulation techniques for software reliability analysis and evaluation can be found in [1]. However, for component-based software systems, it is difficult to analyze the influence to reliability caused by dependency among components. The main contribution of this paper, therefore, lies in demonstrating the effectiveness and flexibility offered by an architecture-oriented simulation framework to analyze reliability measures for software systems with dependent components. The results of a practical application of our techniques will also be provided.

The rest of this paper is organized as follows: Section 2 presents rate-based simulation methods as building blocks for software reliability measurements. Section 3 gives the implementation description of the rate-based simulation techniques, in which two approaches are given to take into account the interdependency of components. Section 4 gives the simulation results of a project application. Finally, Section 5 presents conclusions.

2. Simulation method

2.1 General description

Here, simulation refers to the technique of imitating the character of an object or process in a way that permit us to make quantified inferences about the real object or process. In the area of software reliability, simulation can mimic key characteristics of the processes that create, validate, and revise documents and code. Furthermore, simulation can distinguish faults that have been removed from those that have not, and thus can readily reproduce multiple failures due to the same as yet unrepaired fault cases applied. Generally, there are two main types of software reliability simulation ways, one is rate-based simulation, the other is artifact-based simulation. For the artifact-based simulation: we consider many aspects of program construction and testing to investigate the effect of static features on dynamic behavior, the inputs may

include those which characterize code structure, coding errors, test input data, test conduct, failure characteristics, debugging effectiveness, and computing environment. In this paper, we used rate-based simulation way to get some results for studying software reliability measurements.

2.2 Rate-based simulation

It is a rate-controlled event process simulation way, the fundamental basis of this simulation method is the representation of a stochastic phenomenon of interest by a time series x(t) whose behavior depends only on a rate function, call it $\beta(t)$, where $\beta(t)*dt$ acts as the conditional probability that a specified event occurs in the infinitesimal interval (t,t+dt).

We use the following rate functions in our implemented simulation scheme. Except these may differ significantly in their assumptions about underlying failure mechanism, they differ mathematically only in the forms of their rate functions.

(1) The Goel-Okumoto (GO) model treats an overall reliability growth process with $\beta(t)=n_0\phi e^{-\phi t}$, where n_0 and ϕ are input parameters, $n_0\phi$ is the initial failure rate, and ϕ is the failure rate decay factor.
(2) The Jelinski-Moranda (JM) model describes statistics of failure time intervals under the presumption that $\beta_n(t)=\beta_0(1-n/n_0)$, where n_0 is the estimated (unknown) number of initial software faults and β_0 is initial failure rate.
(3) The Littlewood-Verrall inverse linear model is an overall reliability growth model with $\beta(t)=\beta_0/(1+\theta t)^{1/2}$ where β_0 is the initial failure rate and θ is a rate decay factor.
(4) The Musa-Okumoto model [6], in which $\beta(t)= \beta_0/(1+\theta t)$, where β_0 is the initial failure rate and θ is a rate decay factor. Both β_0 and θ are input parameters.
(5) The Yamada S-shaped model, its failure rate function is $\beta(t)=ab^2te^{-bt}$, where a is the number of failures to be expect occur and b corresponding to a failure detect rate.

3. Simulation implementation

In this section we describe the implementation of the simulation. It is a failure rate-based simulation, in which the above seven failure rate functions are used as simulation models. It can treat a software system as a whole for simulating reliability measures, which is also known as black-box simulation. We also provides white-box simulation, in which the components of a software system are not be treated as independent each other, the dependencies among components of a software system are considered.

3.1 General simulation assumptions

Assumptions and observed data are very important for software reliability study [7]. For the simulation we have the following assumptions. Note they can be seen as the most common assumptions for software reliability models [8].

(1) The software under testing remains essentially unchanged throughout testing, except for the removal of faults as they are found.
(2) Removing a fault does not affect the chance that a different fault will be found.
(3) "Time" is measured in such a way that testing effort is constant.
(4) All faults are of equal importance (i.e., they contribute equally to the total failure rate).
(5) At the start of testing, there is some finite total number of faults, which may be fixed or random; if random, their distribution may be known or of known form with unknown parameters.
(6) Between failures occurrence, the failure rate follows a known functional form.

3.2 Black-box simulation

For the black-box simulation, we treat software as a whole as only its interactions with the outside world are modeled, while the internal structure and component combinations are not concerned. This is relatively a simple simulation approach.

The input to the black-box simulation is a failure behavior file. This file includes the parameters of failure rate functions. The parameters can be obtained by using CASRE (Computer Aided Software Reliability Estimation) which is a tool for software reliability measurement [6]. The output or results of black-box simulation are the number of cumulative failures and the failure intensity of the software.

3.3 White-box simulation

In the black-box simulation, the software system is treated as a whole. The internal structure and features of software (e.g. the components

correlation) are not concerned. There are some shortcomings in this approach for software reliability measurements analysis. On the one hand, we must perform modeling based on availability of the whole system data, without using the unit testing data which is usually available earlier for each component. On the other hand, only one model can be applied to the simulation process; however, it maybe more appropriate that different components be applied different models. White-box simulations can remove these disadvantages. In order to have a more accurate simulation for software reliability measures, we developed two white-box simulation approaches. In the first approach, we use a variable as the correlation coefficient between components of software system. This coefficient can be calculated from testing data of each component. In the second approach we take into account the transition probabilities among components of a software system. Actually, the dependency or correlation are mainly caused by the existing transition between components.

3.3.1 Using dependency coefficient

In this kind of simulation, the basic idea is similar to black-box simulation. We also treat the software as a whole and the failure event producing algorithm is same as the black-box simulation, however, a dependency or correlation coefficient is introduced into the simulation process. We think some failures (or faults) in a component have some relation with another component or other several components. It means that removing some faults from a component may prevent two or several failures in different components. Therefore, we can use all the observed data of each component to simulate the whole software system reliability measures (e.g. cumulative failures) then adjusting the results by the dependency coefficient (we call this dependency coefficient as p). The key things are deciding p and adjusting policy. In practical, for deciding p is basically based on observed data sets of each component. A simple calculation method for p is: ND/NT, where ND is the total numbers of reported failures which lead to the correction of more than one component; NT is the total number of reported failures of all components. Table 1 illustrate the failures reports and the number of dependency failures in three software systems , we call them sys1, sys2 and sys3.

Software System	Failures Reports	Sum of failures of all components
sys1	380	440
sys2	143	161
sys3	51	58

Table 1. Number of failures and corrected faults in sys1, sys2, and sys3

For sys1 NT = 440, ND = 440-380=60, p = 60/440≈0.136. The p for sys2 and sys3 are 0.126, 0.137 respectively. Because it is easy to get the sum of failures in all components (NT), in practical, we can use it to obtain the input parameters for simulation, then adjusting the results by p. Adjusting policy is to change the cumulative failures number at each time point in whole simulation process. We do it use the equation: $CUM-CUM\times p$, CUM is the cumulative number of failures at each time point produced in simulation process. There are 27 months, 32 months, 45 months observed data for sys1 sys2 and sys3 respectively. We simulated 47 months, Table 2 shows the comparison of this simulation with the sum of failures in all components. From Table 2 we can see that the sum of failures in all components is larger than the system observed data (real data), however, the simulate results are close to the observed data. Table 2 has validated the concept of using dependency coefficient. In practical project, we usually get the failure number of each component first and easy, then using experienced dependency coefficient we can have a good prediction for software system reliability.

3.3.2 Using transition probabilities

This is another kind method to consider the dependency among components in simulation process. We think the dependency are caused by the transition from one component to another component during execution of program. In simulation, the transition probabilities can be assigned in advance, after producing failure event the transition probabilities are used to decide which component will be executed.

In this kind of white-box simulation, it is necessary to decide which component should be executed after each step. For this decision, the transition probabilities between components should be used. The input of this simulation are failure behavior of all components and transition probabilities file. Table 3 shows a four-component software failure behavior file

Component ID	Model	Parameter 1	Parameter 2
1	GO	130.6	0.0048
2	GO	108.7	0.0053
3	JM	63.78	0.3288
4	S-shaped	88.5	0.00988

Table 3. A four-component software failure behavior file

In Table 3, each row corresponds one component (first row is for component 1, second row is for component 2,etc.). First column indicate which model will be used for the component (e.g. "GO" represent the Goul-Okumoto model, "S-shaped" indicate the Yamada S-shaped model is being used for component 4). The other real numbers of each row are the parameters of the used software reliability model. These parameters can be estimated by using CASRE. The Table 4 shows a four-component software transition probability file

Component ID	1	2	3	4
1	0.00	0.80	0.00	0.20
2	0.00	0.00	0.70	0.30
3	0.00	0.30	0.00	0.70
4	0.80	0.00	0.20	0.00

Table 4. A four-component software transition probability file

In Table 4, the first row and firs column are integer numbers, they indicate the components ID number. Each real number in the table represents the transition probability from component i to component j.

The output or results of this simulation are number of cumulative failures for each component and the whole software. Using this kind of simulation we can apply different model to different component and we can get the software reliability measures for both the whole system and each component. However, in practical application, it is difficult to decide the transition probability for a component-based software system. In our simulation process for real project data, the transition probabilities are randomly assigned.

We also use the three software systems sys1, sys2 and sys3 as example to illustrate this approach. For simple reason, we just apply GO model to the three software system in simulation. Table 5 shows the comparison of this simulation with the sum of failures in all components. From Table 2 and Table 5 we can see that the results of these two kinds of white-box simulation have better prediction for the software systems than just adding the failure number in all components.

4. Project application

We have applied the two white-box simulation approaches into a project software for analyzing its reliability features. This section introduce the application results and some comparisons.

4.1. General description of the software

This is the system software of three successive generations of the Brazilian switching system, TROPICO-R [9,10]. It is developed jointly by the R&D center for Brazilian Teleco-mmunications and some Brazilian manufacturers. To dates, three successive products have been developed, and referred to as PRA, PRB and PRC. The software can be decomposed into two main parts; the applicative software and the executive software. Two categories of components can be distinguished in the TROPICO-R software: i) Elementary Implementation Blocks (EIB), which fulfil elementary functions and ii) groups of elementary implementation blocks according to the main four functions of the system. We think PRA, PRB and PRC are software systems consists of four components, and their reliability measurements can be simulated by the two approaches described above.

4.2 Simulate results and comparisons

We have applied our scheme to simulate the software reliability of three successive generations products (TROPICO-R, PRA, PRB and PRC). First , we simulated each function of each product, then we made simulations for each product. Three models are applied in these simulation processes, they are: GO model, JM model and Yamada S-shaped model. The results of simulations are cumulative number of failures for each function component and whole system. Here, we just give the simulation results of system for each product. Figure 1 and Figure 2 show the results of simulation and comparisons in using coefficient and transition probabilities respectively for the three products.

Figure 1 and Figure 2 demonstrate the two types of white-box simulation methods can provide a good prediction for software reliability measures in some software projects. Although, some

	Actual system failures	Prediction by sum of failures in components		Prediction by simulation		Improvement factor
		Total component failures	Error percentage	Simulated system failures	Error percentage	
sys1	380	410	7.89%	385	1.32%	5.98
sys2	143	161	12.59%	150	4.9%	2.57
sys3	51	58	13.73%	53	3.92%	3.5
Avarage	191.33	209.67	11.4%	196	3.38%	4.02

Table2. Comparison for simulation and sum of failures in all components (Using coefficient)

	Actual system failures	Prediction by sum of failures in components		Prediction by simulation		Improvement factor
		Total component failures	Error percentage	Simulated system failures	Error percentage	
sys1	380	410	7.89%	377	0.79%	9.98
sys2	143	161	12.59%	145	1.4%	8.99
sys3	51	58	13.73%	54	5.88%	2.33
Avarage	191.33	209.67	11.4%	192	2.69%	7.10

Table 5. Comparison for simulation and sum of failures in all components (Using transition probabilities)

component-based reliability models have been proposed [11] and there are some research about failure correlation in software reliability models [12], in practical projects, using simulation methods are easier and more operational especially for considering the failure correlation of components. From the simulation results we also get some evaluation for software reliability measures. For PRA system, GO model and JM model have better fitting than S-shaped model during early phase. This may be explained as: PRA is the first generation product, there was no inherited experience for software developers and testers. Therefore, the faults have more homogeneous exposure rate during testing phase. For PRB and PRC systems the S-shaped model has better fitting during early phase, it can be thought that in successive generations software, latent faults are more difficult (take more time) to be found. In Figure 1 and Figure 2 the simulation results curves and observed data curves have similar trends, however, with taking account into the time, it is difficult to have accurate failure evaluation or prediction with exact time point. In other words, for a random failure process, simulation methods can give a trend or general prediction, and it can not give the accurate measures with exact occurrence time.

In order to evaluate the accuracy of simulation, we calculated the MSE (Mean Square Error) for both simulation approaches and every model.

$$MSE=\frac{\sum_{i=1}^{n}(S_i-R_i)^2}{n}$$, in which, S_i is the simulate result at that time point, R_i is the real data at that time point, n is the total number of observed data. Table 6 and Table 7 give the results of MSE of every model for the two simulation approaches.

Model	MSE (PRA)	MSE (PRB)	MSE (PRC)
GO	322.98	166.93	15.23
JM	305.53	288.22	32.23
S-shape	1810	69.31	11.87

Table 6. MSE for the simulations (using dependency coefficient)

Model	MSE (PRA)	MSE (PRB)	MSE (PRC)
GO	327.48	215.71	8
JM	305.53	371	8.25
S-shape	360.55	57	7.12

Table 7. MSE for the simulations (using transition probabilities)

From the Table 6 and Table 7, we know that for few number of failures system S-shaped model is more appropriate (both MSE values of S-shaped for PRC are least). Comparing the two white-box simulation approaches, for S-shaped model using transition probability is much better than using dependency coefficient, for GO model the results of the two approaches are close, for JM model using dependency coefficient is better than using transition probabilities. While we applying the simulation to PRA, PRB and PRC, because we know little about the internal architecture of these software systems, the transition probabilities are randomly assigned. However, we think the correlation and dependency of software components are mainly caused by the transition between components, if knowing more information about the software internal structure it is possible to get more accurate simulation results using transition probabilities.

5. Conclusions

In our work, we combined analytical models with simulation approaches to give effective and practical method for software reliability measures. The main contribution of our work is: the design and implementation of a set of rate-based simulation techniques. The advantages of the simulation are: It is not computation complexity; it enables models combination approach; It provides two methods to take into account the correlation or dependency between components, so we can treat a software system as a combination of some correlative components. This is more appropriate for analyzing reliability measurements of component-based software system. The project application demonstrates that it can be used for analysis, prediction and evaluation in software reliability literature.

Acknowledgement

The work described in this paper was supported by two grants from the Research Grant Council of the Hong Kong Special Administrative Region: France / Hong Kong Joint Research Scheme (1999-2001), and Earmark Grant Project No. CUHK 4193/00E.

References

[1] Michael R. Lyu, *Handbook of Software Reliability Engineering,* McGraw-Hill, New York, 1996.

[2] R.C. Cheung. "A User-Oriented Software Reliability Model". *IEEE Trans. On Software Engineering,* SE-6(2): 118-112, March 1980

[3] S. Gokhale, T. Philip, and P. N. Marinos. "A Non-Homogeneous Markov Software Reliability Model with Imperfect Repair". In *Proc. Intl. Performance and Dependability Symposium (IPDS '96),* pages 262-270, Urbana-Champaign, IL, September 1996.

[4] S. Gokhale and K. S. Trivedi. "Structure-based Software Reliability Prediction". In *Proc. of Fifth Intl. Conference on Advanced Computing (ADCOMP '97),* pages 447-452, Chennai, India, December 1997.

[5] S. Gokhale, Michael R. Lyu, and K.S. Trivedi. "Reliability Simulation of Fault-Tolerant Software and Systems". In *Proc. Of Pacific Rim International Symposium on Fault-Tolerant Systems (PRFTS '97),* Page 167-173, Taipei, Taiwan, December 1997.

[6] Musa, J.D., and Okumoto, K., "A Logarithmic Poisson Execution Time Model for Software Reliability Measurement," *Proceedings seventh International Conference on Software Engineering,* Orlando, Florida, 1984, pp. 230-238.

[7] Defamie, Patrick Jacobs, Jacques Thollem. "Software Reliability: Assump-tions, Realities and Data," Software Maintenance, 1999. (ICSM '99). Proceedings. IEEE International Conference on, 1999, Page(s): 337 -345

[8] S. Dalal, M.R. Lyu, and C. Mallow, "Software Reliability," Chapter in Encyclopedia on Biostatistics, P. Armitage and T. Colton (eds.), vol. 5, Wiley 1998, pp. 4550-4555.

[9] Karama Kanoun, Marta Rettelbusch de Martini, and Jorge Moreira de Souza. "A Method for Software Reliability Analysis and Prediction Application to the TROPICO-R Switching System," *IEEE Trans. on Software Engineering,* vol.17, no. 4, 1991.

[10] M. Kaaniche, K. Kanoun, M. Cukier, and M. Bastos Martini. "Software Reliability Analysis of Three Successive Generations of a Switching System," *Proceedings of first European Dependable Computing Conference (EDDCC-1),* Berlin, Germany, 1994, pp. 473-490.

[11] Wen-Li Wang, Ye Wu, and Mei-Hwa Chen. "An Architecture-Based Software Reliability Model," *Dependable Computing, 1999.*

Proceedings. 1999 Pacific Rim International Symposium on Dependable Computing , 1999, Page(s): 143 -150

[12] Goseva-Popstojanova, K.; Trivedi, K. "Failure Correlation in Software Reliability models" *Software Reliability Engineering, 1999. Proceedings. 10th International Symposium on, 1999* , Page(s): 232 -241

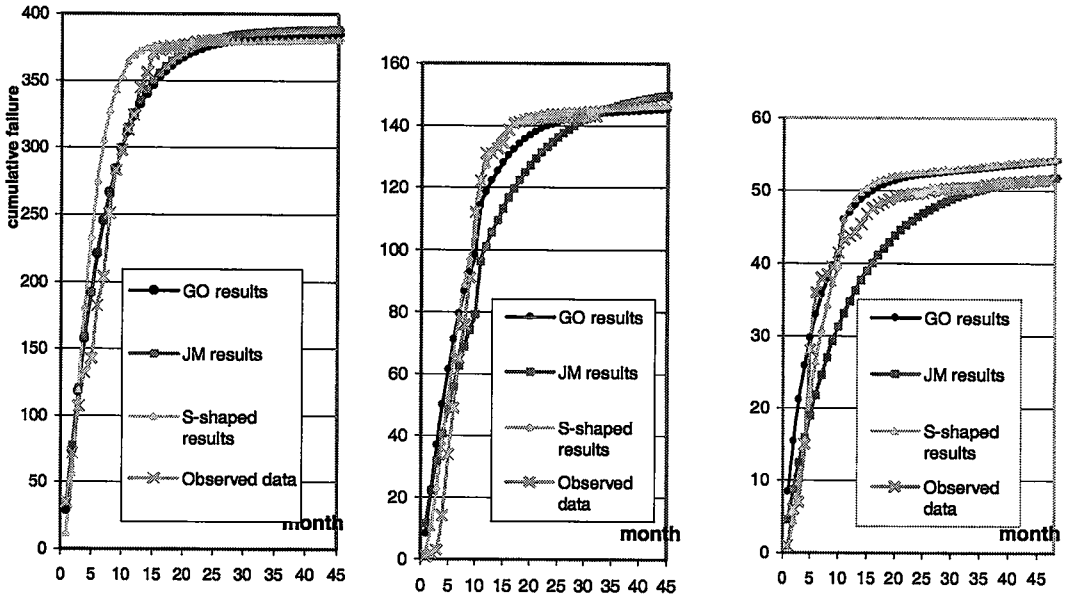

Figure 1. Simulate results for PRA, PRB, PRC (using dependency coefficient)

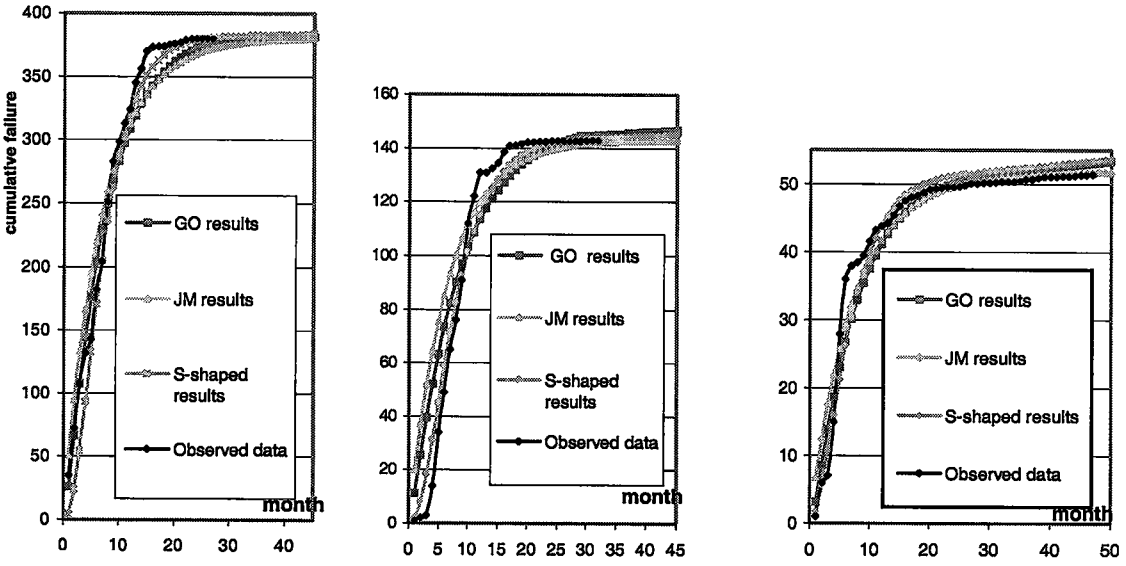

Figure 2. Simulate results for PRA, PRB, PRC (using transition probabilities)