In *Proc. of Workshop on Application-Specific Software Engineering and Technology (ASSET)*'98 (invited paper), Richardson, TX

Dependency Characterization in Path-Based Approaches to Architecture-Based Software Reliability Prediction*

Swapna S. Gokhale and Kishor S. Trivedi
CACC, Dept. of Electrical and Computer Engineering
Box 90291, Duke University, Durham, NC 27708-0291
Email: {ssg,kst}@ee.duke.edu

1 Introduction

Prevalent approaches to software reliability modeling are black-box based [1], i.e., the software system is considered as a whole and only its interactions with the outside world are modeled, without looking into its internal structure. Several critiques of these time-domain approaches have appeared in the literature [3] and some of these include the fact that they are applicable very late in the life-cycle of the software, ignore information about testing and reliabilities of the components of which the software is made, and do not take into consideration the architecture of the software. With the advancement and widespread use of object oriented systems design and development, the use of component-based software development is on the rise. The software components can be commercially available off the shelf (COTS), developed in-house, or developed contractually. Thus, the whole application is developed in a heterogeneous (multiple teams in different environments) fashion, and hence it may be inappropriate to model the overall failure process of such applications using only one of the several software reliability growth models (black-box approach). Modern programs can no longer be treated as monolithic entities, but are likely to be made up of thousands or millions of little parts distributed globally, executing whenever called, acting as parts of one or more complex systems[6]. Thus predicting reliability of the application early in the life-cycle, taking into account the information about its architecture, testing and reliabilities of its components, is absolutely essential.

The approaches to architecture-based prediction fall broadly into two categories: *state-based approaches* [2], and *path-based approaches* [4]. The path-based approaches to architecture-based software reliability prediction generally assume that the successive executions of the components are independent. This assumption leads to very pessimistic estimates of reliability, and largely impedes the applicability of these approaches. In this paper we propose a solution to the dependency characterization issue facing the path based approaches to architecture-based reliability prediction. We first describe the time-dependent view of software reliability in Section 2, outline the path based approaches in brief along with the dependency characterization issue in Section 3, and present the solution in Section 4. Section 5 concludes the paper.

*Supported by a contract from Charles Stark Draper Laboratory and in part by Bellcore as a core project in the Center for Advanced Computing and Communication

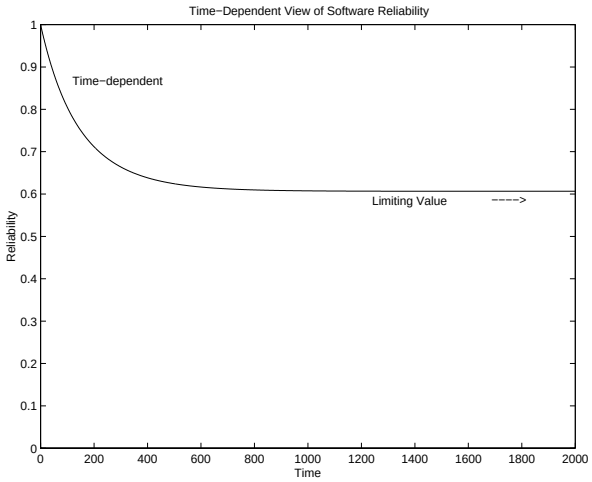

Figure 1:

2 Time-dependent View of Software Reliability

Most common notion of reliability of software is the ratio of all inputs completing execution successfully to the total number of inputs. The difficulty in using this "static" notion of reliability is that rarely, if ever, all possible inputs are applied in practice. If we then consider the above ratio based on the fraction of all inputs actually applied, we may consider the following model. This notion has been discussed exhaustively in [5]. Suppose the software is executed on a fixed number of inputs, say $N_0(t)$, where t denotes the time taken to execute the inputs. Let $N_s(t)$ denote the number of inputs which complete execution successfully, and $N_f(t)$ denote the number of inputs which cause the software to fail, such that $N_s(t) + N_f(t) = N_0(t)$. As per the frequentists approach, the reliability of the software denoted by $R_f(t)$ is then given by:

$$R_f(t) = \frac{N_s(t)}{N_0(t)} = \frac{N_0(t) - N_f(t)}{N_0(t)} = 1 - \frac{N_f(t)}{N_0(t)} \tag{1}$$

In order for $R_f(t)$ to represent the actual reliability of the software, we need to execute the software using all possible values from its input domain. In most practical applications, this implies that the software needs to be executed on a infinite number of inputs. As $N_0(t)$ increases, the estimate of $R_f(t)$ decreases, and it finally settles to a limiting value as $t \to \infty$. Thus, as shown in the Figure 1, the conventional notion of a fixed value of reliability, is nothing but time-dependent value in the limit.

3 Dependency Characterization – Issue

In this section, we briefly outline the path-based approaches to architecture-based prediction, and discuss the dependency characterization issue. Given the architecture of the software in terms of the transition probabilities among its components, and the reliabilities of the individual components and the interconnections between the components, the path-based approaches consist of running the software for various inputs. For each run, the execution path is specified in terms of the sequence of components. If all the components along this path are considered independent, then the product of the reliabilities of the individual components and interconnections between them is the path reliability. A number of such individual path reliabilities are computed, and an average of all such path reliabilities gives an estimate of the reliability of the software. If the application code is available, then the execution paths are determined by executing the application

against test inputs. If the application code is unavailable, then a simulation model of the software is developed, and is run using the discrete event simulation approach to determine the paths that are likely to be executed when the software is exercised. Thus, if an execution path consists of n components, viz., $1, \ldots, n$, with individual reliabilities denoted by R_1, ..., R_n, then the reliability of the path, denoted by R_p, assuming perfect interconnections among the components is given by:

$$R_p = R_1 \ldots R_n \tag{2}$$

A major drawback of both the execution as well as the simulation approaches is the assumption of independence among the successive executions of the components, which leads to very pessimistic estimates. This is illustrated with the help of an example as follows: Consider an execution path where component 2 is invoked inside a loop m times. Then assuming independence among all the m executions, an estimate of the equivalent reliability of component 2 denoted by $R_{2,eq}$ is ${R_2}^m$. This estimate of $R_{2,eq}$ is far too pessimistic for large m, due to which the estimate of the reliability of the software obtained using this methodology is also pessimistic. If most paths executed have components within loops, and that these loops are traversed a sufficiently large number of times, individual path reliabilities will tend to be low resulting in low software reliability despite the possibility that the reliability is actually higher. This leads to the problem of modeling the dependencies among the components.

4 Dependency Characterization - Solution

A possible way to overcome the problem of a pessimistic estimate is to treat the multiple executions as a single execution [4]. In this section, we propose a solution to the problem of pessimistic prediction of the reliability of a component in case of repeated executions based on the failure intensity of the component. Suppose that the time-dependent failure intensity of the component 2 is given by $\lambda_2(t)$, the component is executed m times, and the time spent in the component per visit is ϕ, then assuming that the successive executions are independent, the equivalent reliability of the component 2, denoted by $R_{2,ind}$ is given by [5]:

$$R_{2,ind} = \left(e^{-\int_0^{\phi} \lambda(\tau) d\tau}\right)^m \tag{3}$$

which can be written as:

$$R_{2,ind} = e^{-m \int_0^{\phi} \lambda(\tau) d\tau} \tag{4}$$

The equivalent reliability of the application assuming single execution, denoted by $R_{2,sin}$ is given by [5]:

$$R_{2,sin} = e^{-\int_0^{\phi} \lambda(\tau) d\tau} \tag{5}$$

The "actual" equivalent reliability of the component, denoted by $R_{2,act}$ is given by:

$$R_{2,act} = e^{-\int_0^{m*\phi} \lambda(\tau) d\tau} \tag{6}$$

The equivalent reliability assuming independent executions is lower than the reliability assuming single execution, whereas the actual equivalent reliability is lower than the reliability assuming single execution.

This can be better understood with the help of an example. Consider a component whose failure behavior is captured by the failure intensity function $34.05 * 0.0057 * e^{(-0.0057*t)}$. We assume that during a particular run, the component is visited twice ($m = 2$), and 50 time units are spent in the component per visit. Thus a total of 100 time units is spent in the component during this particular execution. The failure intensities assuming independent executions, single execution, and the "actual" failure intensity are as shown in Figure 2. The "actual" estimate of the reliability of the component obtained using this approach is the realistic estimate.

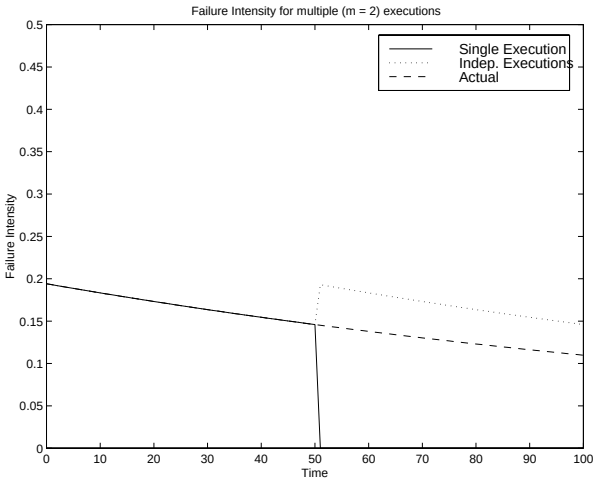

Figure 2:

5 Conclusions

In this paper we have presented a solution to the dependency characterization issue, which poses a major hurdle in the successful application of the path-based technique for architecture-based reliability prediction of the software. The solution relies on 1) time-dependent notion of reliability, and 2) time-dependent failure intensities of the individual components.

References

[1] W. Farr. *Handbook of Software Reliability Engineering, M. R. Lyu, Editor*, chapter Software Reliability Modeling Survey, pages 71–117. McGraw-Hill, New York, NY, 1996.

[2] S. Gokhale, , and K. S. Trivedi. "Structure-Based Software Reliability Prediction". In *Proc. of Fifth Intl. Conference on Advanced Computing*, Chennai, India, December 1997.

[3] J. R. Horgan and A. P. Mathur. *Handbook of Software Reliability Engineering, M. R. Lyu, Editor*, chapter Software Testing and Reliability, pages 531–566. McGraw-Hill, New York, NY, 1996.

[4] S. Krishnamurthy and A. P. Mathur. "On the Estimation of Reliability of a Software System Using Reliabilities of its Components". In *Proc. of Eighth Intl. Symosium on Software Reliability Engineering*, Albuquerque, New Mexico, November 1997.

[5] K. S. Trivedi. *"Probability and Statistics with Reliability, Queuing and Computer Science Applications"*. Prentice-Hall, Englewood Cliffs, New Jersey, 1982.

[6] J. Voas, A. Ghosh, G. McGraw, and K. Miller. "Glueing Together Software Components: How Good is Your Glue?". In *Proc. of Pacific Northwest Software Quality Conference*, Portland, OR, October 1996.