Estimating Software Component Reliability by Leveraging Architectural Models

Roshanak Roshandel\textsuperscript{1,2}, Somo Banerjee\textsuperscript{2}, Leslie Cheung\textsuperscript{2}, Nenad Medvidovic\textsuperscript{2}, Leana Golubchik\textsuperscript{2,3}

\textsuperscript{1}Dept. of Comp. Science & Software Engr. Seattle University Seattle, WA 98122, USA roshanak@seattleu.edu
\textsuperscript{2}Computer Science Department University of Southern California Los Angeles, CA 90089, USA \{sbanerje,lccheung,neno,leana\} @usc.edu
\textsuperscript{3}EE-Systems Dept, IMSC University of Southern California Los Angeles, CA 90089, USA

ABSTRACT
Software reliability techniques are aimed at reducing or eliminating failures in software systems. Reliability in software systems is typically measured during or after system implementation. However, software engineering methodology lays stress on doing the “correct things” early on in the software development lifecycle in order to curb development and maintenance costs. In this paper, we propose a framework for reliability estimation of software components at the level of software architecture.

Categories and Subject Descriptors
D.2.4 [Software Engineering]: Software/Program Verification – Reliability.

General Terms
Measurement, Design, Reliability

Keywords
Software Architecture, Reliability, Components

1. INTRODUCTION

1.1. Overview of the Problem
Software reliability techniques are aimed at reducing or eliminating failures in software systems. Existing software reliability techniques are typically rooted in the field of reliability engineering, and particularly hardware reliability. They complement software testing and generally assume the availability of implementation artifacts. However, the conventional software engineering wisdom suggests that assessing reliability (or any other software quality) at system implementation-time may be too late. If problems are identified at this stage, the system might have to be redesigned and reimplemented, which is overly costly. Many critical design decisions about a system are made long before it is implemented. In this paper, we argue that software reliability should be assessed throughout a system’s life span.

It is widely recognized that the linchpin of the software development process is software architecture [6]. Software architectures provide high-level abstractions for representing the structure, behavior, and key properties of complex software systems [6]. Identifying and mitigating problems early in development can help to increase the quality of a system in a cost-effective manner. To achieve this goal, reliability and other quality attributes must be “built into” the software system during the architectural design phase. In this paper, we propose a framework for component reliability estimation at the software architecture level.

1.2. Motivation
The lack of execution artifacts is a major constraint when estimating component reliability at the architectural level. Hence, traditional reliability measurement techniques are not suitable in this context. To address this constraint, one viable approach can be to utilize the components’ architectural specifications at our disposal to construct a model that provides an early and meaningful reliability estimations of the components. In doing so we are presented with two major challenges: (1) how can we utilize architectural specifications of a component to construct an effective reliability estimation model?; and (2) how do we handle the uncertainties present in this model due to the lack of operational profile?

In this paper, we posit that architectural modeling and analysis can provide meaningful insights into a component’s structure and intended behavior and hence can be used as the building block of our reliability estimation framework. Modeling software components from different perspectives can provide complementary views of a component that can lead to sophisticated analyses of its structure, behavior, and non-functional properties. A software component is traditionally modeled from one or more of four functional perspectives: interface, static behavior, dynamic behavior, and interaction protocol [9]. The interface view of a component shows its provided and required services; the static behavior view shows the functionality of the component discretely (i.e., at different “snapshots” during the system’s execution), using invariants on the component states and pre- and post-conditions associated with the components’ operations; the dynamic behavior view shows a continuous view of the component’s internal execution details; and the interaction protocol view shows a continuous external view of a component’s execution by specifying the allowed execution traces of its operations (accessed via interfaces). In this work we leverage these architectural models in two important ways. Firstly, we use the dynamic behavior model as a basis for our estimation framework in the absence of the component’s operational profile. Secondly, we utilize view-level inconsistencies between the differ-
ent architectural models to obtain architectural errors (or defects). These defects are used to model the failure behavior of the component.

The motivation for the second challenge comes from a review of existing research in the area of software reliability estimation at the architectural level [2,8,11]. These approaches estimate system reliability and assume that component reliability, or the reliability of some of its elements (e.g., component’s services) is known. We do not believe this assumption to be reasonable and argue that component reliability can be estimated at the architectural level by leveraging architectural specifications. Another assumption that these approaches have made is that component behaviors exhibit the Markov property\(^1\), and have then used Markov models of component interactions as a basis for estimating system-level reliability. We also rely on this assumption, but use it as the basis of constructing a Markov model of interaction between individual states of a component in order to estimate component reliability. However, we are still faced with the challenge posed by the lack of operational profile, which shall result in unknown parameters in the above-mentioned Markov model. We consequently argue that a Hidden Markov model (HMM) [7] is a formalism that can estimate hidden/unknown parameters in a model, and hence is a suitable approach to handling the lack of an operational profile.

To summarize, our reliability estimation framework leverages the architectural models of a component and builds a Hidden Markov Model. The unknown parameters of this HMM are then estimated, and the resulting Markov model is used to estimate the reliability of the component. The contribution of this paper is a framework based on component’s architectural models that is used to generate a stochastic reliability model. This model provides an estimation of component’s reliability during the architectural design phase and before it is implemented.

1.3. Related Work

The central role in our approach to component reliability estimation is played by a collection of modeling views commonly used to represent a software system’s architecture. Our previous work has identified four functional views (called the Quartet) on which architectural specification predominantly focuses [9].

Modeling, estimating, and analyzing software reliability—during testing—is a discipline with over 30 years of history. Many reliability models have been proposed: Software Reliability Growth Models (SRGMs) are used to predict and estimate software reliability using statistical approaches [1,3,4,5]. The common theme across all of these approaches, however, is their applicability to implementation-level artifacts, and reliability estimation during testing. At the architectural level, existing reliability estimation approaches consider only the structure of the system. The only exceptions are [2,8,11,12]. However, none of these approaches consider the effect of a component’s internal behavior on its reliability. They simply assume that the component reliability, or some of its elements (such as reliability of component’s services) is known. They then use these values to obtain system reliability.

---

\(^1\) The Markov property dictates that the probability of transition between two states only depends on the current state, and is independent of the path by which the current state was reached or the amount of time spent in the current state.

---

2. COMPONENT RELIABILITY ESTIMATION

In this section, we present our reliability estimation framework, illustrated in Figure 1. The framework has three phases. The working example used in this paper is that of a CruiseControl component, whose dynamic behavior model is shown in Figure 2. The dynamic behavior model of a software component is often depicted by a state transition diagram that shows the internal states of a component, the transitions between them, and the event/action pairs as causes and effects of these transitions. The CruiseControl component has three states, stop (S\(_1\)), manual (S\(_2\)), and cruise (S\(_3\)).

As we have discussed in Section 1.2, HMMs provide an appropriate formalism to compensate for unknown parameters due to the lack of execution artifacts. An HMM is defined by a set of states \(S=\{S_1, S_2, \ldots, S_N\}\), a transition matrix \(A\) representing the probabilities of transitions between states, a set of observations \(O=\{O_1, O_2, \ldots, O_M\}\), and an observation probability matrix \(B=\{b_{ik}\}\), which represents the probability of observing event \(k\) given that we are in state \(i\). We now discuss the three phases of our framework, relating the discussion to this definition of HMMs.

**Phase 1: Architectural Modeling, Analysis, and Quantification.** In this phase, we apply standard analysis techniques to the architecture-level models of the component and identify architectural defects. Our multi-view approach to component’s architectural modeling results in architectural view inconsistencies that can
be leveraged to represent defects. Examples of such defects are a signature mismatch for the gas interface, or a mismatch at the level of pre-condition (static behavior) for the cruise interface.

Defects identified in this stage contribute to unreliability of the component. We leverage our previous work on defect classification [10] and use a cost-framework to quantify the architectural defects obtained from the analysis. The cost-framework uses a cost function to quantify defects based on the observations that different defects affect a component’s reliability differently, and that the cost associated with defect mitigation varies based on the type of defect. The values obtained from the defect quantification are used in the component reliability computation phase discussed below.

Phase 2: Operation Profile Modeling. Unknown operation profile of the component during architectural design induces uncertainty in the reliability estimation process. Our reliability modeling approach thus includes a stage in which, given available information about the component, we construct an HMM that directly leverages the component’s behavior model. In particular, we map the dynamic behavior model of the component to an HMM as follows: the states in the dynamic behavior model become corresponding states in the HMM (set \( S \)), and the event/ action pairs of the dynamic behavior model become observations of the HMM (set \( O \)).

Our HMM solver then leverages a set of synthesized or simulated training data and applies the well-known Baum-Welch algorithm [10] to estimate the unknown parameters. These parameters are transition probabilities that conceptually correspond to the component’s operation profile.

It is important to note here that defect quantification and training data generation are both “pluggable” modules for our framework. In other words, the method used for either of these two processes is independent of the framework itself.

Phase 3: Reliability Computation. Upon solving the HMM and estimating its unknown parameters, a Markov model is obtained. This model is then extended with a failure state \( F \) that represents the state of the component when an error occurs. We add a transition from each state in the model to the failure state, to designate erroneous behavior. The probabilities of the transitions to the failure state are obtained from the defect quantification step. We also add a single transition from the failure state to a designated state in the model, to represent failure recovery. We hypothesize that the probability of the recovery transition can be obtained from a human expert; we are currently investigating alternative ways of obtaining this value. At this point, standard techniques involving numerical methods are applied to estimate the reliability of the component in terms of the probability that it is not in the failure state during its execution.

Figure 3 shows the dynamic behavior model of the CruiseControl component with failure state \( F \) and the corresponding transitions. State \( S_f \) (stop) is designated to be the active state of the component once it recovers from a failure.

3. EVALUATION

Our preliminary results indicate that our model is suitable for identifying trends when estimating reliabilities of software components (e.g., comparing the reliability of various components). It is also capable of offering analyses aimed at identifying the impact of various factors on a component’s reliability. Therefore, we aim to evaluate our framework along two main dimensions: (1) validation of the approach against actual component reliability measurements, and (2) usefulness of the analyses enabled by the approach. We realize that the full validation of the approach is a longer term goal and requires availability of real-world systems with architectural models that are verifiably faithful to system implementations. Here we have thus focused on the second dimension—analyses enabled by our approach—and provide a couple of examples to illustrate its use.

Example 1. We want to identify how a particular state affects the overall component reliability. While Markov-based modeling of the system enables just such analysis, the integration with our defect classification and cost framework enables us to not only identify the critical components, but also identify the operations that have the largest influence on the component’s reliability. Such an analysis can be particularly useful for a component with a large number of states. Improving the reliabilities of the critical states can result in improvement in the reliability of the whole component. Figure 4(a) shows the reliability of the individual states of the CruiseControl component \( (R_i \text{ being the reliability of state } S_i) \) and

![Figure 4](image.png)

Figure 4. (a) Effect of state reliability on component reliability; (b) Effect of recovery probability on component reliability
the overall component reliability in three different cases. It is clear from the figure that states $S_1$ and $S_3$ are more critical than state $S_2$.

**Example 2.** We want to identify how variations in recovery probability affects the overall component reliability. In a simple experiment, we varied the recovery probability of the CruiseControl component from 0 to 1, keeping everything else constant. Figure 4(b) plots the obtained component reliability vs. recovery probability. This analysis is directly related when designing fault-tolerant components. If for instance we could increase the recovery probability of a particular defect, the impact on the overall reliability would be critical in identifying the defects whose recovery should be addressed first.

4. **CONCLUSIONS AND FUTURE WORK**

Our reliability estimation framework attempts to bridge the gap between architectural modeling and analyses, and software reliability measurement. We have focused on the estimation of component reliability as a first step in this direction.

We believe that Hidden Markov models (HMMs) provide an appropriate formalism for modeling the reliability of software components in the presence of uncertainties. Our framework is directly tied to state of the practice techniques in architectural specification, and leverages a defect quantification and cost framework to quantify failure behavior of a component. Our preliminary evaluation has indicated that the framework can perform well in presence of uncertainty.

Here, we present our ongoing work related to this research.

**Training data generation.** Predicting the intended operation profile of a component is not trivial. Recall that, in our framework, we have used synthesized training data. We are investigating into better ways of training data generation using the dynamic behavior model of a component, e.g., statechart simulation methods, and trace assertion method for module specification.

**System-level reliability.** In order to estimate system-level reliability, we are investigating into how we can construct a system-level model based on our component reliability models. We are aiming at building a system-level model that can be solved efficiently. Otherwise, the model becomes intractable for larger systems, and hence not useful.

**Validation through simulation.** We plan to evaluate our framework by comparing our models with real system simulations and emulations. One problem that we are trying to resolve is that in performance modeling it is accepted that simulation may be functionally different from the real system, while in the software engineering domain such a relationship between a model and its realization may not be acceptable.

**Application to real systems.** We also plan to evaluate our framework using already implemented software systems for accuracy of estimation. Web service-based systems are gaining more attention recently, and we are currently applying our technique to these systems.

**Failure and Recovery.** We are in the process of extending the reliability estimation framework to include multiple types of failure states. Each failure state relates to a defect category specified in the defect classification framework. Appropriate transitions to each failure state and a recovery transition from each failure state must be designated. The recovery behavior for different types of failures also may be generalized: recovery can cause a transition to any state of the component rather than to a specific recovery state as shown in the example in this paper. Currently, we are investigating these issues in more detail.

5. **ACKNOWLEDGEMENTS**

This work was supported by the NSF 0509539, 0312780, 0091474, and 9985441 grants. This work also made use of Integrated Media Systems Center Shared Facilities supported by the NSF under Cooperative Agreement No. EEC-9529152. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect those of the National Science Foundation. The authors also wish to thank the anonymous reviewers for their helpful comments.

6. **REFERENCES**


