RELIABILITY ASSESSMENT OF COMPONENT BASED SOFTWARE SYSTEMS USING TEST SUITE - A REVIEW

R.Chinnaiyan
Research Scholar,
Department. of Mathematics,
Coimbatore Institute of Technology
Assistant Professor,
Department of Computer Applications,
AVC College of Engineering

Dr.S.Somasundaram
Assistant Professor, Department. of Mathematics,
Coimbatore Institute of Technology

Abstract

Software reliability has become one of the main issues for software developers. Aggregating components into software is a perfect approach to construct software with the maturity of component market. How to analyze software reliability from the reliabilities of its components and architecture should be answered. However, software in most of the proposed reliability analysis methods is static, while software development is a dynamic process, especially for component-based software, where pervasive process is iterative and incremental. Based on functional abstractions, this paper presents a general model for estimating the reliability of Component Based Software Systems using Role's of Components with Test Suite

Keywords: Software Reliability, Components, Component Based Software, Test Suite

I Introduction

IEEE 610.12-1990 defines reliability as "The ability of a system or component to perform its required functions under stated conditions for a specified period of time." IEEE 982.1-1988 defines Software Reliability Management as "The process of optimizing the reliability of software through a program that emphasizes software error prevention, fault detection and removal, and the use of measurements to maximize reliability in light of project constraints such as resources, schedule and performance." Software reliability is often defined as the probability of failure-free software operation for a specified period of time in a specified environment. Over the past 30 years, many software reliability growth models (SRGM) have been proposed for estimation of reliability growth of products during software development processes. Using these definitions, software reliability is comprised of three activities:

1. Error prevention
2. Fault detection and removal
3. Measurements to maximize reliability, specifically measures that support the first two Activities

Successful modeling has been done to predict error rates reliability.

These activities address the first and third aspects of reliability, identifying and removing faults so that the software works as expected with the specified reliability. These measurements have been successfully applied to software as well as hardware.

II Background

Several reliability models and estimation techniques have been proposed to assess the reliability of component-based applications. Gokhale et al. [3] discuss the flexibility offered by discrete-event simulation to analyze component-based applications. Their approach relies on random generation of faults in components using a programmatic procedure which returns the inter-failure arrival time of a given component. The total number of failures is calculated for the application under simulation, and its reliability is estimated. This approach assumes the existence of a control flow graph of a program. The simulation approach assumes failure and repair rates for components, and uses them to generate failures in executing the application. It also assumes constant execution time per component interaction, and ignores failures in component interfaces and links (transition reliabilities). Sanyal et al. [9] introduce Program Dependency Graphs and Fault Propagation Analysis [11], [12] for analytical reliability estimation of component based-applications. The approach is code-based (reverse-engineering) where dependency graphs are generated from source code, which may not be available for off-the-shelf components. Krishnamurthy et al. [5] assess the reliability of component- based applications using a technique called Component Based Reliability Estimation (CBRE). The approach is based on test information and test cases. For each test case, the execution path is identified. The path reliability is calculated using the reliability of the components assuming a series connection. This approach does not consider component interface faults, although they are considerable factors in reliability analysis of component-based software. This paper presents a general model for estimating the reliability of Component Based Software Systems using Role's of Components with Test Suite

III Proposed Model

The Component Based Software Systems Reliability measurement method (CBSSysRel) is a technique for making empirical measurements of the reliability of a software component. The technique does not necessarily require access to, or knowledge of, the source code; however, the purpose of CBSSysRel is to aid the component developer in providing information to potential acquirers of the component. We assume the public specification of the component, as exposed through the interfaces, i.e. ports, that it implements, identifies the various roles with which the component is intended to be compatible. Each role is implemented by a group of methods. CBSSysRel uses "role" as the unit for which reliability measurements are made. The definition of each role is used to create an operational profile. The role's operational profile is then used to create a set of test cases that are used to measure the reliability.

IV Probability for Successful Execution

The probability of successful execution is measured by repeatedly operating a system according to the selected operational profile, i.e. selecting inputs according to the frequency constraints of the profile, for the specified unit of time. The reliability is computed by measuring the percentage of those executions that terminate successfully. For a system a reliability value is reported for each operational profile. If certain critical or heavily used profiles correspond to lower than acceptable reliability values, the system may be modified to improve those values. For a component that will be used in a variety of contexts, a large number of reliability values should be reported so that the architect can compute the effective reliability that will be observed in a particular system. For example, a component is provided that plays a role in each of five protocols. The reliability portion of a component datasheet contains the role reliabilities and abbreviated role descriptions as depicted in Figure 1.

Role	Role Description	Reliability
A	Provides Basic Computation	0.90
B	Provides Graphing	0.91
C	Provides Database Access	0.92
D	Provides Security	0.93
E	Provides Transaction	0.94

Figure 1: Example role reliability description

The effective reliability of the component in a particular deployment is the reliability that the architect will experience with a specific component used in a specific manner. It is computed by multiplying the relative frequency for a role, as anticipated by the architect, and the measured reliability of the component in that role. These combined values are summed to give the effective reliability of the component. Suppose the architect intends to use the first, third, and fifth roles with equal frequency. The architect's reliability analysis worksheet includes a computation that looks something like Figure 2 for each component defined in the architecture.

Role	Reliability	Relative Frequency	Contribution
A	0.90	0.333	0.30
B	0.91	0.0	0.0
C	0.92	0.333	0.31
D	0.93	0	0.0
E	0.94	0.333	0.31
Component Reliability			0.92

Figure 2: Example reliability analysis worksheet

This value is then fed into the reliability analysis for the system, which uses the topology of the components in the architecture to compute estimated system reliability.

V CBSSysRel Process

CBSSysRel measures and communicates reliability values for a component. In this section we present a detailed outline for applying CBSSysRel given a component and its documentation.

5.1 CBSSysRel Process

Step 1 : Create a structure for measurements
Step 2 : Identify the Roles of Components in the Software System
Step 3 : Create an operational profile for each role.
Step 4 : Build a reliability test suite for each role.
Step 5 : Apply each test suite in an appropriate Environment
Step 6 : Evaluate the results
Step 7 : Extend the test suite

Step 1: Create a structure for measurements

Establish the confidence level you wish to have in the accuracy of the reliability value. The typical value is 95% or higher. Note, this is not a reliability target. It is an expression of how certain we wish to be that the measured value, whatever it is, is reported accurately. We need to be very certain of the accuracy of the reliability of the client since it is a part of the infrastructure upon which other applications depend. We will choose a confidence level of 99%. The reliability value will be reported as an interval of values. This confidence interval is sufficiently large that we have the specified level of confidence, e.g. 99%, that the real reliability value is within the reported interval. Since the component acquirer will not care if the actual reliability is greater than believed, a one-tailed confidence interval is used.

Step 2: Identify the Roles of Components in the Software System

This identification comes from the documentation of the component. The designer may have listed the roles and identified the services that participate in those roles. The reliability test plan identifies each of the roles and for each role the services that implement the role. A review of the protocol definition identifies the state machine described earlier.

Step 3: Create an operational profile for each role.

The profile describes the relative frequency with which each service is used. The reliability test plan defines how often the test suite should invoke each service of the role. Each test case will be one complete cycle of the role. In that cycle some of the services may be invoked multiple times. For cases such as a transaction manager where some methods may be called an indefinite number of times, this becomes a quantity to vary from one run to another. A single test case for client includes establishing the connection, authenticating to the server and then applying one or more services such as telnet or ftp. From one run to another a test case could be varied by connecting to different servers and manipulating different types of files.

Step 4: Build a reliability test suite for each role.

For each role, a test script is created that obeys the constraints of the role. The constraints usually include the state transitions that are possible. Other constraints include the types for the parameters on each service. We assume no access to the source code so the analysis is limited to the public interface documented in the role description.

An analysis of each parameter type in the role description leads to a set of partitions of values from the type. Each partition contains values for which we believe the component will behave the same. The test suite includes test cases that (1) test each possible ordering of service invocations and that (2) adequately sample over the parameter partitions for each service. These two conditions provide a means of measuring how completely the test suite covers the component roles. If the initial test suite is not sufficient to achieve a reliability measure in which we have sufficient confidence, the test designer searches for additional orderings of invocations or for parameter values that are likely to exercise new sections of the component. The most likely source of new test cases is unique combinations of parameter values. A thorough analysis of the client and the protocol indicates that, since the client runs on top of standard TCP implementations, the test cases do not need to cover multiple platforms. Also the number of orderings of invocations is very limited since establishing the transport layer followed by authentication must occur in that order. The mixture of service invocations after the connection is established is the main source of variation among test cases.

Step 5: Apply each test suite in an appropriate environment

A test case will cover a complete cycle of the protocol. For example, if the protocol is a transaction, a test case would extend from transaction start to transaction commit or transaction rollback. A test run will be one complete execution of all the test cases in the test suite. Multiple test runs will be conducted where one run differs from another by varying input parameters. A reliability value is computed at the end of each test run. The confidence in this value is computed. The cycle of test runs terminates when an acceptable confidence interval is computed.

Step 6: Evaluate the results

Each test case is evaluated and marked as either passed or failed. Each failure case is evaluated to determine whether the environment produced the failure or whether the component is responsible. The reliability computation uses the number of success and the number of failures to compute the reliability.

Step 7: Extend the test suite

Once the analysis is completed, if the level of confidence has not been reached, additional test cases may be created and executed. The coverage criteria provide direction on how to effectively expand the test suite to cover additional ground.

VI Conclusion

This work is exploring how to provide useful information about reliability to acquirers of components. Rather than provide a single value for the entire component, we provide reliability information about each role that the component is intended to support. The acquirer can then compute the effective reliability they would experience given their intended use of the component. The intention is to provide accurate information about reliability in support of component commerce and prediction of assembly reliability.

VII References

[1] Allen, Robert and David Garlan. A Formal Basis for Architectural Connection, ACM Transactions on Software Engineering and Methodology, 1997.

[2] Cho, Il-Hyung and McGregor, John D.“Component Specification and Testing Interoperation of Components”, IASTED 3rd International Conference on Software Engineering and Applications, Oct.1999.

[3] S. Gokhale et al., “Reliability simulation of component-based software systems,” in Proc. 9th Int. Symp. Software Reliability Engineering (ISSRE’98), Paderborn, Germany, Nov. 1998, pp. 192–201.

[4] Hissam, Scott, Gabriel A. Moreno, Judith Staffod, Kurt C. Wallnau. “Packaging Predictable Assembly with Prediction- Enabled Component Technology,” Carnegie Mellon

University Software Engineering Institute, CMU/SEI-2001-TR-024, 2001.
[5] S.Krishnamurthy and A. P. Mathur, "On the estimation of reliability of a software system using reliabilities of its components," in Proc. 8th Int. Symp. Software Reliability Engineering (ISSRE'97), Albuquerque, New Mexico, Nov. 1997, pp. 146–155.
[6] Mason, D. "Probabilistic Analysis for Component Reliability Composition," Proceedings of the 5th ICSE Workshop on Component-Based Software Engineering, Orlando, Florida, May 2002).
[7] Musa, John. Software Reliability Engineering, New York, NY, McGraw-Hill, 1998.
[8] "Perameterized Contracts for Adapter Synthesis," Heinz W. Schmidt and Ralf Reussner, Proceedings of the 5th ICSE Workshop on Component-Based Software Engineering, Orlando, Florida, May 2002.
[9] S. Sanyal et al., "Framework of a software reliability engineering tool," in Proc. IEEE High-Assurance Systems Engineering Workshop (HASE'97), Washington, DC, 1997, pp. 114–119.
[10] Selic, Bran and Jim Rumbaugh. Using UML for Modeling Complex Real-Time Systems, Rational Corp., 1998.
[11] Shah and S. Bhattacharya, "Fault propagation analysis based variable length checkpoint placement for fault tolerant parallel and distributed system," in Proc. 21st Annu. Int. Computer Software and Applications Conf. (COMPSAC'97), Bethesda, Maryland, Aug. 1997.
[12] J. Voas, "Error propagation analysis for COTS systems," IEEE Comput.Control Eng. J., vol. 8, no. 6, pp. 269–272, Dec. 1997.
[13] Stafford, Judith A. and McGregor, John D., "Issues in Predicting the Reliability of Components," Proceedings of the 5th ICSE Workshop on Component-Based Software Engineering, Orlando, Florida, May 2002.
[14] Szyperski, Clemens. Component Software: Beyond Object-Oriented Programming, Addison-Wesley, 1998.

AUTHOR'S BIOGRAPHY

R.Chinnaiyan is working as an Assistant Professor in the department of Computer Applications, A.V.C College of Engineering, Mannampandal, Mayiladuthurai. He is having 8 years of teaching experience. He is a life member of ISTE, CSI of INDIA. He is now doing his research in Anna University at Coimbatore Institute of Technology, Coimbatore. His research interest includes Software Reliability, Qos and Object Oriented Analysis and Design.

Dr.S.Somasundaram is working as an Assistant Professor in the department of Mathematics, Coimbatore Institute of Technology, Coimbatore. He is having 19 years of teaching experience. He had guided over 10 M.Phil Candidates in Bharathiyar, Bharathidasan and Annamalai Universities. Now he is guiding 5 Research Scholars under Anna University Chennai and Coimbatore, His research interest includes Reliability Engineering, Software Reliability, QoS and Networking.