Identifying and Addressing Uncertainty in Architecture-Level Software Reliability Modeling

Leslie Cheung[1], Leana Golubchik[1,2], Nenad Medvidovic[1], Gaurav Sukhatme[1]

[1]Computer Science Department
University of Southern California
Los Angeles, CA 90089, USA

[2]EE-Systems Dept, IMSC
University of Southern California
Los Angeles, CA 90089, USA

{lccheung,leana,neno,gaurav}@usc.edu

Abstract

Assessing reliability at early stages of software development, such as at the level of software architecture, is desirable and can provide a cost-effective way of improving a software system's quality. However, predicting a component's reliability at the architectural level is challenging because of uncertainties associated with the system and its individual components due to the lack of information. This paper discusses representative uncertainties which we have identified at the level of a system's components, and illustrates how to represent them in our reliability modeling framework. Our preliminary evaluation indicates promising results in our framework's ability to handle such uncertainties.

1. Introduction

Software reliability prediction techniques are important tools in the process of ensuring and improving quality in software systems. Conventional software engineering wisdom suggests that assessing reliability (or any other software quality) at later stages of the Software Development Life-Cycle (SDLC), such as testing or maintenance, can be costly. Therefore, assessing reliability at early stages of SDLC is desirable and can provide a more cost-effective way of improving a software system's quality.

It is widely accepted that software architecture is the linchpin of the software development process [10]. Architectural modeling and analysis can provide meaningful insights into a software system's structure, intended behavior, and key properties. Existing research has recognized that software architectural modeling and analysis can be used as a building block for reliability prediction. Several existing techniques have used system structure and behavior as a basis for predicting reliability [3,5,8,14]. However, when viewed at the architecture level, these techniques invariably assume that the reliabilities of the individual components in a system (in the case of system-level reliability modeling), or the reliabilities of a component's elements such as its services (in the case of component-level reliability modeling), are known. We do not believe this assumption to be reasonable. It is unclear how the component reliabilities are obtained in these approaches. The prevailing assumption appears to be that the components have already been implemented and one of the code-level reliability estimation techniques has been applied to them. [3,11,14]. We recognize that since a software system typically comprises multiple components, component reliabilities have a significant affect on system reliability, and therefore component reliability prediction is an important *first* step in system reliability prediction.

A major challenge for predicting reliability at the architectural level is the presence of uncertainties due to the lack of information about the intended behavior of an artifact, e.g., operational profile, implementation details, failure behavior, etc. Existing reliability prediction techniques usually assume that information about the artifact's behavior is either available or can be easily obtained, and do not investigate how these uncertainties are handled.

We have identified several sources of uncertainty and points of variation when considering a software system early in its development, from an architectural perspective:

(1) Different modeling *techniques* for reliability prediction may be effective in different situations. For example, system modelers may choose to model a software system using (a) Markov Chains (MC), (b) Hidden Markov Models (HMM), (c) Bayesian Networks (BN), etc.;

(2) Software developers may work within different *development scenarios*. Example development scenarios may include (a) implementing entirely new systems

1-4244-0910-1/07/$20.00

from scratch, (b) reusing components and/or architectures from previous projects, (c) purchasing software from a vendor, etc.;

(3) The *granularity* of the architectural models may vary significantly. A system may be accompanied by (a) coarse-grained models for very large components, (b) partial models for commercial-off-the-shelf components, (c) very detailed models for safety-critical components, etc.; and

(4) Different *sources of information* about the system's likely usage may be available. Developers may have at their disposal (a) little or no information about an unprecedented system, (b) a functionally similar system whose usage will also likely be similar, (c) access to experts or extensive domain knowledge, etc.;

We posit that an architecture-level software reliability modeling framework should be flexible enough to accommodate this range of possibilities. Our previous work [1] has investigated uncertainties due to different development scenarios and choices of techniques (sources of uncertainty 1 and 2 above). Our focus in this paper will be on uncertainties with regard to granularity of architectural models and different sources of information (sources 3 and 4).

This paper discusses and evaluates how well our component reliability prediction framework proposed in [1,12] addresses these uncertainties. Specifically, we discuss the sensitivity of our results to changes in system parameters, using as a baseline the reliability of a prototype implementation of an existing component. Our preliminary results indicate that our framework is well equipped to handle the uncertainties described above.

The rest of the paper is organized as follows. Section 2 briefly discusses the related research. Section 3 gives an overview of our component reliability prediction framework. Section 4 discusses uncertainties involved in predicting a component's reliability at the architectural level. Preliminary results to date are presented in Section 5. Finally, we conclude this paper in Section 6.

2. 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 [13].

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 [4,6,7,9]. 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 [5,11,14]. However, none of these approaches consider the effect of a component's internal behavior on its reliability. They simply assume that the component's reliability, or the reliability of some of its elements (such component services) is known. They then use these values to obtain system reliability.

3. Component Reliability Prediction Framework

Here, we give an overview of our component reliability prediction framework. A more detailed description of the framework can be found in [1,12]. Broadly, our framework leverages a component's architectural models and behavioral information to construct a stochastic process model of that component. Appropriate solution of that stochastic model gives us the component's predicted reliability.

To construct our stochastic process model for reliability prediction, we need to determine the states and transitions of the stochastic process. For ease of exposition, we present our reliability prediction framework as a three-phase process.

Phase 1: Determining states. This phase is concerned with determining the states of the reliability model; this is done through the use and analysis of architectural models. Specifically, we determine the states corresponding to a component's desirable behavior by leveraging its architectural models, and we determine the states corresponding to undesirable (failure) behavior by detecting inconsistencies between architectural models [13]. Different software architects may come up with models of different granularity, which may have an impact on the prediction. Uncertainties with regard to models of different granularity will be discussed in Section 4.3.

Phase 2: Determining transitions. This phase is concerned with determining state transitions between the states generated in Phase 1 above. This involves predicting the component's desired behavior (i.e., operational profile) and undesirable behavior (i.e., failure information) before it has been implemented. As a result, this phase greatly depends on the type of information available about the component, which will be discussed in detail in Section 4.2.

Phase 3: Computing reliability. This phase is concerned with computing the reliability prediction by solving the model constructed in the first two phases.

4. Uncertainties in Architecture-Level Reliability Modeling

As foreshadowed earlier, this section discusses two of the sources of uncertainty we have identified in predicting a component's reliability at the architectural level. To facilitate our discussion, we first present an example application we will use throughout in the remainder of the paper.

4.1. Example Application

We use the *Controller* component of SCRover as an example application. SCRover is a component-based robotics testbed based on NASA JPL's Mission Data System (MDS) [2]. Here, we focus on the behavior of the robot in a wall-following mission: it should maintain a certain distance from the wall; if the robot is too far from or too close to the wall, it has to turn in an appropriate direction to correct this. The robot has to avoid obstacles by turning in an appropriate direction. As soon as the state of the robot changes, it has to update a database with its new state.

The dynamic behavior model is illustrated using a state-based diagram in Figure 1. Transitions between states are triggered by events, each of which may in turn result in the invocation of an action. Note that an event corresponds to the component's provided interface. For instance, once in the *estimating sensor data* state, the event *move* may trigger the transition to either *turning* or the *going straight* state. The two cases are determined based upon the transitions' guards: if there is an obstacle ahead, or when the distance from the wall is smaller than d_1 or larger than d_2, then the robot will transition into the *turning* state; otherwise it will transition to the *going straight* state.

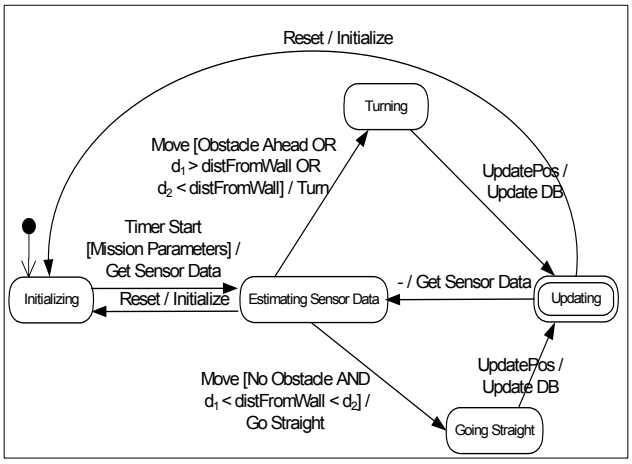

Figure 1. Dynamic behavior model of the *Controller* component.

4.2. Information Sources

As discussed earlier, the lack of information about the behavior of a software artifact is a major constraint for reliability prediction at the architectural level. Here, we explore and identify possible sources of such information and discuss how our reliability prediction framework deals with them. We note here that while our emphasis in this paper is on individual component reliability, the discussion in this section is applicable to both component- and system-level reliability models.

Consider the following example which we use below to illustrate each of the information scenarios. Let us try to estimate the probability, p, of going from the *estimating sensor data* state to the *turning* state in our *Controller* component depicted in Figure 1.

We now present several potential information sources, and how we handle each of them in our component reliability modeling framework.

Data from a functionally similar system. If we have a functionally similar system, we may use its operational profile in reliability prediction. This would provide us with training data needed to apply our HMM-based approach [1] to obtain the corresponding reliability model. In the *Controller* example, such a system may be a robot that performs different missions. The corresponding operational profile will allow us to generate the training data needed toestimate the value of p.

Domain knowledge supplied by an expert. If we have an expert at our disposal, we can ask her to suggest possible operational profiles for reliability prediction. In the *Controller* example, if an expert says the robot is moving straight most of the time, we can estimate p to be relatively small.

Little or no information. When we are missing information about the intended behavior of a component, we may generate a large number of operational profiles to predict the component's reliability in set increments. In our *Controller* example, this may involve varying p from 0 to 1.

4.3. Models of Different Granularities

A component's software architectural model may be provided at different levels of granularity. A software architect may choose to elaborate or elide certain details about the system being modeled depending on the current development context (e.g., whether the model is to be formally analyzed for correctness, passed on to system developers for implementation, or communicated to non-technical stakeholders such as managers or customers).

For example, Figure 2 shows two other possible dynamic behavior models of the *Controller* component at different levels of abstraction. In modeling the *Controller*

component, a software architect may decide to abstract away Figure 1's *estimating sensor data* and *updating* states, as shown in the model in Figure 2a. In another situation, the architect may decide that it is necessary to model the underlying MDS framework that SCRover runs on, and thus describe the *Controller* component using the model in Figure 2b.

There is a trade-off between the resulting accuracy and complexity of the model: a more detailed model is very likely produce more accurate results, but is also likely result in a more complex reliability model that will be more difficult to solve. In Section 5.1, we will study the sensitivity of the results to the model's granularity.

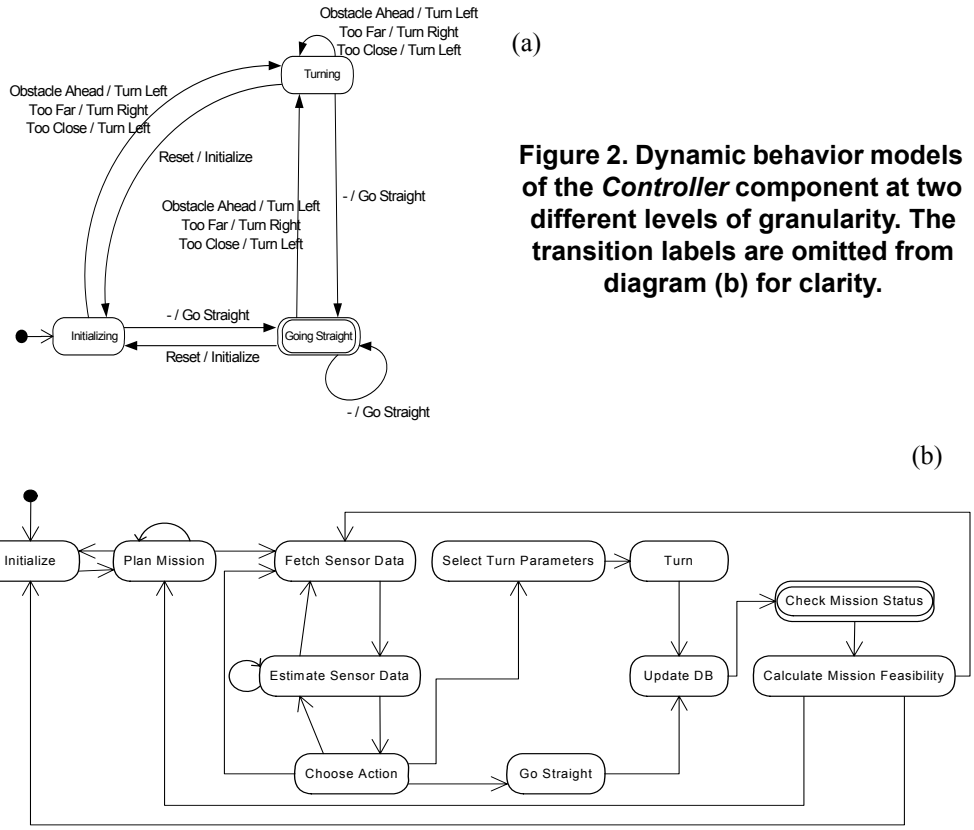

Figure 2. Dynamic behavior models of the *Controller* component at two different levels of granularity. The transition labels are omitted from diagram (b) for clarity.

5. Preliminary Results

For illustration, we will present a set of preliminary results that have emerged from our work to date, focusing specifically on the sensitivity analysis of the SCRover *Controller* component.

5.1. Sensitivity to Information Sources

Here, we study the sensitivity of our technique to different information sources. To validate our results, we constructed a detailed behavioral (control-flow) model of the *Controller* from a prototype implementation of the component that had existed previously. This code-level model is based on a directed graph that represents the component's control structure. We then built a Markov model by leveraging this graph, where a node in the graph translates to a state in the Markov model. Based on the available component maintenance records, we injected defects into the code to simulate failure behavior. We should note that we were not interested in implementation-level faults in this model (e.g., division by zero), but only in architectural defects. We then introduced failure states and transitions in the control structure to represent erroneous behavior corresponding to the injected defects. The results obtained from this model were used as "ground truth" in a large number of experiments.

The following sources of component usage information were considered in this evaluation. Here, we present the parameter values we used in generating Figures 3 and 4. We have performed similar analyses with different inputs, and we obtained qualitatively similar results.

Case (i) – Data from a functionally similar component: As a functionally similar component we selected a robot that walks from one point to another, and avoids obstacles along the way. We then used an operational profile of this component in our reliability prediction of the *Controller* component using our HMM-based approach [1].

Case (ii) – Lack of domain knowledge: We are given the architectural models, and we explore the design space by randomly generating a large set of operational profiles in order to explore the design space. In our *Controller* component example, we vary the probability of turning from 0 to 1 at intervals of 0.01.

Case (iii) – Domain knowledge supplied by an expert: This is similar to the previous case, except that the operational profiles are suggested by the expert, and hence, this reduces the space of operational profiles we consider. In our example, since our expert predicted the robot to be

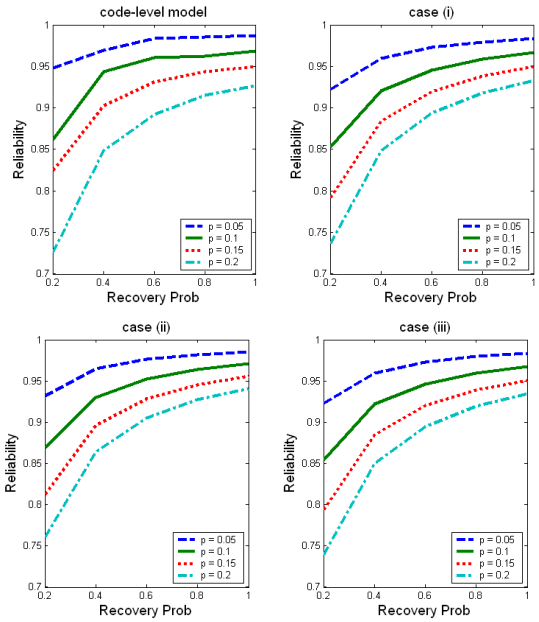

Figure 3. Sensitivity Analysis of Different Information Sources (*sensor* defect)

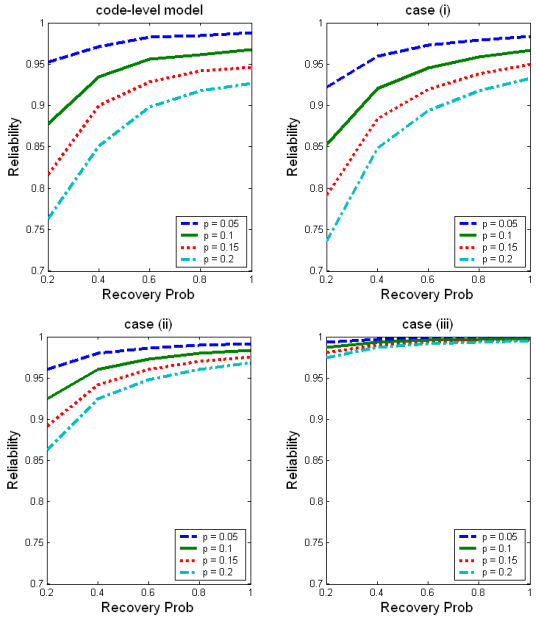

Figure 4. Sensitivity Analysis of Different Information Sources (*turn* defect)

walking straight most of the time, we varied the probability of turning from 0.05 to 0.15.

In one set of experiments, we were interested in the sensitivity of the component reliability when the probabilities of recovering from defects change. To this end, we fixed the failure probability, and varied recovery probabilities from 0.2 to 1.0 in 0.2 increments. We repeated the experiments for different failure probabilities. One set of these experiments was performed by using only a single active class of defect at a time (i.e., by assigning zero probability to failures associated with all other classes of defect). In turn, this resulted in the *Controller* component's dynamic behavior model with only one failure state. Specifically, in Figure 3 we introduced a *sensor* defect, affecting the reliability of the *turning* state, and in Figure 4 we introduced a *turn* defect, affecting the reliability of the *turning* state.

Not surprisingly, we observe that the trends conform to our expectations in all three cases: as recovery probability increases, the reliability of the component increases since the time taken to recover from a failure becomes shorter. Moreover, as failure probability increases, component reliability decreases.

We note that in Figure 4 (*turn* defect), the slope of the curves in Case (iii), where we have domain knowledge, is different from other cases. The reason is that our expert incorrectly predicted the robot to be walking mostly straight: in the prototype the robot walked at an angle most of the time, such that occasionally it was too far from, or too close to, the wall, and had to turn. As a result, the robot spends less time in the *turning* state of the model generated based on our expert's predictions for Case (iii) than it does in the turning state of the actual system. Hence, the turn defect had less impact on the component's reliability.

5.2. Sensitivity to Model Granularity

We have performed sensitivity analysis on models at different levels of abstraction. Figures 5 and 6 show the results of such analysis using the *sensor* and *turn* defect, respectively for Case (iii) discussed in Section 5.1. The five-state model is the example we have used throughout this paper (depicted in Figure 1). The eleven-state and three-state models are depicted in Figure 2. We observe that in both cases, when recovery probability is fixed and failure probability increases from 0.05 to 0.2, reliability values are most sensitive in the three-state model. The other observation is that the three-state model is more sensitive than the five-state model to recovery probability, while the eleven-state model is least sensitive.

This trend can be explained as follows. Failures corresponding to the *turn* defect only occur in the *turning* state in the eleven-state model. On the other hand, time spent in the *turning* state in the five-state model also includes the time spent in the *selecting turn parameters* state in the eleven-state model. As a result, the robot spends more time in the *turning* state in the five-state model than in the eleven-state model, hence the sensitivity is higher in the five-state model. Analogously, since time spent in the *turning* state in the three-state models includes the time spent in *estimating sensor data* and *updating* states in the five-state

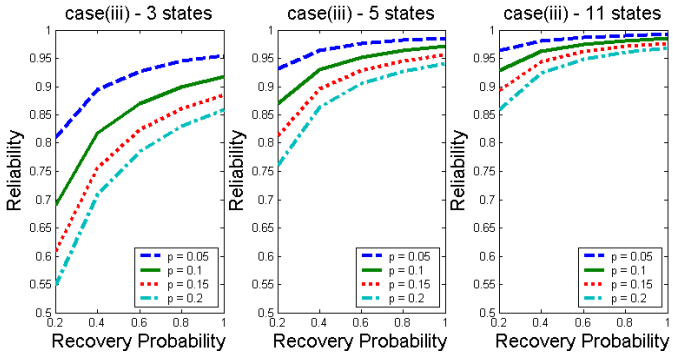

Figure 5. Sensitivity Analysis of Architectural Models of Different Granularities (*sensor* defect)

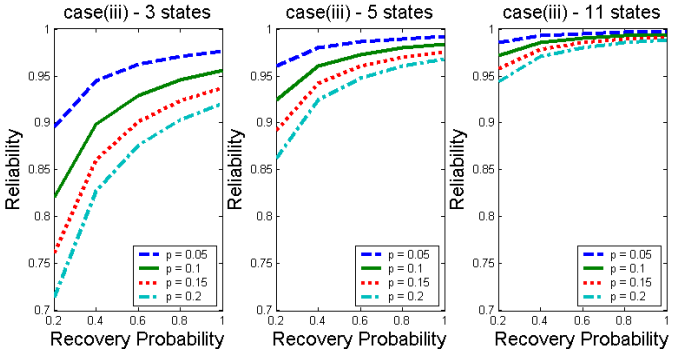

Figure 6. Sensitivity Analysis of Architectural Models of Different Granularities (*turn* defect)

model, the sensitivity of the three-state model is higher than that of the five-state model.

We realize the *Controller* component is small as compared to real world applications. Our on-going evaluation efforts are targeted at several larger-scale systems.

6. Conclusions

The presence of uncertainties, due to the lack of information about a software system, is a major challenge to any architectural-level reliability modeling technique. We identified several sources of uncertainties, and illustrated how to incorporate them into our reliability modeling framework. We performed a preliminary evaluation of our framework, and our initial results indicate that our proposed framework is a promising direction. Our current research direction focuses on a study of how to model uncertainty in the context of system-level reliability estimation. We will also further evaluate our reliability prediction framework on more realistic examples. We will particularly study the trade-off between the accuracy of our framework, as a result of models of different levels of granularity, and the complexity of solving those models.

7. Acknowledgments

This research was funded in part by the NSF 0509539 and 0417274 grants. This work made use of Integrated Media Systems Center Shared Facilities supported by the National Science Foundation 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.

References

[1] Banerjee S., Cheung L., Golubchik L., Medvidovic N., Roshandel R., Sukhatme G., "Engineering Reliability into Hybrid Systems via Rich Design Models: Recent Results and Current Directions", In *Workshop on NSF Next Generation Software (NGS) Program*, Apr 2006.

[2] Boehm B., et al., "Using Empirical Testbeds to Accelerate Technology Maturity and Transition: The SCRover Experience", in *Proceedings of ISESE'04, pp. 117 - 126*, 2004.

[3] Cheung, R.C., A User-Oriented Software Reliability Model, *IEEE Trans. on Software Engineering, 6 (2), pp. 118-125*, 1980.

[4] Goel A.L., Okumoto K., Time-Dependent Error-Detection Rate Models for Software Reliability and Other Performance Measures, *IEEE Trans. on Reliability*, 28(3):206–211, 1979.

[5] Goseva-Popstojanova K. et al., Comparison of Architecture-Based Software Reliability Models, in *ISSRE 2001*, pp. 22-31

[6] Jelinski, Z., Moranda, P. B., Software Reliability Research, *Statistical Computer Performance Evaluation*, edited by W. Freigerger, Academic Press, 1972.

[7] Littlewood, B.A., Verrall, J.L., A Bayesian Reliability Growth Model for Computer Software, *Applied Statistics, Volume 22, pp. 332-346,* 1973.

[8] Littlewood, B., Software Reliability Model for Modular Program Structure, *IEEE Tran. on Reliability, 28 (3),* 1979.

[9] Musa J.D., Okumoto K., Logarithmic Poisson Execution Time Model for Software Reliability Measurement, in *Proceedings of. Compsac 1984, pp. 230-238,* 1984.

[10] Perry, D.E., Wolf, A.L. Foundations for the Study of Software Architecture, *Software Engineering Notes, 17(4),* 1992.

[11] Reussner R., Schmidt H., Poernomo I., Reliability prediction for component-based software architectures, In *Journal of Systems and Software, 66(3), Elsevier Science Inc,* 2003.

[12] Roshandel R., et al., "Estimating Software Component Reliability by Leveraging Architectural Models", *28th Int'l Conference on Software Engineering (ICSE'06),* May 2006.

[13] Roshandel R., Medvidovic N., Multi-View Software Component Modeling for Dependability, in *Architecting Dependable Systems II,* LNCS, 2004.

[14] Yacoub S.M., Cukic B., Ammar H.H., Scenario-Based Reliability Analysis of Component-Based Software, in *10th Int'l Symposium on Software Reliability Engr.,* Boca Raton, Nov. 1999.