

Toward Architecture-Based Reliability Estimation

Roshanak Roshandel, Nenad Medvidovic

*Computer Science Department
University of Southern California
Los Angeles, CA 90089-0781 U.S.A.
{roshande, neno}@usc.edu*

Abstract

Over 30 years of research have gone into software reliability engineering during testing. With today's complex software systems however, reliability has to be built into the early phases of development, including architectural design. We present a software architecture-based approach to estimating component reliability. Our stochastic reliability model is applicable to early stages of development when the implementation artifacts are unavailable and the exact execution profile is unknown.

1. Introduction

Reliability is defined as the probability that a system will perform its intended functionality under specified design limits. *Software* reliability techniques are aimed at reducing or eliminating failures of software systems. Existing software reliability techniques are typically rooted in the field of reliability engineering, and particularly hardware reliability. Such approaches use advanced mathematical models for quantifying software reliability. They complement *testing* by providing estimates of a program's ability to operate without failure. However, they are not always geared to today's complex software systems, and the challenges in their development processes.

As the complexity of software systems increases, it becomes necessary to model them in terms of coarse-grain building blocks. The field of software architecture addresses this issue via high-level abstractions for representing the structure, behavior, and key properties of a software system [12]. Architectural decisions directly affect aspects of system dependability. Identifying and mitigating architectural problems early in development helps to increase dependability of a system in a cost-effective manner. To achieve this goal, reliability and other quality attributes must be "built into" the software system throughout the development process, including during the architectural design phase. Therefore, building reliable software systems requires understanding reliability at the architectural level. However, while some recent approaches have started to quantify reliability at the level of architectural models, they still rely on system implementation to provide either (1) a behavioral model of the system, or (2) individual components' reliabilities.

Our work focuses on the structural and behavioral aspects of a software system's architecture and its effect on the reliability of the system. In this paper, we discuss our

preliminary model for estimating components' reliability when no implementation is available and the operational profile is unknown. Our ongoing research strives to extend this work to estimate a system's overall reliability. In addition to reliability estimation, our model may be used to perform sensitivity analyses to understand the effect of different components on the overall system reliability.

The rest of this paper is organized as follows: Section 2 discusses related works on software reliability. Section 3 provides a high-level overview of our approach to architectural reliability estimation. Section 4 introduces our component reliability model. We conclude in Section 5 and offer an overview of our ongoing and future work.

2. Related work

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 during testing using statistical approaches [5,8,10,11]. *Black-box* approaches are applied to the software system as a whole and largely ignore its internal structure. *White-box* approaches consider a system's internal structure in reliability estimation and directly leverage the reliability of individual components and their configuration in order to calculate the system's overall reliability [6,7,9]. The general assumption is that the individual component reliability is known or can be obtained via SRGMs. White-box techniques are further classified into *path-based* and *state-based* [7]: 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.

The common theme across all of these approaches however, is their applicability to implementation 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. The only exceptions are [14] and [20], both offering compositional approaches to reliability modeling based on system's software architecture. In [14] architectural reliability models are built based on both structural and behavioral specifications of a system. A parametrized reliability estimation technique assumes the reliability of individual component services to be known. Similarly in [20], archi-

tectural configuration is leveraged while focusing on architectural styles for building a prediction model that is mostly concerned with sequential control flow across components in a system. Both approaches assume that component behaviors exhibit the *Markov property*, and build Markov models of component interactions as the basis for calculating system reliability. However, neither approach considers the effect of a component’s internal behavior on its reliability. Additionally, they rely on the availability of a running system to obtain the frequency of component service invocations. Finally, both approaches assume that each state in the underlying Markov model corresponds to an observable event. The latter two assumptions may not always hold when modeling complex systems.

3. An Approach to System Reliability

We propose a three-fold approach to architectural reliability estimation that can be adapted for different modeling notations. At its core, we leverage the analysis results of software architectural artifacts. These result will be fed into a formal reliability model that estimates component’s reliability. Finally, the component reliability values will be plugged into a second formal model to estimate the overall system’s reliability. Figure 1 depicts a high-level view of our approach. Below we highlight these steps.

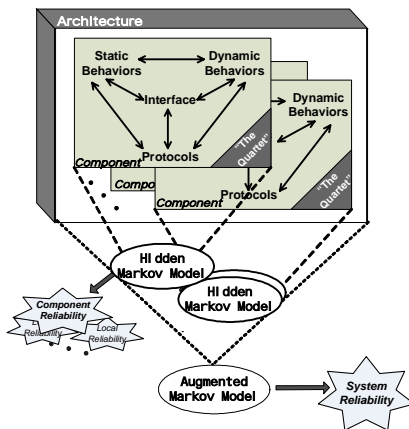


Figure 1. Our approach to component and system reliability modeling

Step 1 - Architectural Modeling. Analysis of architectural specification reveal potential design problems that could translate into system malfunctions once implemented. Architectural description languages (ADLs) and formal modeling notations are widely used to offer critical analyses of software system designs. Our previous work has identified four primary functional aspects on which architectural specification focuses. These four aspects or views (called the *Quartet*) comprehensively model properties of component *interfaces*, their *static behaviors*, their *dynamic behaviors*, and *protocols of interaction* among communicating components [16,17].

The conceptual dependencies among the Quartet views (depicted in Figure 1), both within and across components form the basis for architectural analysis [16,17] supported by our modeling environment [19]. The stochastic models for reliability estimation directly use these analysis results.

Step 2 - Component Reliability. Various classes of architectural defects (obtained in *Step 1*) may affect component

reliability differently. Our preliminary classification of architectural defects [18] can be used to assign weights to various classes of detected defects. The result constitutes a pluggable cost-framework that can be instantiated for different domains [15]. The cost-framework thus provides an input to the component reliability model.

The *dynamic behavior* view of a component (modeled in a state-based notation) provided by the Quartet is the basis for a Hidden Markov Model (HMM) [13] that will be used to estimate the component’s reliability. This paper focuses on describing our HMM in more detail.

Step 3 - System Reliability. Our ongoing research leverages the component reliability values (obtained from *Step 2*) to estimate the overall system reliability in a compositional manner. A series of *concurrent* state machines, each representing an individual component’s *interaction protocol*, will be used to build an augmented Markov model. This model is further extended to include event/action pairs representing complex interactions (the *glue*) among the components. Using this Markov model, components can be ranked to prescribe an order for mitigating faults, while maximizing overall reliability.

4. Component Reliability Estimation

Analysis of architectural specifications reveals potential design problems that could affect a given component’s reliability. We use the results of such analysis, in addition to the state-based model describing the component’s internal behavior, to estimate component reliability via a stochastic model. The stochastic model accounts for uncertainties associated with early development stages, such as lack of knowledge about the operational profile. Below we first discuss our stochastic model conceptually, and then formally define it. We will also demonstrate how it will be used to estimate reliability.

Conceptual Hidden Markov Model. A component’s internal behavior modeled in a state-based notation is used to build an HMM [13] that estimates the component’s reliability. Generally, Markovian models compute reliability as the probability of reaching a final state without failure. Instead of a regular Markov model, we use an HMM for two reasons. First, in a state-based model of a component’s behavior, the correspondence between states and observable events may not exist (i.e., the states are “hidden”). The second rationale for using an HMM is related to the probability of transitions (i.e., frequency of interactions). Early in development, the exact operational profile of the system may be unknown, and thus these values are at best an “educated guess”, possibly obtained by simulating architectural models, or at worst arbitrary.

As an example, consider the dynamic behavioral model of a hypothetical vehicle’s *Controller* component shown in Figure 2(a), where *init* is the initial state and *normal* is a final state. The *Controller* component issues commands to an *Actuator* to change the direction or speed of the vehicle. In this state-based diagram simply observing an event such as *getObstacleDist* cannot help to determine the current

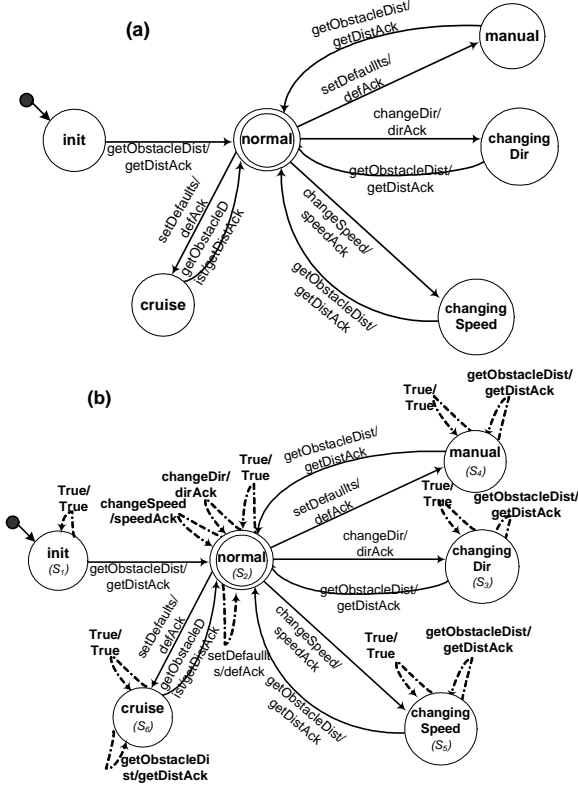


Figure 2. *Controller's* internal behavior model (a), and corresponding Markov extension (b).

(active) state of the system (e.g., both *normal* and *changingSpeed* could be the active state).

An automatable two-step process converts a dynamic behavioral model (Figure 2a) into the desired Discrete Time Markov model (DTM) (Figure 2b) to be used for reliability estimation. The first step extends the state-transition diagram with some implicit transitions not previously modeled. In the second step the transitions are decorated with (initial) probability values. The model then can be solved to obtain re-estimation of transition probabilities as well as the probability of reaching a final state (i.e., reliability).

For the first step, consider Figure 2(b) built from the Figure 2(a). The dotted lines demonstrate some of the additional *implicit* transitions required to build the HMM. Examples include self-transitions at a given state with a *True* label. These self-transitions implicitly exist at every state to indicate that the current state of a system may remain unchanged during some time intervals. Another example of implicit transitions deals with those modeling erroneous behavior. For instance, even though invocation of the *changeDir* interface –when the current state of the system is *normal*– should result in a transition to the *changingDir* state, when an error occurs the service may be invoked but the transition may not be successfully completed (i.e., an exception may happen). This *unwanted* behavior is modeled using transitions that are destined to improper states (e.g., *changeDir* originated at *normal* and destined to *normal* in Figure 2(b)). The extended

Origin state	Observation	P(O)	Reaction	P(R)	Total P P(O).P(R)	Destination state	
init	TRUE	0.1	0.1	TRUE	1	init	
init	getObstacleDist	0.9	0.9	getDistAck	1	normal	
normal	TRUE	0.1	0.1	TRUE	1	normal	
normal	changeDir	0.2	0.1	DirAck	1	normal	
normal	changeDir	0.2	0.1	DirAck	1	changingDir	
normal	changeSpeed	0.4	0.1	speedAck	1	normal	
normal	changeSpeed	0.4	0.3	speedAck	1	changingSpeed	
normal	setDefaults	0.3	0.15	defAck	1	normal	
normal	setDefaults	0.3	0.15	defAck	0.6	0.09	manual
normal	setDefaults	0.3	0.15	defAck	0.4	0.06	cruise
manual	getObstacleDist	0.7	0.7	getDistAck	1	0.7	manual
manual	getObstacleDist	0.8	0.1	getDistAck	1	0.1	normal
manual	TRUE	0.2	0.2	TRUE	1	0.2	manual
changingDir	getObstacleDist	0.4	0.4	getDistAck	1	0.4	changingDir
changingDir	getObstacleDist	0.7	0.3	getDistAck	1	0.3	normal
changingDir	TRUE	0.3	0.3	TRUE	1	0.3	changingDir
changingSpeed	getObstacleDist	0.6	0.59	getDistAck	1	0.59	changingSpeed
changingSpeed	getObstacleDist	0.6	0.01	getDistAck	1	0.01	normal
changingSpeed	TRUE	0.4	0.4	TRUE	1	0.4	changingSpeed
cruise	getObstacleDist	0.2	0.1	getDistAck	1	0.1	cruise
cruise	getObstacleDist	0.2	0.1	getDistAck	1	0.1	normal
cruise	TRUE	0.8	0.8	TRUE	1	0.8	cruise

Figure 3. Initial transition probabilities for the *Controller*

model that includes these implicit transitions is used for reliability estimation.

The second step in building our Markov model is associating probability values (i.e., frequency of interactions) with transitions in the diagram. In cases of reliability modeling when the running system is available, these values may be obtained by observing the system's operational profile. However, in early phases of reliability modeling these probabilities may be unknown. In these cases an initial value (possibly arbitrary, subject to probability axioms) may be assigned, and then the Baum-Welch algorithm [13] may be applied to re-estimate the transition probabilities, given sequences of interface invocations. A sample *initial* transition table for the *Controller* component in terms of probabilities of observations $P(O)$, reactions $P(R)$, and total probability of an observation followed by an action $P(O).P(R)$, is shown in Figure 3. The result of applying the Baum-Welch algorithm to obtain transition probability values is shown later in this section.

Formalized Model. Following this informal discussion on the conceptual steps needed to build a Markov model, we now formally define the HMM used to estimate component reliability. Assuming:

$$S : \text{Set of all possible States, } S = \{S_1, \dots, S_N\}$$

$$N : \text{Number of states}$$

$$q_t : \text{state at time } t$$

$$E : \text{Set of all Interfaces, } E = \{E_1, \dots, E_M\}$$

$$M : \text{Number of Interfaces (i.e., events)}$$

$$F : \text{Set of all Actions, } F = \{F_1, \dots, F_K\}$$

$$K : \text{Number of reactions (i.e., actions)}$$

$$\lambda = (A, B, \pi) \text{ is a Hidden Markov Model such that :}$$

$$A : \text{state transition probability distribution}$$

$$A = \{a_{ij}\}, a_{ij} = \Pr[q_{t+1} = S_j | q_t = S_i], 1 \leq i, j \leq N$$

$$B : \text{Interface probability distribution in state } j$$

$$B = \{b_j(m)\}$$

$$b_j(m) = P[E_m / F_k \text{ at } t | q_t = S_j], 1 \leq j \leq N, 1 \leq m \leq M, 1 \leq k \leq K$$

π : The initial distribution $\pi = \{\pi_i\}$

$$\pi_i = \Pr[q_1 = S_i], 1 \leq i \leq n.$$

for two states S_i and S_j , there may be several transitions with different event/action pairs (E_m/F_k) , for $1 \leq k \leq K$, and $1 \leq m \leq M$. Then, the probability of a transition from S_i to S_j by means of a given E_m via any of the possible

$$\text{actions } F_k \text{ on } E_m \text{ is } \sum_{k=1}^K P_{ijE_m F_k}.$$

For instance, consider the state *normal* in Figure 2(b). Three *setDefault* transitions originate from this state and are destined to the *cruise*, *normal*, and *manual* states. The probability table of this component in Figure 3 specifies that the probability of an occurrence of a *setDefault* event in the *normal* state is 0.3, with equal chances of 0.15 for destination *normal* or *cruise* in the table.

We define the probability T_{ij} of reaching j from i via

$$\text{any of the event/action pairs } E/F \text{ as: } T_{ij} = \sum_{m=1}^M \sum_{k=1}^K P_{ijE_m F_k}.$$

Finally, at each state i (S_i) the following condition among

$$\text{all outgoing transitions exists: } \sum_{m=1}^M \sum_{k=1}^K \sum_{j=1}^n P_{ijE_m F_k} = 1.$$

For instance consider the state *manual*: three events (*getObstacleDist*, *getObstacleDist*, and *True*) originate from this state. According to the probability table there is 80% chance that *getObstacleDist* occurs while a 20% chance that a *True* transition is triggered.

Parameter Re-estimation. Now that the Markov model consisting of set of states S and set of transitions T is constructed, we will apply the Baum-Welch algorithm [1,13] to estimate the parameters of our Hidden Markov Model. The Baum-Welch algorithm is based on the Expectation Maximization (EM) algorithm. EM is an iterative optimization algorithm, which under certain condition find the local maximum of the likelihood function. In particular, it maximizes the posterior probability (i.e., probability distribution *after* observation) of the unknown parameters given the data by defining two variables: *forward* and *backward*.

The forward variable determines the probability of reaching state j from state i , given a sequence of transitions (x_0, \dots, x_t) . Each of these transitions x_i corresponds to an event/action pair. Conversely, the backward variable determines of occurrence of a (future) sequence of transitions, given the current state i . Using the two complementary forward-backward probability the Baum-Welch algorithm evaluates various probabilities, including the probability of a given observation sequence, the probability that the HMM was at a given state S_i at time t , as well as the probability that the HMM was at a given state i at time t and transitioned to state j at time $t+1$.

The details of the two iterative steps may be found in [15] and are outside the scope of this paper. For the rest of the discussion, we assume that the value of the parameter

a_{ij} has converged for all i, j .

The matrix below shows the result of applying the Baum-Welch algorithm to our *Controller* example by initializing the model with a set of *training data* obtained from the transition probability table demonstrated in Figure 3. We generated 100 random sequences of transitions (T_1, \dots, T_5) of length 10 as *training data*. The algorithm then used this dataset to estimate transition probabilities, and obtain maximum likelihood of arriving at a final state. In this particular case, the algorithm converged after 44 iterations and the following transition matrix was calculated. We briefly discuss these results below.

$$T_{\text{new}} = \begin{matrix} & S_1 & S_2 & S_3 & S_4 & S_5 & S_6 \\ \begin{matrix} S_1 \\ S_2 \\ S_3 \\ S_4 \\ S_5 \\ S_6 \end{matrix} & \begin{bmatrix} 0.2917 & 0.3030 & 0.0881 & 0.0015 & 0.3143 & 0.0015 \\ 0.1659 & 0.0052 & 0.1990 & 0.0002 & 0.6286 & 0.0011 \\ 0.0024 & 0.0061 & 0.3982 & 0.0006 & 0.5234 & 0.0693 \\ 0.4930 & 0.0030 & 0.1354 & 0.0292 & 0.0545 & 0.2849 \\ 0.0001 & 0.0535 & 0.1939 & 0.0000 & 0.7405 & 0.0121 \\ 0.6541 & 0.1507 & 0.0028 & 0.1898 & 0.0026 & 0.0000 \end{bmatrix} \end{matrix}$$

Reliability Estimation. Every time transition probabilities are estimated in the iterative process outlined above, the component's reliability gets re-calculated. The final convergence is a result of optimizing the probability of transitions while maximizing the component's reliability. The reliability of a software component depends on the sequence of executions of its services and the reliability of individual states (R_i) in the dynamic behavioral model. If none of the outgoing transitions of a given state are identified as mismatched (in the analysis phase), the reliability of that state is 1. Our approach enables calculation of R_i , based on the Quartet analyses results [15]. In the interest of simplicity, however, for the rest of this discussion let us assume that the reliability at each state of the transition diagram is known to be R_i .

We adopt Cheung's model [2] for reliability estimation. The model leverages the described HMM and extends it by adding two states C and F as the *new* final states representing *Correct* termination and *Failure*, respectively. The details of Cheung's model adaptation may be found in [15]. A Markov transition matrix \hat{P} can then be calculated, where $\hat{P}(i, j)$ is the probability of transition from state S_i to S_j . For any positive integer k , let \hat{P}^k be the k^{th} power of \hat{P} . $\hat{P}^k(i, j)$ represents the probability that, starting from state S_i , the models enters the final state S_j at or before k steps. As demonstrated in similar approaches [2,14,20], the theory of Markov chains can now be used to compute the probability of reaching S_j from S_i , via finite, arbitrarily long sequences. The reliability of the component or the probability of reaching *Correct* final state C from the initial state S_1 via finite arbitrary sequences can thus be calculated as $R_{\text{comp}} = \hat{P}^k(S_1, C)$ [2,3]. The details of this computation may be found in [15].

In the case of the *Controller* component with the calculated transition matrix T_{new} obtained from the HMM re-estimation (discussed above), and the state reliability R_i of

$R_1=0.9$, $R_2=1$, $R_3=0.7$, $R_4=0.95$, $R_5=0.89$, $R_6=0.81$, we now can estimate *Controller*'s reliability. Since the state *normal* is specified to be the final state in the model, the *Controller* component's reliability may be estimated as:

$$R_{comp} = \hat{P}^k(\text{init}, \text{normal}) \times R_{normal} = 0.6446 \times 1 \approx 64\%$$

5. Conclusion and Future Work

Despite the maturity of software reliability techniques, architecture-based reliability modeling has not received much attention. The field of software architecture offers sophisticated modeling and analysis capabilities that often lack quantification and measurement. Our approach is aimed at closing the gap between architectural specification and its effect on software reliability.

HMMs offer an effective stochastic formalism to estimate reliability and other probabilistic dependability attributes. Their flexibility, especially with respect to the partial knowledge of a system's operational profile, is exactly what is needed for early reliability estimation. The re-estimation algorithm can randomly generate transition probabilities, or it can leverage initial values (e.g., as in our example), for parameter re-estimation. Intuitively, if the initial probabilities are representative of the system's (existing or intended) operational profile, the estimated reliability will more closely reflect that of the running system, and the parameters will converge more quickly. Our initial experience confirms this intuition: in the case of the *Controller* component, when operating on arbitrary transition probabilities, the model took 96 iterations to converge on new transition values; when operating on an "educated guess" of the system's expected behavior, the algorithm converged in only 44 iterations.

We plan to provide a comprehensive evaluation of this approach as part of our future work. Our work will focus on a tight integration of the reliability models with our architectural modeling and analysis environment. We are also working on the extended Markov model to estimate the system's overall reliability based on the reliability of individual components. We plan to apply our technique to various applications including a testbed built based on NASA's High Dependability Computing initiative.

6. Acknowledgment

This work is supported by the NASA-HDCP contract to University of Southern California. This material is also based upon work supported by the National Science Foundation under Grant Number CCR-9985441.

7. References

- [1] Baum L. E., An inequality and associated maximization technique in statistical estimation for probabilistic functions of Markov processes. *Inequalities*, 3:1-8, 1972.
- [2] Cheung R.C., A user-oriented software reliability model, *IEEE Trans. on Software Engineering*, (2):118-125, 1980.
- [3] Cinlar E., *Introduction to Stochastic Processes*, Englewood Cliffs, NJ, Prentice-Hall, 1975.
- [4] Dalal, S.R., Software Reliability Models: A Selective Survey and New Directions, *Handbook of Reliability Engineering*, edited by H. Pham, Springer, 2003.
- [5] 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.
- [6] Gokhale S., Philip T., Marinos P., Trivedi K., Unification of finite-failure nonhomogenous Poisson process models through test coverage, in *Proc. of ISSRE-96*.
- [7] Goseva-Popstojanova K., Mathur A.P., Trivedi K.S., Comparison of Architecture-Based Software Reliability Models, in *Proceedings of the 12th IEEE International Symposium on Software Reliability Engineering (ISSRE-2001)*, 2001.
- [8] Jelinski, Z. and Moranda, P. B., Software Reliability Research, *Statistical Computer Performance Evaluation*, edited by W. Freigerger, Academic Press, 1972.
- [9] Krishnamurthy S., Mathur A.P., On the estimation of reliability of a software system using reliability of its components, in *Proc. of the 8th IEEE International Symposium on Software Reliability Engineering*, November 1997.
- [10] Littlewood, B.A., and Verrall, J.L., A Bayesian Reliability Growth Model for Computer Software, *Applied Statistics, Volume 22*, pp. 332-346, 1973.
- [11] Musa J.D., and Okumoto K., Logarithmic Poisson Execution Time Model for Software Reliability Measurement, in *Proceedings of Compsac 1984*, pp. 230-238, 1984.
- [12] Perry, D.E., and Wolf, A.L. Foundations for the Study of Software Architecture, *ACM SIGSOFT Software Engineering Notes*, 17(4):40-52, 1992.
- [13] Rabiner L.R., A Tutorial on Hidden Markov Models, in *Proceedings of the IEEE*, vol. 77, pp. 257-286, 1989
- [14] 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.
- [15] Roshandel R. Calculating Architectural Reliability via Modeling and Analysis, (Qualifying Exam Report), *USC Technical Report Number USC-CSE-2003-516*, December 2003.
- [16] Roshandel R., Medvidovic N., Modeling Multiple Aspects of Software Components, in *Proceeding of Workshop on Specification and Verification of Component-Based Systems, ESEC-FSE03*, Helsinki, Finland, September 2003.
- [17] Roshandel R., Medvidovic N., Multi-View Software Component Modeling for Dependability, in *Architecting Dependable Systems II*, LNCS, 2004 (to appear).
- [18] Roshandel R., Schmerl B., Medvidovic N., Garlan D., and Zhang D., Understanding Tradeoffs among Different Architectural Modeling Approaches, in *Proc. of the 4th Working IEEE/IFIP Conference on Software Architecture, WICSA 2004*, Oslo, Norway, June 2004. (to appear).
- [19] Roshandel R., van der Hoek A., Mikic-Rakic M., Medvidovic N., Mae - A System Model and Environment for Managing Architectural Evolution. To appear in *ACM Trans. on Software Engineering and Methodology*. (2004)
- [20] Wang W., Wu Y., Chen M., An architecture-based software reliability model, in *Proc. of Pacific Rim International Symposium on Dependable Computing*, 1999.