A Classification of Architectural Reliability Models

Roshanak Roshandel
Computer Science & Software Engineering Dept.
Seattle University
Seattle, WA 98122, USA
roshanak@seattleu.edu

Nenad Medvidovic
Computer Science Dept.
University of Southern California
Los Angeles, CA 90089, USA
neno@usc.edu

Abstract

With the widespread use of software systems in the modern society, reliability of these systems have become as important as the functionality they provide. Building reliability into the software development process thus becomes critical for cost effective development and quality assurance. Existing reliability models (applied in post-implementation phases) may not be suitable to address reliability analysis at the software architecture level, as they often rely on implementation-level artifacts. In this paper, we present a framework for classifying reliability models based on their applicability to architectural artifacts, and assess several representative approaches based on the proposed classification. This study highlights several areas for future research.

1. Introduction and Motivation

With the prevalent use of complex software systems in people’s daily lives, reliability of these systems have become as critical as the functionality they provide. Ensuring the reliability of the system as part of integration or qualification testing (i.e., during testing, deployment, and maintenance) is no longer an acceptable or cost-effective approach to quality assurance. Reliability –like other dependability attributes– must be “built into” software systems, and doing so requires models to represent and further analyze dependability throughout the software development life cycle (SDLC). Our research has focused on reliability modeling and analysis of software architectures [28,29]. Analyzing software reliability during early stages of the SDLC poses special challenges. For example, the exact context in which the system will be used, as well as system’s operational profile may be unknown. Reliability models applicable to early stages of development thus, must be able to address the uncertainties associated with early reliability modeling.

Software reliability is defined as the probability that the system will perform its intended functionality under specified design limits [25]. Software reliability techniques are aimed at reducing or eliminating failures of software systems. Existing software reliability techniques are often rooted in the field of reliability engineering in general, and hardware reliability in particular. Such approaches provide significant experience in building reliability models, and advanced mathematical formalisms for analytical reasoning. However, they often do not sufficiently leverage available architecture-level knowledge about the system, and instead rely on implementation-level data to provide failure behavior.

A literature review in the area of software reliability modeling reveals that there is no easy way to determine whether a specific reliability model is suitable to be applied to architecture-level artifacts. More specifically, criteria to determine the extent to which a reliability model leverages architecture-level or implementation-level knowledge are unclear.

In this paper, we present a framework that identifies a set of criteria, and their associated dimensions that can be used to classify reliability models based on their suitability to software architectures. A survey of several representative approaches to software reliability modeling, based on this framework is presented, which highlights areas of future research. The rest of this paper is organized as follows. In Section 2, we offer some background information on several reliability models, with a specific eye on their architectural relevance. Section 3 outlines our classification framework and how some of the representative reliability models may be classified using our approach. We conclude in Section 4.

2. Background

Modeling, estimating, and analyzing software reliability – during testing – is a discipline with over 30 years of history. Based on their applicability to various phases of the software development life cycle, Software reliability models are classified into allocation, prediction and estimation models [16]. Allocation models are used early in the
SDLC to establish reliability goals for various subsystems. Prediction models precede testing phase of the development and often rely on historical failure data or data available from similar systems to predict the reliability of the system at a given time in the future. The results are typically fed back into the development process to ensure that various allocated reliability goals are met. Finally, estimation models are those that are applicable to testing, deployment, and maintenance phases. They rely on the actual failure data available from the system and provide a (more accurate) quantification of the reliability of the system given its observed behavior.

As mentioned, this distinction specifically relates to the development stage where models are suitable. However, this classification fails to reveal the reliance of the model on different development artifacts. For example, many prediction models still rely on the system’s operation profile (obtained via testing) to perform their analysis. Our study narrows on prediction type models as they relate to software architectural modeling and analysis more specifically.

Over the past few decades, many reliability models have been proposed: Software Reliability Growth Models (SRGMs) are used to predict and estimate software reliability using statistical approaches [9,14,17,21]. Extensive overview of these approaches are previously provided [8,10].

The major shortcoming of SRGM approaches is that they treat the software system as a monolithic entity. They ignore the internal structure of the system, and thus are known as black-box approaches. Consequently, these approaches cannot be used when relating the reliability of the overall system to the reliability of its constituent components. This is a major shortcoming in case of large and complex software systems, where decomposition, separation of concerns, and reuse play important roles their architecture. Finally, these black-box techniques directly leverage failure data, and thus cannot be applied to stages before testing. Estimating the reliability of the system during testing does little in the way of a cost-effective software development process. The defects detected during testing will be significantly more costly to fix than if detected in earlier stages of the development. Additionally, knowing the estimated reliability value at such a late stage leaves few options in meeting the reliability requirements of a software system.

Another category of software reliability modeling techniques is white-box: they consider a system’s internal structure in reliability estimation. These approaches directly leverage the reliability of individual components and their configuration in order to calculate the system’s overall reliability [11,15].

Other classification of software reliability models also exist [2,13]. Goseva-Popstojanova [13] categorizes white-box reliability models (those that consider system’s internal structure in reliability estimation) These approaches directly leverage the reliability of individual components and their configuration in order to calculate the system’s overall reliability [11,15]. They usually assume that the individual component reliability is known or can be obtained via SRGM approaches. Goseva-Popstojanova et al. further classify white-box techniques into path-based, state-based, and additive [13]: path-based models compute software reliability based on the system’s possible execution paths; state-based models use the control flow graph to represent the system’s internal structure and estimate its reliability analytically; finally, additive models simply add the failure rates of each individual unit to determine the overall failure rate of the application and do not consider software structure. In summary, white-box approaches leverage two independent models in calculating reliability: a structural model describing software’s internal structure, and a failure model, describing software’s failure behavior.

Asad et. al [2] leverage Goseva-Popstojanova’s classification and expand it beyond the design level. They provide a high-level classification that discusses model properties applicable to requirements, design, implementation, testing, and validation phases. Furthermore, they leverage their classification and offer a selection method to choose appropriate reliability model. However, since their classification treats “design phase” models at the same level of detail as [13], it lacks specific information that helps an architect determine the reliance of the model on non-design related artifacts such as implementation-level data.

As mentioned, the common theme across many reliability models is their applicability to implementation-level artifacts, and reliability estimation during testing. Even those approaches assumed to be applicable in other development phases rely on estimates of the code size [4]. When architectural, existing approaches consider only the structure of the system. Exceptions are [12,27,35,36]. Reussner et al. [27] build architectural reliability models based on both structural and behavioral specifications of a system. Their parametrized reliability estimation technique assumes the reliability of individual component services to be known. Wang et al. [36] leverage architectural configuration while focusing on architectural styles for building a prediction model that is mostly concerned with sequential control flow across components in a system. Gokhale et al. [12]
focus on uncertainties associated with unknown operational profiles, and provide extensive sensitivity analysis to demonstrate the effectiveness of their approach. Their architectural model represents the control flow among the components, but cannot model concurrency and hierarchy often represented in architectural models [18]. Finally, Yacoub et al. [35] leverage a scenario-based model of system’s behavior and build component dependency graphs to perform reliability analysis. With the exception of [12], they rely on the availability of a running system to obtain the frequency of component service invocations (operational profile).

With the exception of [28, 29], these approaches do not consider the effect of a component’s internal behavior on its reliability. Instead, they 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. Roshandel et al rely on software architecture specification and their associated analysis aimed at revealing existing defects, to build two reliability models. The first model is used to quantify the reliability at the level of individual components. The second model leverages the interactions among those components to quantify the reliability of the entire system.

When predicting software reliability at early stages of development, such as during architectural design, proper knowledge of the system’s operation profile may not be known. This contributes to some level of uncertainty in the parameters used for reliability estimation. In general, if a considerable uncertainty in the estimates of the system’s operational profile exists, then the uncertainty may be propagated to the estimated reliability. Consequently, traditional approaches to software reliability estimation may not be appropriate since they cannot take such uncertainties into consideration. Few approaches assess the uncertainties in reliability estimation heuristically, with variable operational profile, via techniques such as method of moments and simulation-based techniques such Monte Carlo simulation [12]. Other techniques however, assume fixed (a priori known) operational profile and varying component reliability and apply traditional Markov-based sensitivity analysis [3,32]. Hidden Markov Models (HMMs) [26] have been used by [6, 28] in reliability modeling. The focus of [6] is on imperfect debugging during testing and does not relate components’ interaction and reliability estimations. In [28], HMMs are used to predict the reliability of individual components, given the components specification, and the results of standard architectural analysis.

Another related formalism for stochastic modeling of software reliability is Bayesian Networks (BNs). Bayesian Networks have long been used in various areas of science and engineering where a flexible method for reasoning under uncertainty is needed. They also been applied to modeling reliability of engineering systems [1,17,22,23,34]. However, with the exception of [29], they have not been used in modeling reliability of the software architectures. Roshandle uses Dynamic Bayesian Networks (DBNs) [20] to model the architectural reliability of the entire system, when uncertainties associated with early reliability prediction results in lack of information about the system’s operational profile.

In the next section, we present our classification framework, and categorize some of the representative approaches discussed here using our classification.

3. Classification

Extensive studies of software reliability techniques are provided elsewhere [4,8,13]. For the purpose of our research, we are interested in approaches relevant to software architecture and its artifacts. To understand the level of reliance of existing reliability models on software architectural artifacts, we have developed a framework that can be used to classify reliability models. The classification is depicted in Figure 1. We will now discuss different dimensions of this classification.

Basis. At their cores, reliability models can be grouped as those that rely on implementation artifacts (i.e. code), process, or specification artifacts (i.e., architecture-based). The approaches applicable to code are further classified to be black-box, where the system structure is not taken into consideration, or white-box, where the system structure is considered in the reliability model.

Traditional reliability models are implementation-based and may be black-box (e.g. SRGMs), or white-box [13]. Process-based approaches such as [19,24] consider the software development process and its various activities (such as architecture and design stage), and measure the reliability of the software development process. The focus of this paper is on specification-based models, where analytical reliability models may be applied to a model of the software system’s architecture.

Architectural Relevance. Architectural models provide an abstraction of software system’s static and dynamic properties. These models represent a system’s structure and behavior in terms of a collection of components (loci of computation) and connectors (loci of communication)
as organized in an architectural configuration. Furthermore, they model the behavior of the systems both statically and dynamically at runtime [31]. Architecture Description Languages (ADLs) [18] specify software properties in terms of a set of components that communicate via connectors through interfaces. Finally, an architectural style defines a vocabulary of component and connector types and a set of constraints on how instances of these types can be combined in a system [31]. We believe that a useful model to quantify system reliability at the level of software architecture should consider the above modeling elements.

**Model Richness.** Functional properties of software systems are typically described using one or more of the following four views: interfaces, static behaviors, dynamic behaviors, and interaction protocols [30]. Even though other aspects of a system may be modeled using other modeling views, we believe that the above four models provide a comprehensive basis for specifying functional properties of systems. Explicit emphasis on these views (including the consistency among them) has not been much of a focus in existing reliability models. With the exception of Reussner et al. [27], which leverages interfaces, static behaviors (pre/post conditions), and interaction protocols, other approaches focus only on components interaction protocols to estimate system reliability. By focusing on a subset of these modeling views, simplifying assumptions in estimating overall reliability must be made.

As discussed earlier, state-based approaches to reliability modeling use a control flow graph to represent the application structure and the application reliability is estimated analytically. Path-based approaches on the other hand, estimate the reliability by considering the possible execution paths of the application. As a result, the path-based approaches provide only approximate estimates for applications which have infinite paths due to the presence of loops. In our classification, we adopt the same principle and categorize models of interaction protocols as those that are specifying the control flow vs. those that specify interactions among components. The latter enables modeling overall system’s behavior in terms of the behavior of individual components that are being executed concurrently.

**Overall Reliability Assessment.** In general there are two classes of approaches to estimating a system’s overall reliability. The flat techniques (often seen in black-box models), take a non-compositional approach to estimating the

---

**Figure 1. Classification of software reliability models**

<table>
<thead>
<tr>
<th>Criteria</th>
<th>Dimension</th>
<th>Value</th>
<th>Sub-Value</th>
<th>Approaches</th>
</tr>
</thead>
<tbody>
<tr>
<td>Basis</td>
<td>Implementation-level</td>
<td>Black-box</td>
<td></td>
<td>Wang99, Reussner03, Gokhale98, Whittaker93, Li97, Krishnamurthy97, Everette99, Zequeira00, Roshandel06</td>
</tr>
<tr>
<td></td>
<td></td>
<td>White-box</td>
<td></td>
<td>Wang99, Hamlet01, Bondavalli01, Whittaker93, Roshandel06</td>
</tr>
<tr>
<td></td>
<td>Specification-level</td>
<td></td>
<td></td>
<td>Wang99, Mockus03, Nei96, Smidts96</td>
</tr>
<tr>
<td></td>
<td>Process-based</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Architectural Relevance</td>
<td>Components</td>
<td></td>
<td></td>
<td>Wang99, Reussner03, Dolbec95, Gokhale98, Krishnamurthy97, Everette99, Singh01, Roshandel06</td>
</tr>
<tr>
<td></td>
<td>Connectors</td>
<td></td>
<td></td>
<td>Reussner03, Whittaker93, Everette99, Singh01, Roshandel06</td>
</tr>
<tr>
<td></td>
<td>Interfaces</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Configurations</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Architectural Style</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Interface</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Model Richness</td>
<td>Static Behaviors</td>
<td>Dynamic Behaviors</td>
<td>Protocol Control flow</td>
<td>Reussner03, Singh01, Roshandel06</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Interaction protocols</td>
<td>Reussner03, Roshandel06</td>
</tr>
<tr>
<td></td>
<td>Protocols</td>
<td></td>
<td></td>
<td>Reussner03, Li97, Roshandel06</td>
</tr>
<tr>
<td>Overall Reliability Assessment</td>
<td>Flat</td>
<td></td>
<td></td>
<td>Hamlet01, Reussner03, Dolbec95, Krishnamurthy97, Everette99, Roshandel06</td>
</tr>
<tr>
<td></td>
<td>Analytical</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Compositional</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Assumption-based</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
overall reliability. Such approaches are inconsistent with software architecture and its goal of decomposition, reuse, and separation of concerns. The compositional approaches take system’s structure and components’ interactions into account when estimating overall reliability. We further classify them into those that assume a components’ reliability, or reliability of components’ constituent elements (e.g., components’ services) is known. Such assumptions undermines usability of these models. Alternatively, component-level reliability may be predicted analytically given proper functional models of the component itself and based on the result of advanced analyses.

The right-hand column of the classification depicted in Figure 1, illustrates a survey of several representative reliability models classified using our framework. Specifics of these models were discussed in Section 2. The figure identifies several areas where more research in developing relevant reliability models is needed.

4. Conclusion

In this paper, we presented a classification framework for studying reliability modeling techniques. Understanding various approaches using this classification allows us to determine their suitability to software architecture level reliability modeling and analysis. Early reliability analysis and prediction, can be use as an indication of the quality of the software architecture, and may be further used to compare and contrast strengths and weaknesses of several architectures. Such analysis can help the architect to make important trade-off analysis, and select architectures that result in better overall quality.

REFERENCES


