

Measuring Component Reliability

John D. McGregor
Clemson University
johnmc@cs.clemson.edu

Judith A. Stafford
Tufts University
jas@cs.tufts.edu

Il-Hyung Cho
Clemson University
ihcho@cs.clemson.edu

Abstract

Much of the research on component-based software engineering assumes that each component has a single service. This simplifies analyses but it is a significant departure from actual components. This paper reports on an investigation of the feasibility of using design constructs as a means of treating several methods as a single unit. Grouping the services of a component into a few sets satisfies the goal of simplicity while still providing the designer with a more usable model of component reliability.

1.0 Introduction

The reliability of a software system expresses the probability that the system will be able to perform as expected when requested. This probability is interpreted in the context of an expected pattern of use and a specified time frame. A system's reliability might be reported as:

System X has a reliability of .99 when used in normal mode during an eight-hour period of operation.

Normal mode refers to a specific system mode that is defined by a pattern of inputs to the system. This may include specific selections from menus or certain characteristics of data in the input files.

Most components have multiple entry points just as most systems have multiple menu selections. It is more difficult to identify expected patterns of use for components than for systems, particularly for

components being developed for reuse across several systems.

The long-term goal of this research is to define a technique to provide component-level information to support prediction of assembly reliabilities based on properties of the components that form the assembly. The hypothesis of this specific investigation is that the role a component plays in a specific protocol is a useful unit upon which to base a reliability measure that can be used to calculate the reliability of the assembly. A protocol is a sequence of service invocations between a set of components, which accomplishes a specific task through the interaction of these components. We use *protocol* here in the sense that it is used in the real-time extension to UML [7], in the architecture description language Wright [1], and in our previous work [2]. We begin our definition of component reliability by measuring the ability of a component to perform its role in a protocol as expected.

A fundamental assumption in this work is that components are designed to satisfy roles in various standards or design patterns. We assume that no component is developed in the absence of some rationale for its existence. That rationale might be that the component is intended to play the role of server in a communication protocol or to transform data from the transport format into a computation format. The contribution of this paper is a method for measuring and communicating the reliability of a component in a way that is useful in describing components that are intended to be used by other developers. The method provides a technique for decomposing the specification of the component into logical pieces about which it is possible to reason. The method then provides a context in which

the reliabilities of these logical pieces can be combined to match the complete use a developer defines for the component in a specific system.

In the next section we provide background on components and reliability sufficient to understand the investigation. Sections 3 and 4 define the method and provide an example application of its use respectively. We conclude with a look at how this approach fits into a compositional reliability prediction method. The problem of compositional reliability, which is wrought with many difficult problems [4,6,8], is not the subject of this report but rather a method for dealing with one of the underlying problems – how to describe component reliability information for use in a variety of settings.

2.0 Background

The work reported in this paper is part of an ongoing research project in compositional reliability analysis. Our research is based on the following understanding of the software components and software reliability.

2.1 Components

The work described in this paper conforms to the standard properties of a *component* defined by Szyperski[10]:

- ?? unit of independent deployment,
- ?? unit of third-party composition,
- ?? no persistent state,

but the reliability measurement method is not sensitive to variations in these properties. We assume that a component supports multiple public services that have been specified in one or more interfaces. Each interface describes a set of services that relate to some common purpose such as being a participant in a design pattern.

A component is connected to other components in conformance to an architecture, which can be described in an architecture description language such as Wright [1], in which connectors are first class objects. A Wright description includes a definition of a protocol and contains a set of role definitions needed for the protocol as shown in Figure 1. Each component defines a set of ports that identify the roles the component implements. Any component that implements a role has the potential to be attached to the role via a specific port. The protocol is characteristic of the design such as a client/server protocol.

Both the port and role descriptions provide sufficient information to check the compatibility of a specific port and specific role. A port of a component is compatible with a role if the port can be substituted for the role with

no noticeable difference in behavior. In addition to the usual type compatibility notion, a protocol usually imposes a partially-ordered sequence on the service invocations provided by the roles involved in the protocol.

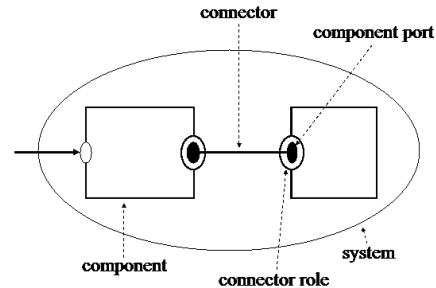


Figure 1

2.2 Reliability

The reliability of a software product is usually defined to be “the probability of execution without failure for some specified interval of natural units or time”[5]. This is an operational measure that varies with how the product is used. Reliability of a component is measured in the context of how the component will be used. That context is described in an operational profile.

Reliability of a piece of software may be computed or measured. If the software has not been built yet, its reliability can be computed from a structural model and the reliabilities of the individual parts that will be composed to form the software. There is error associated with this technique due to emergent behaviors, i.e. interaction effects, which arise when the components must work together.

If the software is already assembled, the reliability can be measured directly. The measurement is taken by repeatedly executing the software over a range of inputs, as guided by the operational profile. The test results for the range of values are used to compute the probability of successful execution for a specific value. The error associated with this approach arises from the degree to which the actual operation deviates from the hypothesized operation assumed in the operational profile.

3.0 CoRe Method

The *Component Reliability* measurement method (CoRe) 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 CoRe 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. CoRe 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.

3.1 Two types of confidence

The use of a confidence interval is intended to measure how certain we are about the measurement we have made. It does not necessarily describe how confident we are that we have measured the correct quantity. For example, we might choose parameters for three test cases, all of which result in the same ten lines of code being executed. Since the same lines of code are executed each time, there will be very close agreement among the three measures. As a result, our confidence in the accuracy of the measurement is high even though choosing other parameter values might have led to a very different result; the variance among the measurements, or the lack thereof, which gives confidence.

The operational profile for a role gives confidence that the reliability test suite will measure the correct quantity. The more closely the test suite resembles the actual use of the functionality described in the role, the more accurate the resulting reliability value is. If portions of the software are not executed, it is because the role does not use that software and it is not pertinent to the measurement.

CoRe uses both an operational profile for the unit being measured and a confidence interval about the measured value to report the reliability for a component role. In this way we have a high confidence that the measurement is an accurate value for the correct quantity.

3.2 Method Definition

The definition of reliability given in the previous section has three significant parts:

1. A probability
2. An operational profile that defines context

3. Duration of operation defined in some natural units

Next we consider the parts of the definition in reverse order and follow this with its application to an example component, an SSH client. This component can be configured to act as an SSH1 client or an SSH2 client. We selected this component because the SSH state machines are well-defined and the protocols between clients and servers are clearly specified.

3.2.1 Natural units

The reliability of a complete system is usually defined in terms of execution time. The duration of a reliability test run is then chosen to correspond to some time period of interest, such as an eight-hour workday. Further, the execution time can be translated into calendar time if that is a more meaningful level for the analysis. This can be used to report reliability values for a system that has a natural period of a week’s worth, a month’s worth or even a year’s worth of operation.

For an individual component, the duration of execution of a specific service may be very brief. Some components might only execute for a very brief time during a lengthy system execution time. Associating a duration of eight hours with a component that only executed a few milliseconds during that period is not representative. When the component is placed in a system that invokes it more often, its contribution to the system reliability may vary markedly from that observed in the original system.

Although the reliability of an individual component could be expressed in terms of number of invocations of individual services, this seems too fine-grained to be useful. Architects are unlikely to know exactly how many times a service of a component will be invoked and thus would not carry out their analysis of a system at that level. CoRe reports reliability values based on the probability that a component will execute its role in a protocol successfully. For example, our SSH client might require multiple invocations of the decryption service of the SSH component, but the architect is concerned with whether a file transfer is completed or not. The exact number of invocations of the decryption algorithm is not important to the analysis of the overall reliability.

3.2.2 Operational profile

Reliability is an operational concept. A piece of software’s reliability value will vary depending upon how that software is used. Suppose a component has five services, four of which always succeed and one of which always fails. If all services are used at an equal rate, the system is 80% reliable over an aggregate of many uses.

But if the service that always fails is never used, the system is 100% reliable.

An operational profile describes the frequency with which a service is expected to be invoked. The typical component's public set of services is large and diverse. Some applications may never use some of the services while those same services may be heavily used when the component is deployed a different application. In CoRe, an operational profile is created for each protocol role in which the component is designed to participate. We assume that the services for a particular role are contained in interfaces implemented by the component. The SSH Client component implements separate interfaces for SSH1 and SSH2 types of connections.

A state machine and a set of frequency tables are used to describe the operational profile for a role. The state machine defines the sequence in which the services of the component can be invoked. For the SSH client, the basic state machine defines a sequence of (1) establish transport layer; (2) authenticate client to server; and (3) connect to desired services.

A frequency table is used to capture the relative frequency with which each service is invoked in one complete cycle of the protocol. There may be multiple frequency tables particularly if there are multiple paths through the state machine. Even a single path may have cycles that permit multiple frequency values for some services. For the SSH client, the transport layer is established once per session. Authentication occurs once per client login. The connection is used an indefinite number of times for a variety of services. This last action has the greatest impact on reliability since it is the action taken most frequently.

A system may have many operational profiles corresponding to different users and it may have many profiles for one user if there are different modes of the system. Similarly, a component may be able to play many roles in many protocols and thus an operational profile is created for each role. CoRe reports a reliability value for each role. It is possible to identify users who only use the SSH client for SSH1 connections, others only for SSH2 connections, and others who use both. It is also possible to identify users who primarily use the SSH client for single operations in which case all three services are used with the same frequency and those users who maintain a connection for a long time using several services with a single transport/authentication action.

3.2.3 Probability of 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 2. (This assumes that role descriptions are also available as part of the component documentation.)

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 2: 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 3 for each component defined in the architecture.

Role	Reliability	Relative Frequency	Contribution
A	.9	0.333	0.30
B	.91	0.0	0.0
C	.92	0.333	0.31
D	.93	0	0.0
E	.94	0.333	0.31
Component Reliability			0.92

Figure 3: 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.

4.0 CoRe Process

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

Step 1: Establish the context 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 SSH client's reliability 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 component's roles

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. The documentation for the SSH client states that the client provides a "complete" implementation of the SSH1 and SSH2 protocols. A review of the protocol definition identifies the state machine described earlier.

Step 3: Establish 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 the SSH 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: Construct 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 SSH client and the SSH 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: Analyze 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: Expand 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 given in step 4.3 provide direction on how to effectively expand the test suite to cover additional ground.

5.0 Future Work

The next step in the maturation of CoRe is to complete a set of experiments in which the reliability of actual components are measured. Initial experiments have been conducted to prove the concept. We have been able to develop initial test suites and exercise each of the roles. We have not yet stressed either protocol to show less than 100% reliability. More comprehensive experiments will explore this component and a variety of other components.

The CoRe method is a fundamental element in an effort to construct a Prediction-Enabled Component Technology (PECT) that predicts the reliability of a component assembly from the reliabilities of the individual components [3]. The Reliability PECT works with components that have been evaluated using the CoRe method. The PECT provides an analysis technique that models the proposed assembly, selects the appropriate roles and corresponding reliability values for each component used in the assembly, and predicts the reliability value of the assembly.

6.0 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.

7.0 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] Hissam, Scott, Gabriel A. Moreno, Judith Stafford, Kurt C. Wallnau. "Packaging Predictable Assembly with Prediction-Enabled Component Technology," Carnegie Mellon University Software Engineering Institute, CMU/SEI-2001-TR-024, 2001.
- [4] Mason, D. "[Probabilistic Analysis for Component Reliability Composition](#)," Proceedings of the 5th ICSE Workshop on Component-Based Software Engineering, Orlando, Florida, May 2002).
- [5] Musa, John. *Software Reliability Engineering*, New York, NY, McGraw-Hill, 1998.
- [6] "Parameterized 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.
- [7] Selic, Bran and Jim Rumbaugh. *Using UML for Modeling Complex Real-Time Systems*, Rational Corp., 1998.
- [8] 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.
- [9] Szyperski, Clemens. *Component Software: Beyond Object-Oriented Programming*, Addison-Wesley, 1998.