Availability Evaluation of Software Architectures through Formal Methods

João M. Franco, Raul Barbosa and Mário Zenha-Rela
Faculty of Science and Technology
University of Coimbra, Portugal
Email: {jmfranco, rbarbosa, mzrela}@dei.uc.pt

***Abstract*—The quantitative assessment of quality attributes on software architectures allow to support early decisions in the design phase, certify quality requirements established by stakeholders and improve software quality in future architectural changes. In literature, only few of these quality requirements are verified and most often they are manually checked, which is time-consuming and error-prone due to the overwhelmingly complex designs. The goal of this thesis is to provide means for architects predict and analyze availability constraints on software architectures. We plan to generate a stochastic model from an architectural description specified by an Architecture Description Language (ADL) properly annotated to be solved by a probabilistic model-checking tool. This model will allow to quantitatively predict availability and identify bottlenecks that are negatively influencing the overall system availability. Hence, our approach will help architects to avoid undesired or infeasible architectural designs and prevent extra costs in fixing late life-cycle detected problems.**

***Index Terms*—Availability, Prediction, Analysis, Software Architecture**

I. Introduction

Today software-intensive systems need to fulfill varied and complex requirements, forcing the evolution of current approaches of software development to include explicit support for non-functional requirements (*e.g.*, performance, security, scalability or availability). This evolution creates major challenges for software development, maintenance and evolution as software professionals struggle with tools and methodologies that have not been designed to handle those qualities, which determine the success or the failure of large, modern and distributed systems.

This proposal presents a method that gives means for architects predict, analyze and identify quality issues in their designed architecture. Our objective is to support availability as a quality attribute and apply our method to different phases of the software development life-cycle. In the design phase, as an early stage of development, our approach is able to assess the availability of the designed artifact, allowing architects to freely transform, correct and adapt their architectures to meet the desired non-functional requirements before actual implementation and deployment.

On the other side of the development phase, in the maintenance or evolution phase, our approach supports changes performed at the architecture-level by promoting comparisons between different topologies, properties and trade-offs in quality goals. In addition, our approach performs analysis to identify availability bottlenecks and architectural paths that are more prone to error which allows architects to evolve their systems by improving their overall quality.

The main contributions of this PhD thesis are summarized as: (1) automated predictions of quality attributes from architectural software descriptions, (2) a thorough analysis to identify availability constraints in the architecture and (3) the integration of our approach into an architectural-design framework, making publicly available our approach for use by architects.

This paper is organized as follows. Section II describes the state-of-the-art, while Section III states the research objectives addressed in this PhD study. Section IV details the method adopted and Section V introduces the work done so far. Section VI presents the future work before Section VII concludes.

II. State-of-the-art

Software architecture has evolved in the past 20 years and many research studies tried to define the concept, all of them with minor differences depending on domain and people's experience. However, most definitions share common characteristics that can be exemplified by looking at the definition by Len Bass *et al.* [1]: "The software architecture of a program or computing system is the structure or structures of the system, which comprise software elements, the externally visible properties of those elements, and the relationships among them."

Availability is defined as "a measure of the delivery of correct service with respect to the alternation of correct and incorrect service" and Reliability can be understood as "a measure of the continuous delivery of the correct service - or equivalently, of the time to failure" [2]. Availability is measured through Equation 1, where MTTF is defined as Mean Time To Failure and MTTR means Mean Time To Repair.

$$Availability = \frac{MTTF}{MTTF + MTTR} \tag{1}$$

To the best of our knowledge, literature lacks on studies addressing the prediction and assessment of availability as a quality attribute on software architectures. On the other

hand, reliability has been widely considered, tested and several method have been presented. Since reliability and availability are close topics in which they only differ on the Mean Time To Repair (MTTR), we focused our study on the current reliability approaches that are related or can be adopted to our topic of research.

Several studies address the reliability assessment from a software architecture description [3]–[6], among the firsts to propose architecture reliability modeling using Markov chains was Cheung [7] and several surveys were presented since then [8]–[11].

According to these surveys and as depicted by Figure 1, the reliability assessment of software architectures can be performed through three different approaches which combine the architecture with the failure behavior [9]: additive, path-based and state-based models.

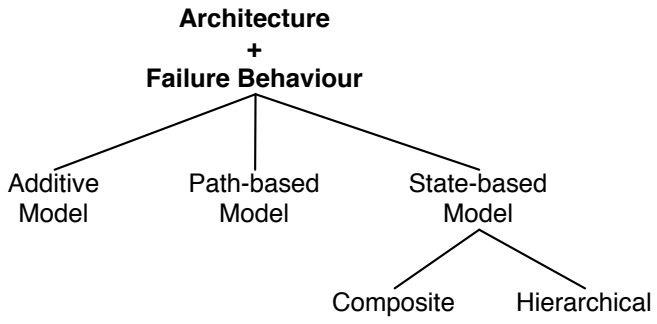


Fig. 1: Approaches to combine the architecture with the failure behavior

The former estimates the reliability using the components failure data and does not consider explicitly the topology of the architecture. Evaluating reliability as an assembly of components considering their interactions and system properties, rather than in isolation, provides a much useful and accurate assessment. The path-based model assesses the reliability of the system according to the possible execution paths, which can be obtained experimentally, by testing or algorithmically. System reliability is calculated by averaging the path reliabilities which assumes an equal likelihood of transversing each path, providing only an approximate estimation of the system reliability. The latter, state-based model, assumes that the transitions between states satisfy the memoryless Markov property, meaning that at any time the future behavior of components or transitions between them is conditionally independent of the past behavior. In addition, the state-based models can be divided into composite [12] and hierarchical [13] methods. The former combines the architecture of the system and the failure behavior of its components into a single model. The hierarchical method considers that the architecture and the failure behavior are detached, more specifically the architecture is modeled by a semi-Markov process and the failure behavior can be modeled according to a Poisson process [14] or by a time-independent failure rate [2]. Hierarchical methods are simpler to compute than the composite ones, since the failure behavior is detached from the architecture. However, the major drawback of the hierarchical method is that it only provides an approximation of the reliability, and hence the reliability metrics obtained using this model are not as accurate as the ones from composite models.

III. Research Objectives

The main goal of this proposal is to research towards a method that supports automatic evaluation and analysis of availability as a quality attribute from a software architecture description. The first step of our approach consists on the generation of a stochastic model from an architecture description specified by an Architecture Description Language (ADL) that exhibits the architectural behavior regarding availability. The second step resides in predicting the overall system availability from the generated stochastic model, according to four architectural degrees of freedom. These degrees consist on the disposal of components and connectors, the system usage profile, the different architectural styles used by the architect and the specified values of properties such as the Mean Time To Repair (MTTR) and the Mean Time Between Failures (MTBF). The final step of this thesis, consists on a sensitivity analysis on the stochastic model, allowing to identify bottlenecks that are negatively influencing the overall system availability.

The availability evaluation during the design phase addresses architect's uncertainties on the assignment of the initial values of the degrees of freedom. In addition, a sensitivity analysis supports the evolution of an architecture by informing the architect about availability bottlenecks and by allowing to perform comparison between different architectures.

The complete development of our approach, will allow to perform architectural validation, avoiding undesired or infeasible designs and prevent extra costs in fixing late detected problems.

Hereupon, and with what was previous stated in mind, this thesis aims to propose new approaches to assess the availability from a software architecture description, helping both the research and practitioner community.

IV. Methodological Approach

This thesis focuses on conducting automated availability evaluations from software architectures. Thus, we divided the big picture into four smaller and well defined tasks: detail the ADL specification to integrate availability features, automate the generation of stochastic models, perform availability evaluation and finally, carry out a sensitivity analysis to identify bottlenecks that require improvement.

A. ADL Specification

We employ ADL annotations by extending architectural entities with the required information to perform availability prediction [15]. More specifically, our approach supports the following annotations to generate an accurate mathematical model of the system:

- Specification of the control flow of the system. The architect can specify the flow of transitions that are being held from a component to another, by using annotated ports to distinguish between output and input transition.
- Assignment of system usage profile. The architect can describe the usage profile of the system by specifying a transition probability to a connector.
- Component failure behavior specification. Each component must be annotated with a probability of failure on demand which can be determined in the design phase through consultation with the commercial entities that developed Commercial Off-The-Shelf (COTS) components or estimated from expert knowledge or historical data for components developed in house. Regarding already developed or deployed systems, the probability of failure can be extracted from the running system [16].
- Identification of used architectural styles. The identification of what styles are being used in the architecture is a requirement for a faithful translation from the architecture to the mathematical model.

B. Stochastic Model Generation

In this step we take as input a system described in the ADL format and we parse it into an intermediate representation, allowing any ADL that complies with the ISO/IEC/IEEE 42010 [17] standard to be parsed and analyzed. After parsing the ADL file, we build the stochastic model which represents the availability behavior of the system. Cheung in 1980 modeled reliability through Markov processes [12] and since then several studies have extended this concept to apply new methods and techniques [5], [18]. Thus, we used the composite model through the generation of an absorbing DTMC (Discrete-Time Markov Chain), where we add two absorbing states C and F, which represent the correct output and the failed one, respectively. Our approach acts in accordance to the following assumptions:

- Every software component can fail. Each module that is mapped from the architecture to the mathematical model has a direct edge to the absorbing state F, which is weighted by its probability of failing.
- The failures are independent between software components. Components in a software system can be viewed as logically independent modules, which can be developed and tested independently from each other [13], [19].
- The transfer of control among modules follows a Markov Process. The transition probability from one component to another is determined through the product of the probability of failure of that component with the estimated usage profile of the system. Therefore, the control transition is independent of the past history of the system and depends only upon the current state, following the memoryless property of a Markov chain [20].
- System availability is the probability of reaching the state C. The computation of the system reliability is performed through the probability of transit between all the components in the system and reaching the absorbing state C, which exhibits the correct behavior of the system or the probability of failure-free of every component in the system.

After the stochastic model has been generated, we load it into the probabilistic model checking tool, Prism [21], in order to perform prediction and analysis on the system availability.

C. Availability Evaluation

In literature exists several availability classifications [22] that may be considered in our approach, depending on the software development phase in which the evaluation takes place. Instantaneous, average up-time and steady-state availability classifications require the current state of system availability at run-time, so they are more suitable to be applied when a system is already deployed. We do not plan to consider achieved and operational availability, because their formulas target more maintenance availability values than the architecture itself, making it not suitable for this thesis.
Thus, inherent availability is the most suitable classification to assess the system availability; however, the architects must estimate MTTR, MTBF and usage profile values by consulting with experts or stakeholders, or by providing an availability range for each architectural element.

D. Sensitivity Analysis

An availability analysis of a software architecture helps to identify architectural elements that are influencing the overall system availability the most. This analysis gives support for different phases of the software development life-cycle. Specifically in the design phase, an analysis provides useful information to address uncertainties about availability and usage profile values that are assigned prior to the deployment of the system, allowing architects to reason about early decisions.

In the evolution phase of an architecture, our approach supports comparison between the original architecture and the evolved one, informing the architect about the trade-offs between them. Also in this phase, the architect may enrich the stochastic model, making it more accurate by obtaining the real values of usage profile and failure data for each component from the system in production.

Therefore, we developed a sensitivity analysis able to solve these uncertainties and support early and future decisions about different architectural alternatives by addressing the following issues:

- Identify component bottlenecks - We study the effect of changes of components' failure data on the overall system availability.
- Analyze usage profile variation - We vary the transition probabilities between components in order to identify which are the usage profiles that have the highest impact on the system availability.
- Analysis ranking - Our approach establishes a ranking system to inform the architect about the impact of the variation on the overall system availability.

To conclude, we intend to apply our method to a real case-study in order to prove the applicability and validity of our

approach. In this case-study we intend to obtain run-time information about the usage profile of the system and the failure data about architectural elements, allowing to assess the real system availability. With this information, we are able to compare the prediction performed in design time with the actual availability obtained from the system, allowing to validate our approach. In addition, we support architectural comparison which allows architects to perform structural changes and identify at real-time the consequences of those changes.

V. Past Work

Methods for reliability prediction and analysis from software architectures have been widely considered and several research studies have focused their attentions on this quality attribute. Although our work targets availability and both non-functional requirements are close topics, we decided to explore the methodology presented in Section IV to the reliability one, since there are more studies to compare and public available case-studies we could analyze, test and confirm our results. With this in mind, following we present the work done so far regarding this PhD thesis.

A. Automated Reliability Prediction from Architectural Descriptions

The methodology presented in subsections IV-A, IV-B, IV-C has been compiled into an article where we demonstrated possible to automate the reliability prediction from software architectural descriptions, depicted through Figure 2 [23]. In more

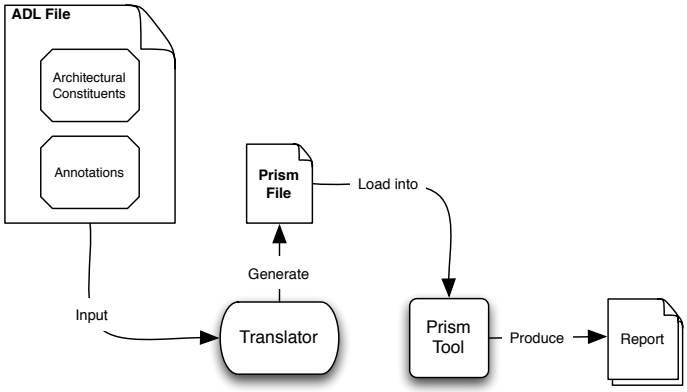


Fig. 2: Translation Process Workflow

detail, our approach parses the ADL file into an intermediate representation of the constituents of the system along with the proper annotations. The intermediate representation allows any ADL that complies with the ISO/IEC/IEEE 42010 [17] standard to be parsed and analyzed. We have successfully used Acme [15], although other ADLs may be target of future work. After the file has been parsed, our application translates the architecture by building a mathematical model into a high-level formal language, which can be loaded into the probabilistic model checking tool, Prism [26]. In turn, Prism allows testing the correctness of the model, checks for the absence of deadlocks and provides the reliability value of the generated model. This work has been validated by comparing our automated procedure against manual predictions carried by other studies and, as a result we obtained $0.0\,\%$ of difference in every comparison with assessments based on the composite model and $0.22\,\%$ on the hierarchical model and also $0.0\,\%$ in every comparison considering architecture styles. As a result, we can conclude that our work has automated the manual assessment architects have been performing until today. This work was published in the joint Working IEEE/IFIP Conference on Software Architecture (WICSA) and the European Conference on Software Architecture (ECSA) in 2012 [23].

B. Support Software Architecture Evolution

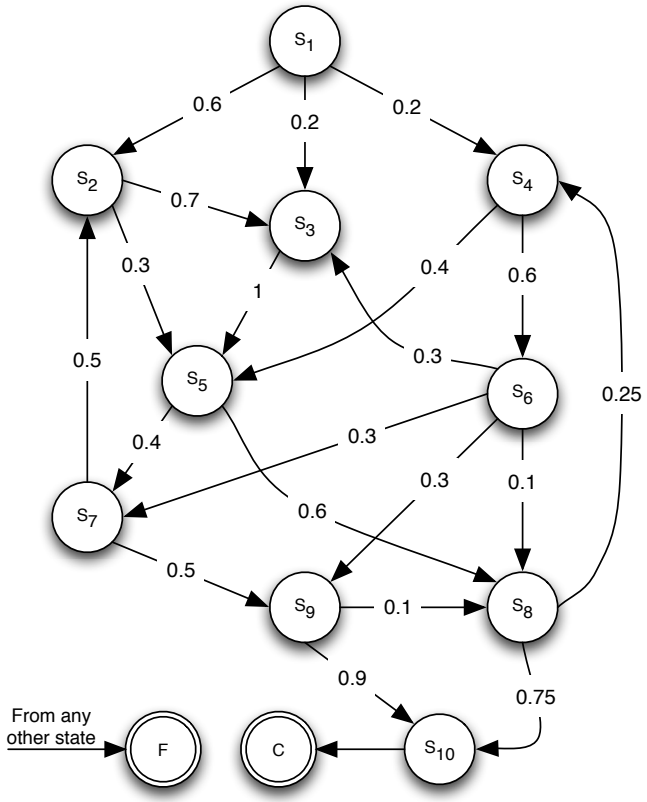
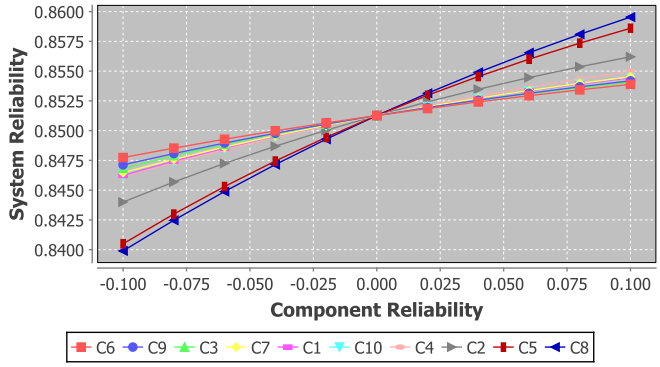


Fig. 3: Example architecture

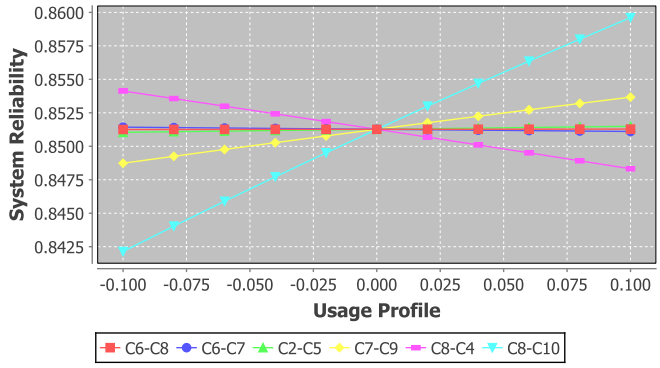
We addressed the method presented in Section IV-D by proposing an approach to evaluate the reliability impact in the evolution of a software architecture. To this end, we perform a sensitivity analysis to determine how the reliability function changes as its input varies, allowing to identify reliability bottlenecks or system paths that are more prone to error. We presented this work in the Latin-American Symposium on Dependable-Computing in 2013 [24] where we showed the effectiveness of our approach by performing a sensitivity analysis in the architecture illustrated in Figure 3.

The results are depicted in Figure 4, showing the impact results on the overall system reliability by varying both reliability (a) and usage profile (b) (also known as operational

profile and it is defined as how the system is used). Figure 4(a)



(a) Reliability



(b) Usage Profile

Fig. 4: Sensitivity Analysis

depicts the variation of 10% of the reliability for each component in the system with respect to the overall system reliability. This graph allows to understand which are the components that influence the system the most. More specifically, components C8 and C5 are on the top of the list, since they have a higher impact in the overall system reliability. Thus, the architect should perform improvements to these components in order to increase the overall system reliability.

The analysis on the variation of the system usage profile is presented in Figure 4(b). Hence, we can conclude that the inter-component transition from C8 to C10 is the one that has a higher impact on the overall system reliability. On the other hand, increasing the usage of the connection between component C8 to C4 will have a negative impact on the system reliability.

C. Affidavit Tool

The two above works have been compiled into a tool targeting system architects and accessible from an architecture development framework. This tool allows architects to reason about the designed architecture, helping to avoid architectural arrangements that might have a negative impact on the overall system reliability, at the same time that it indicates the most suitable arrangement for specific contexts. This paper has been published in INForum a Portuguese Computer Science Symposium in 2013 [25].

Figure 5 presents the Affidavit diagram in which our tool

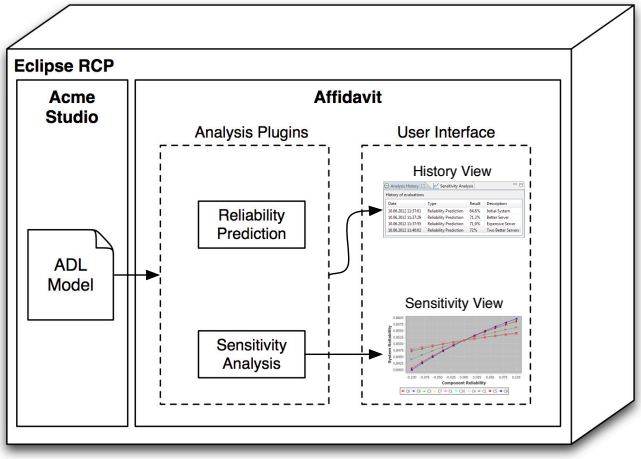


Fig. 5: Affidavit context diagram

was developed as a plugin for AcmeStudio [26]. AcmeStudio is an architecture development environment written as a plugin of the Eclipse RCP Framework [27] and it is a graphical architecture design tool that creates, modifies or iterates over architectural descriptions in the Acme language [15]. Since our tool is close to the architectural design, we can use the formalisms of the Architectural Design Language where the modeled architecture is specified to produce a stochastic model which exhibits the behavior of the system regarding its reliability. Thus, we provide means for architects to predict and analyze system reliability through two different methods: reliability prediction and sensitivity analysis. In each one of these methods, our tool will add the proper information about the simulation to the history view. From the history view, the architect has at his disposal the possibility to visualize the architecture used as input in each simulation. When a sensitivity analysis is performed, Affidavit will also present a graphical representation of the obtained results, as it is illustrated by the sensitivity view in Figure 5.

VI. Future Work

The work done so far has been focusing in reliability as a quality attribute and we prove feasible to generate and solve stochastic models to perform prediction and analysis in an automated fashion. Until now, the methods presented in the scientific community have focused in manual activities which usually are related to time-consuming and error-prone tasks. However, we have raised the bar by building and applying to a software architecture design framework, automated methods to analyze reliability, supporting architects in both design and evolution phases.

As future work, we intend to apply such automated prediction and analysis methods to our target non-functional attribute, availability. The work plan consists on perform availability assessments at runtime to discover failing components and suggest improvements through self-adaptation. In

addition and since formal methods are often related to time-consuming tasks and increased overheads, we plan to show the performance effectiveness of our approach in respect to scalability in terms of the number of architectural elements modeled and also, the time that each availability assessment takes to be executed.

VII. Conclusions

This paper describes the objectives, methods and findings achieved during the PhD period in which we have researched towards a method to support the automated prediction and analysis of availability from formally described software architectures. The main goal of this PhD is to give support for architects to assess, compare and test their designed architectures assuring that they fulfill the availability requirements established by the stakeholders.

The defined objectives for this PhD thesis have been presented in Section III and from the work already done presented in Section V, one can conclude that the objectives have already been achieved regarding reliability with an additional implementation of a tool accessible for practitioners and architects. Future work focuses on applying the developed methods to availability and give support for decision-making of self-adaptive solutions in order to inform the system on what is the best adaptation tactic to achieve a more available system.

Acknowledgment

This material is based upon work supported by the QREN "Programa Operacional Regional do Centro" under Grant CENTRO-07-ST24-FEDER-002003, project ICIS - Intelligent Computing in the Internet of Services. This research was also supported by a PhD research grant from Fundação para a Ciência e Tecnologia (FCT), Portugal [SFRH/BD/89702/2012].

References

[1] L. Bass, P. Clements, and R. Kazman, *Software Architecture in Practice*, 2nd ed. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2003.

[2] M. R. Lyu, Ed., *Handbook of software reliability engineering*. Hightstown, NJ, USA: McGraw-Hill, Inc., 1996.

[3] V. Cortellessa, H. Singh, and B. Cukic, "Early reliability assessment of uml based software models," in *Proceedings of the 3rd International Workshop on Software and Performance*, ser. WOSP '02. New York, NY, USA: ACM, 2002, pp. 302–309.

[4] S. M. Yacoub, B. Cukic, and H. H. Ammar, "Scenario-based reliability analysis of component-based software," in *Proceedings of the 10th International Symposium on Software Reliability Engineering*, ser. ISSRE '99. Washington, DC, USA: IEEE Computer Society, 1999, p. 22.

[5] F. Brosch, B. Buhnova, H. Koziolek, and R. Reussner, "Reliability prediction for fault-tolerant software architectures," in *Proceedings of the Joint ACM SIGSOFT Conference – QoSA and ACM SIGSOFT Symposium – ISARCS on Quality of Software Architectures – QoSA and Architecting Critical Systems – ISARCS*, ser. QoSA-ISARCS '11. New York, NY, USA: ACM, 2011, pp. 75–84.

[6] R. H. Reussner, H. W. Schmidt, and I. Poernomo, "Reliability Prediction for Component-Based Software Architectures," *Journal of Systems and Software – Special Issue of Software Architecture – Engineering Quality Attributes*, vol. 66, no. 3, pp. 241–252, 2003.

[7] R. C. Cheung, "A user-oriented software reliability model bell telephone laboratories, naperville, illinois 60540," *Computer Software and Applications Conference, 1978. COMPSAC '78.*, 1978.

[8] A. Immonen and E. Niemel, "Survey of reliability and availability prediction methods from the viewpoint of software architecture." *Software and System Modeling*, vol. 7, no. 1, pp. 49–65, 2008.

[9] K. Goseva-Popstojanova and K. S. Trivedi, "Architecture-based approach to reliability assessment of software systems." *Perform. Eval.*, vol. 45, no. 2-3, pp. 179–204, 2001.

[10] S. S. Gokhale, "Architecture-based software reliability analysis: Overview and limitations," *IEEE Transactions on Dependable and Secure Computing*, vol. 4, no. 1, pp. 32 –40, January 2007.

[11] D. Pengoria and S. Kumar, "A Study on Software Reliability Engineering Present Paradigms and its Future Considerations," *IEEE International Symposium on Software Reliability Engineering (ISSRE)*, 2009.

[12] R. Cheung, "A user-oriented software reliability model," *IEEE Transactions on Software Engineering*, vol. 6, no. 2, pp. 118–125, 1980.

[13] S. S. Gokhale and K. S. Trivedi, "Reliability prediction and sensitivity analysis based on software architecture." in *IEEE International Symposium on Software Reliability Engineering (ISSRE)*. IEEE Computer Society, 2002, pp. 64–78.

[14] B. Littlewood, "A software reliability model for modular program structure," in *IEEE Trans. on Reliability (Special Issue on Software Reliability)*, vol. R-28, no. 3, August 1979, pp. 241–246.

[15] D. Garlan, R. T. Monroe, and D. Wile, "Acme: Architectural description of component-based systems," in *Foundations of Component-Based Systems*, G. T. Leavens and M. Sitaraman, Eds. Cambridge University Press, 2000, pp. 47–68.

[16] P. Casanova, B. Schmerl, D. Garlan, and R. Abreu, "Architecture-based run-time fault diagnosis," in *Proceedings of the 5th European Conference on Software Architecture*, ser. ECSA'11. Berlin, Heidelberg: Springer-Verlag, 2011, pp. 261–277.

[17] IEEE Computer Society, "ISO/IEC standard for systems and software engineering - recommended practice for architectural description of software-intensive systems," *ISO/IEC 42010 IEEE Std 1471-2000 First edition 2007-07-15*, pp. c1–24, July 2007.

[18] W.-l. Wang, D. Pan, and M.-H. Chen, "Architecture-based software reliability modeling," *Journal of Systems and Software*, vol. 79, no. 1, pp. 132–146, Jan. 2006.

[19] J.-H. Lo, C.-Y. Huang, I.-Y. Chen, S.-Y. Kuo, and M. R. Lyu, "Reliability assessment and sensitivity analysis of software reliability growth modeling based on software module structure." *Journal of Systems and Software*, vol. 76, no. 1, pp. 3–13, 2005.

[20] C. M. Grinstead and L. J. Snell, *Grinstead and Snell's Introduction to Probability*. American Mathematical Society, 2006.

[21] M. Z. Kwiatkowska, G. Norman, and D. Parker, "Prism: probabilistic model checking for performance and reliability analysis." *SIGMETRICS Performance Evaluation Review*, vol. 36, no. 4, pp. 40–45, 2009.

[22] E. Elsayed, *Reliability Engineering*, ser. Wiley Series in Systems Engineering and Management. Wiley, 2012.

[23] J. M. Franco, R. Barbosa, and M. Zenha-Rela, "Automated Reliability Prediction from Formal Architectural Descriptions," in *2012 Joint Working IEEE/IFIP Conference on Software Architecture and European Conference on Software Architecture*. Helsinki, Finland: IEEE computer society, Aug. 2012, pp. 302–309.

[24] ——, "Reliability analysis of software architecture evolution," in *Latin-American Symposium on Dependable Computing (LADC 2013)*, Rio de Janeiro, Brazil, 2013.

[25] J. M. Franco, F. Correia, R. Barbosa, and M. Zenha-Rela, "Affidavit: Automated reliability prediction and analysis of software architectures," in *INForum 2013*. Atas do 5o Simpósio de Informática, 2013, pp. 54–65.

[26] B. Schmerl and D. Garlan, "Acmestudio: Supporting style-centered architecture development," in *Proceedings of the 26th International Conference on Software Engineering*, ser. ICSE '04. Washington, DC, USA: IEEE Computer Society, 2004, pp. 704–705.

[27] W. Beaton and J. d. Rivieres, "Eclipse platform technical overview," The Eclipse Foundation, Tech. Rep., 2006.