Reliability Analysis of Software Architecture Evolution

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

Abstract

Software engineers and practitioners regard software architecture as an important artifact, providing the means to model the structure and behavior of systems and to support early decisions on dependability and other quality attributes. Since systems are most often subject to evolution, the software architecture can be used as an early indicator on the impact of the planned evolution on quality attributes. We propose an automated approach to evaluate the impact on reliability of architecture evolution. Our approach provides relevant information for architects to predict the impact of component reliabilities, usage profile and system structure on the overall reliability. We translate a system's architectural description written in an Architecture Description Language (ADL) to a stochastic model suitable for performing a thorough analysis on the possible architectural modifications. We applied our method to a case study widely used in research in which we identified the reliability bottlenecks and performed structural modifications to obtain an improved architecture regarding its reliability.

1. Introduction

Software architecture is a fundamental activity of software development, in which designers are able to reason about a system's structure and properties at a high level of abstraction, before any design and implementation effort is made. As such, software architecture facilitates early decisions on systemic properties such as reliability, maintainability, and performance. Although it is very useful at the early stages of development, a software architecture is even more valuable as the software evolves and new versions are planned and developed.

Software systems are modified over time in order to address new and changing requirements. The software architecture provides a privileged perspective to reason about the consequences of such changes and how to meet changing requirements in a cost-effective manner. There is therefore a benefit in maintaining a system's architecture updated as the system evolves.

This paper proposes an approach, which we automate, to evaluate the impact on reliability of the evolution of a software architecture. We take a system description written in the Acme architecture description language [10] as input, and generate a reliability model to predict the overall system reliability. To achieve this, we take advantage of the formalism provided by an ADL (Architecture Description Language) to automate the extraction of a probabilistic reliability model. We then conduct a sensitivity analysis on the results, to predict the impact of varying the reliability of individual components, changing the system's usage profile, and modifying the system structure.

As software systems evolve, designers seek to improve reliability or to minimize the impact of accommodating requirement changes on the existing reliability. Although there exist methods for reliability modeling and evaluation [27], surveys [15, 18] outline the need for tool support to make reliability prediction a fluent part of software development.

Acme is a generic software architecture description language which can be used for developing new architectural design and analysis tools. It allows the specification of architectural structures, types and styles as well as the annotation of properties for each one of the constituents on the software architecture, extending the ADL specification in order to perform an analysis on different quality attributes. One important feature of Acme is the human readable interchange format which enables architects to integrate complementary tools and plug-ins to perform architectural analysis or export the architectural description to different architectural design tools.

When planning changes to a software architecture, designers take into consideration the reliability of individual components as well as the system's usage profile. This allows them to identify reliability bottlenecks, *i.e.*, components and connectors which limit the overall reliability, and to recognize diminishing returns on the effort required to improve sub-systems.

To support these activities, we allow architects to con-

duct a sensitivity analysis on the automatically extracted stochastic models, in order to determine how the reliability function changes as its input varies (where the input consists on the reliability of individual components and their usage profile). Regarding the reliability of individual components, our approach determines the magnitude of the impact of each component on the overall reliability; regarding the usage profile, our analysis focuses on the inter-component transition probability to determine which transitions affect the system the most.

We implemented the proposed method and applied it to a case study consisting of a software architecture based on the batch-sequential architectural style. The results of the analysis allow one to identify which evolutions of the architecture are the most promising as well as scenarios in which the returns are already diminished. As the case study shows, it is possible to evaluate not only relatively small architectural changes but also complex architectural evolution scenarios.

This paper is organized as follows. Section 2 describes the background and related work. Section 3 introduces the method adopted in our approach and Section 4 presents the results obtained by conducting an analysis on a case-study. Finally, Section 5 reveals the limitations of our approach before Section 6 concludes this paper.

2 Related Work

The notion that the software is continuously evolving through either intentional or unintentional changes was first discussed at the end of the seventies [22]. More recently, Lehman *et al.* [21] clarified the term *evolution* by distinguishing it in two different groups. The first one addresses evolution as a noun and is concerned with the question *"what"*, focusing in the investigation of the nature of the term and its phenomenon. The second group and the one in which we operate, is concerned with the *"how"* and addresses evolution as a verb, focusing in tools and methods to evolve a software system.

The survey by Chen *et al.* [6] examines the progress in software architecture and its evolution from the year 2000 to 2010 and Breivold *et al.* [3] published a systematic review about software architecture evolution. These studies classified the broad evolution topic into five different categories.

- *Quality considerations during software architecture design*. The software quality is introduced and explicitly considered during the design phase of the software development lifecycle.
- *Architectural quality evaluation*. This category addresses the situation when the architecture is already defined and performs evaluations on the quality of the software, supporting architectural decisions related to quality attribute requirements previously defined by the stakeholders.
- *Economic valuation*. Considers the cost, effort and value of performing changes on the architecture, focusing on the revenue obtained from the software quality.
- *Architectural knowledge management*. Captures architectural information from different sources to enrich the documentation, improving architectural knowledge for quality attributes and their rationale.
- *Modeling techniques*. Techniques to model the behavior and possible impact of the evolution of software architectures. These modeling techniques do not explicitly implement evolvability, but they support decisions and improve software architecture evolution.

Although these studies help to clarify several architectural evolution concepts and they can be the basis of future research and practice, they do not take into account reliability as a quality attribute and neither as a search term in their research. Therefore, we address reliability as a quality attribute of software architecture evolution by developing an approach that belongs to the architectural quality evaluation and in the modeling technique categories. In the former category, our approach takes a software architecture already described in an ADL and evaluates it according to its reliability, helping the architect to reason about decisions related to the topology, interconnection of components, their reliabilities and also the expected usage profile of the system. Our approach also enters in the latter category, modeling techniques, because we generate a stochastic model from the ADL specification reproducing how the system will behave in respect to the system reliability.

Barais *et al.* [1] studied diverse state-of-the-art approaches to evolve software architectures. They conclude that even though there are several ADLs that enable architects to specify their software systems, most of them do not provide means to evolve the architecture. Our approach adds value to these ADLs by assessing the reliability at the software architecture level. In addition, our approach is detached from the architecture, allowing it to be applied to any available ADL that complies with the ISO/IEC/IEEE 42010 [19] standard.

Reliability has been subject to evaluation on the software architecture level since the late seventies by Cheung [7], who was among the first to propose architecture-based reliability evaluation using Markov chains. Since then, several studies have improved his method and proposed new ones [4,11–14,23,24]. Gokhale *et al.* [13] and Lo *et al.* [24] perform a sensitivity analysis on the reliability of a software architecture by varying the expected usage profile and

the reliability of each component. It allows to find existent bottlenecks that are affecting negatively the overall reliability, such as components that are overused or connectors that need to redistribute the system load. Our approach differs from these by applying a non-linear variation on component reliabilities, addressing the issue of exceeding the range of possible values, detailed in Section 3.3.1. In addition, our work is applied directly from an ADL specification, making it possible to automatically build a suitable stochastic model to perform reliability prediction, discarding the issues carried by manual activities (time-consuming and prone to errors).

Our work tries to address some shortcomings identified by research surveys on the architecture-based reliability prediction, such as the lack of automated processes for reliability prediction and analysis, absence of support for different architectural styles and poor method validation. We previously performed an automatic translation from the architecture specification to a stochastic model [8]. This stochastic model exhibits how the system will behave according to the different architectural styles that are applied in the architecture. In addition, we performed validation of our method by comparing our results with previous research studies that used the same case-study. Thus, in this paper we addressed the remaining shortcomings: perform a reliability analysis in an automated fashion from an architecture specification and validation of this method as an extent of the previous work.
In particular, in this paper we accommodate system evolution by providing a thorough analysis on the possible modifications that can be performed in the system. Hence, we provide support for the architect on paramount decisions and insightful architectural trade-offs.

3 Method

A sensitivity analysis can be understood as an exhaustive test on the system in study by performing small changes in the usage profile and reliability of each component or connector. This type of analysis supports the architect deciding on what and where in the system the evolution will have the greatest benefit regarding its reliability.

In our approach we take an ADL specification and generate automatically a stochastic model of the system which will be subject to a sensitivity analysis. In this section we discuss the techniques we adopted and why were chosen in detriment of others.

3.1 Translation from ADL to a Stochastic Model

The translation procedure from a system described in an ADL to a stochastic model was already been specified in a previous work [8] in which was shown that it produces accurate and correct models to predict reliability. Our approach requires the employment of ADL annotations by extending architectural entities with the required information to perform reliability prediction [9]. More specifically, our approach supports the following annotations to build an accurate model of the system:

- *Specification of the control flow of the system*. We used annotated ports to distinguish between output and input transition, describing the transition flow that is being held from a component to another.

- *Assignment of system usage profile*. The average usage of the system is specified through connector properties in which the architect defines the transition probability of passing control from a component to another.

- *Component reliability specification*. Each component has annotated a probability of failure on demand, which will determine its reliability value.

The translation process is illustrated in Figure 1. It can be noticed that we take as input the ADL file with the architectural constituents and the required annotations to generate a stochastic model capable of predicting the system reliability. Our approach parses the ADL file into an intermediate representation, allowing any ADL that complies with the ISO/IEC/IEEE 42010 [19] standard to be parsed and analyzed. We have successfully used Acme, although other ADLs may be target of future work.

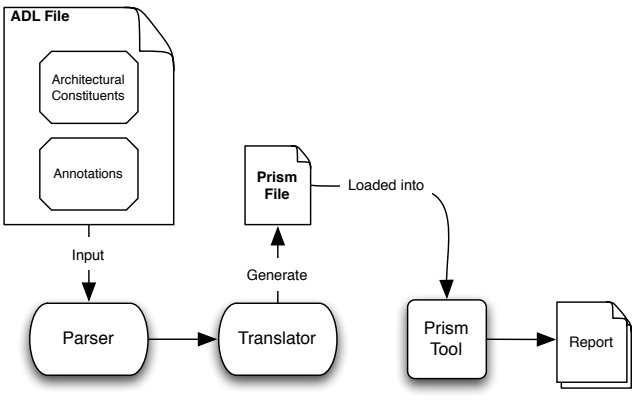


Figure 1: Translation Process Workflow

After the file has been parsed, our application translates the architecture by building a stochastic model in a high-level formal language, which is then loaded into the probabilistic model checker Prism [20].

3.2 Reliability Prediction Process

Several approaches to perform reliability prediction were described in previous research studies [12, 15, 18, 25], targeting different failure behaviors and using distinct reliability assessment methods. Our assumptions, and the methods that we rely upon, were the following:

3.2.1 Architecture and Module Identification

Software architecture defines the software behavior with respect to the interaction among modules. A module is conceived as an independent component of the system, which performs a clear and well-defined function in the system [2]. Moreover, a module may represent an available service (such as a web-service), a function, a class or even a group of classes that have the same functionality or the same interfaces [14, 15].

3.2.2 Failure Behavior

We consider the failure behavior as a failure that may occur during the control transfer between two different components, which is known as the probability of failure on demand. The failure behavior of the modules is specified as a percentage, denoting the number of successful requests over the total requests performed to that specific module. For instance, if a module has 80% of reliability, it means that 8 out of 10 requests are well performed and the other 2 fail on some cause, such as malformed input or other source of failure. The reliability of each component can be either estimated or obtained from failure data collected on a production system.

3.2.3 Combining the architecture with the failure behavior

The reliability assessment on the software architecture level can be performed through three different approaches which combine the architecture with the failure behavior: additive, path-based and state-based models [15]. The former estimates the reliability using the component's failure data and does not consider explicitly the architecture of the system. Hence, as the architecture arrangement is the main focus in our reliability assessment, the additive model will not be considered in this paper.

The path-based model assesses the reliability of the system according to the possible execution paths of the program, which can be obtained experimentally, by testing or algorithmically. System reliability is calculated by averaging the path reliabilities, which can be a drawback due to the presence of loops in the architecture, providing only an approximate estimation of the system reliability.

The latter, state-based model, assumes that the transitions between states have a Markov Property, meaning that at any time the future behavior of components or transitions between them is conditionally independent of the past behavior. This model is considered to be the most accurate for reliability prediction [13] and for this reason we opted to use the state-based model in our approach.

The generated mathematical model is 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 may 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 (*i.e.*,one minus the assigned reliability to the component).
- Failure rates of software components are independent between each other. Components in a software system can be viewed as logically independent modules, which can be developed and tested independently from each other [13, 24]. Thus, each component will have its unique failure rate which is independent of other components that may be developed or executed concurrently.
- 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 reliability 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 [16].
- System reliability 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.

3.3 Sensitivity Analysis

A reliability analysis of a software architecture is an important aspect to identify which components and connectors influence the system reliability the most. It supports the evolution of an architecture by providing information about the trade-off of the changes from the original architecture to the evolved one. Thus, our sensitivity analysis informs

architects about what are the architectural constituents that need urgent attention to be improved and what are the consequences of changing the architecture.

Therefore, we developed a sensitivity analysis able to support decisions about different architectural alternatives by addressing the following issues:

- *Identifying reliability bottlenecks* – We study the effect of changes of components' reliabilities allowing to identify points in the architecture where the variation has a higher impact on the overall system reliability.
- *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 system reliability.
- *Analysis ranking* – Our approach establishes a ranking system to inform the architect about the impact of different variations performed.

3.3.1 Component Reliability Variation

Analyzing the impact of variations in individual component reliabilities (keeping the same usage profile) provides information about the influence on the overall system reliability. With this information in mind, the architect may impose more effort on improving a particular component which is influencing negatively the system by inducting more testing, more code inspections or even, by correcting bugs.

A reliability value is understood as the probability that a system will perform correctly over a given period of time [26] and belongs to the interval $R \in [0, 1]$. Therefore, we do not apply a linear variation as it might exceed the range of the possible reliability values. This is illustrated by the example in Equation 1, where R_i represents the reliability of component i. We perform a linear variation on the reliability value, showing that the result falls outside the interval of possible values for the reliability.

$$R_i = 0.99 \pm 10\% = \{0.891, 1.089\} \notin [\mathbf{0}, \mathbf{1}] \qquad (1)$$

Hence, we apply a logarithmic variation to the reliability of individual components as shown in Equation 2. Although other research studies have applied a linear variation, we use a logarithmic scale which we believe to be more adequate to describe variations in reliability. As we approach 100% reliability it becomes more costly to obtain further increases.

$$R = 1 - 10^{-x} \;\Leftrightarrow\; x = -\log_{10}(1 - R) \qquad (2)$$

Equation 3 shows how we deduced Equation 2 and presents an example on how to calculate the logarithmic variation, where R is the reliability and U represents the unreliability.

$$\begin{aligned} R_i &= 0.99 \\ U_i &= 1 - R_i = 0.01 = 10^{-2} \Leftrightarrow \\ \Leftrightarrow R_i &= 1 - 10^{-2} \Rightarrow x = 2 \\ x \pm 10\% &= 2 \pm 10\% = \{1.8, 2.2\} \\ R_i\, min &= 1 - 10^{-1.8} = 0.9845 \\ R_i\, max &= 1 - 10^{-2.2} = 0.99369 \end{aligned} \qquad (3)$$

3.3.2 Usage Profile Analysis

Each user performs different tasks on the system, leading to distinctive invocations of different methods or functions. The system usage profile is understood as the estimation of how the system will be used and refers to the inter-components transition probability.

The variation of the usage profile must comply with a constraint: the sum of all output transitions of a component must equal to 1.

To solve this constraint we applied the Equation 4, where P_{ij} is the transition probability from the component i to the component j. Then, we calculated P'_{ik} which represents the new transition probability value for the other components k connected to the component i.

$$\begin{aligned} P'_{ij} &= P_{ij} \pm 10\% \\ \Delta p &= |P_{ij} - P'ij| \\ P'_{ik} &= P_{ik} \mp \frac{\Delta p * P_{ik}}{\sum_{k \neq j}^{n} P_{ik}} \end{aligned} \qquad (4)$$

In this analysis we change the system usage profile by varying every transition probability by more and less 10% than the original value.

3.3.3 Analysis ranking

In order to rank the sensitivity analysis performed for the components' reliabilities and usage profile, we calculated the derivative of the reliability function around the point where the variation is null (*i.e.*, variation $= 0\%$).

The value of the derivative of the reliability function describes the impact of the variation on the overall system reliability. In particular, a higher value means that the variation has a greater impact on the system reliability than lower values.

In order to produce an extensive analysis we derived the reliability function with respect to every system variable by calculating the gradient of R. Equations 5 and 6 show how the gradient was calculated according to each component reliability (R_{C1}) and each usage profile (P_{C1-C2}), respectively.

$$\nabla R = \left(\frac{\partial R}{\partial R_{C1}}, \ldots, \frac{\partial R}{\partial R_{C10}} \right) \quad (5)$$

$$\nabla R = \left(\frac{\partial R}{\partial P_{C1-C2}}, \ldots, \frac{\partial R}{\partial P_{C9-C10}} \right) \quad (6)$$

These formulas allow to produce a thorough analysis about what are the system variables that influence the system reliability the most, allowing to inform the user about important architectural changes that need to be performed in the system.

4 Results

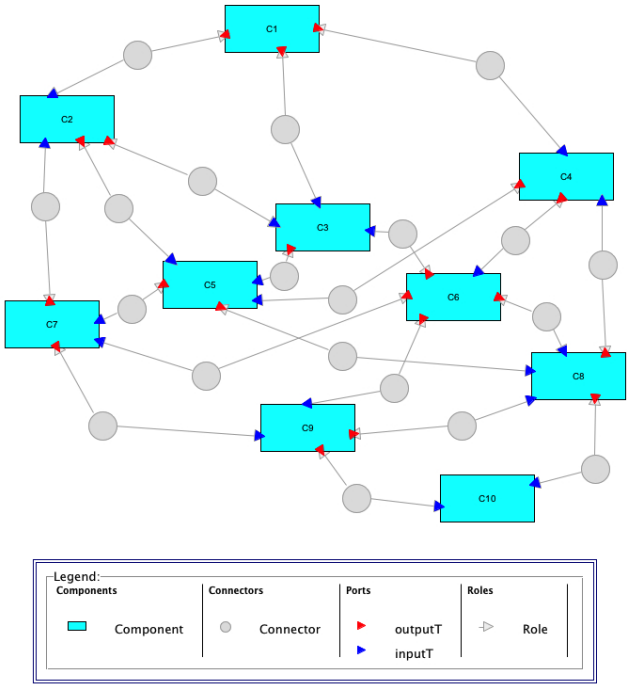
We applied the method described in the previous section to a case study adapted from Gokhale *et al.* [12, 13] and Lo *et al.* [24]. We built an ADL description of the software architecture in Acme, which is illustrated in Figure 2a. Figure 2b depicts the state machine produced automatically from our translation procedure. In this state machine it is also possible to depict the usage profiles used which are specified in each inter-component transition. The used reliability values are specified in Table 1.

Table 1: Component reliabilities

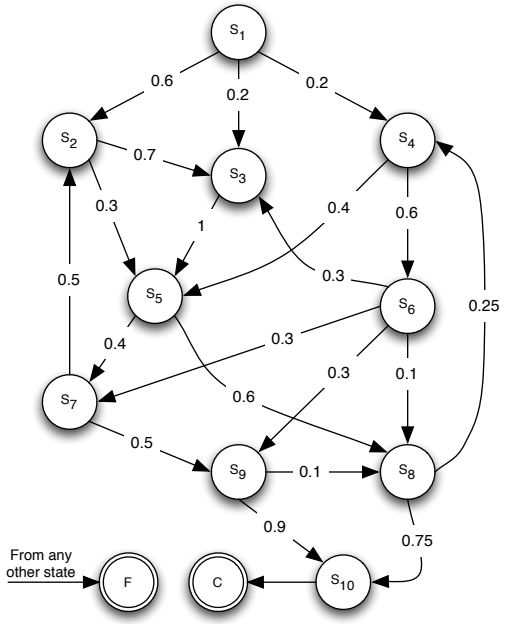
C_i	Reliability of C_i
1	0.99
2	0.98
3	0.99
4	0.96
5	0.98
6	0.95
7	0.98
8	0.96
9	0.97
10	0.99

4.1 Sensitivity analysis

In the sensitivity analysis was performed variation on both the system usage profile and in each component reliability, allowing to identify which are the architectural constituents that are negatively influencing the overall reliability. Following, we detail the obtained results according to each variation performed.



(a)



(b)

Figure 2: Architecture described in Acme (a) and the associated state model (b)

4.1.1 Component Reliabilities

After parsing the ADL and generating the stochastic model, we are able to perform a sensitivity analysis on the system to identify which component reliability is the bottleneck. Figure 3 depicts the variation of 10% of the reliability for each component in the system with respect to the overall system reliability. With this information along with the rank given for each component presented in Table 2, is possible 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, showing that they have a higher impact in the overall system reliability and the architect should perform improvements to these components in order to increase the overall system reliability.

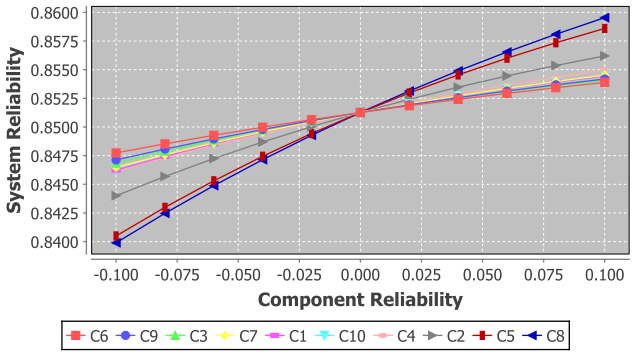


Figure 3: Sensitivity analysis with respect to reliability

Table 2: Results on the component reliability analysis

C_i	Partial Derivative
C8	0.096
C5	0.088
C2	0.059
C4	0.043
C10	0.039
C1	0.039
C7	0.039
C3	0.034
C9	0.034
C6	0.030

4.1.2 Usage profile analysis

The analysis on the variation of the system usage profile is presented in Figure 4, where is illustrated the three best and worst usage profile variations from the total of nineteen inter-component transitions. To support Figure 4, Table 3 lists the sorted ranks obtained from the analysis. It can be concluded 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. This is explained by the fact that the more we raise the usage profile of $C8-C4$, more loops will happen, increasing the number of visits on system components and at same time raising the chance of the requests may fail on some cause.

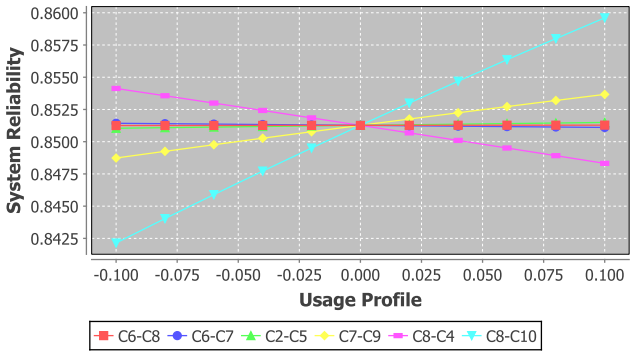


Figure 4: Sensitivity analysis with respect to the usage profile

Table 3: Results on the usage profile analysis

C_i-C_j	Partial Derivative
C8-C10	0.087
C8-C4	0.029
C7-C9	0.024
C2-C5	0.002
C6-C7	0.001
C6-C8	1.5E-4

4.2 Performing structural changes

Implementing changes in the architecture may lead to different reliability values which may not correspond to the reliability requirements established by the stakeholder. Thus, our work aims to provide sufficient information for architects to reason about system evolution, even when very little is known about the system itself. In this case-study, although we use a well-known architectural example, we do not have information about other quality attributes such as cost or performance, as well as if it maps to a real system. Hence, we inform architects on what are the most suitable

design decisions by comparing different architectural alternatives.

The sensitivity analysis performed on the case study showed that the architect could improve the overall system reliability by performing few architectural changes. In particular, the components that act as reliability bottlenecks were identified as the $C8$ and $C5$. Regarding the analysis on the usage profile and if it were possible to redirect traffic or change the usage profile of the system, the architect should focus on the transitions from the component $C8$ (*i.e.*,decrease the usage of $C8 - C4$ and increase the load on the $C8 - C10$).

Thus, we propose an evolution of the architecture by performing the following architectural changes:

- Reliability improvement of 10% on components $C5$ and $C8$.
- Usage profile variation of 10% on $C8 - C4$ and $C8 - C10$.
- Changes on the topology by adding one extra component ($C11$). This component replicates the functionality of $C8$ in order to ease the connection from $C5 - C8$.

Figure 5 illustrates the new architecture and Table 4 presents the used reliabilities for each component in the system.

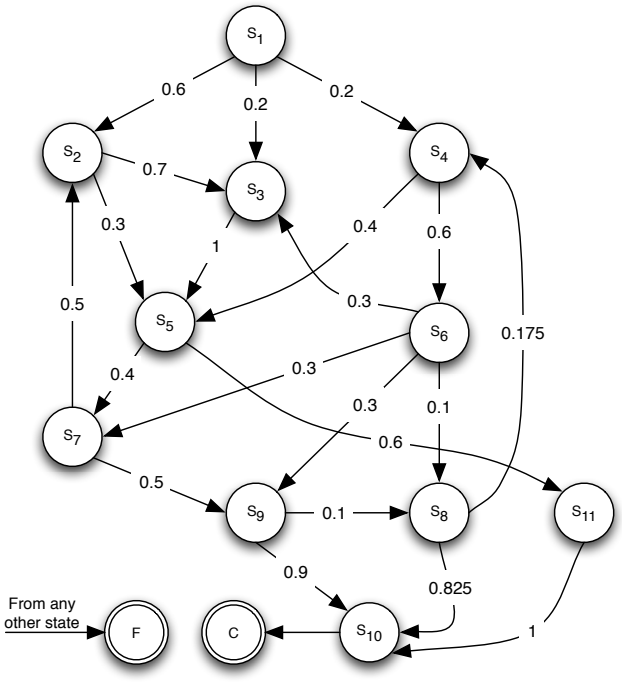


Figure 5: State model of the new architecture

Table 4: Component reliabilities of the new architecture

$\mathbf{C}_i$	**Reliability of** $\mathbf{C}_i$
1	0.99
2	0.98
3	0.99
4	0.96
5	0.9864751
6	0.95
7	0.98
8	0.97100884
9	0.97
10	0.99
11	0.99

The estimated system reliability before the changes were performed, was of 0.8512 and after performing the structural changes, the estimated system reliability was of 0.9023. This signifies a reduction in unreliability of about one third.

As a result, the sensitivity analysis gives important insights to the architect about architectural changes that improve greatly the overall system reliability. In this case, the reliability was improved by applying simple architectural changes, based on observations resulting from the analysis.

To compute these results, our implementation requires the Prism model checker to be executed once for each variation of the architecture. In total, using a MacBook Pro (early 2011) with an intel i7 processor and 8GB of RAM, the iterations of the Prism model checker took about 13 minutes to complete. This completion time can be considered small when compared to the amount of time spent fixing late detected reliability issues in the architecture. These issues can happen when the architect does not perform the required tests and analysis on the system in development, which may not satisfy the reliability requirements established by the stakeholders.

5 Limitations

The limiting factors that our approach face are discussed below along with references to research studies or tools that address those same limitations.

- *Reliability and usage profile values must be known.* We assume that if a system is under evolution, the reliability and usage profile values have already been extracted from the deployed system. The work of Casanova *et al.* [5] show that is possible to monitor a

running system and collect a set of transactions which allow to localize system faults in the architecture. Thus, architects can determine from a deployed system what is the reliability value for each architectural constituent, as well as infer from the transactions what is the average usage profile of the system under evolution.
However, if this information is not available to the architect, he may apply few techniques to extract the required values. Usage profile can be extracted from the source code by using profilers or test coverage tools [28]. Reliability can be obtained by consulting 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 the system architectural specification, we assume that it is updated in each system evolution step in order to have an architecture that matches the deployed system.

- *State-space explosion*. Model checking tools face a common problem, the combinatorial growth of the state-space. This occurs when the model has a large number of states and a great number of transitions between those states exceeding the memory available. In our approach we have not faced this problem, not only because we used simple architectures, but also because the translation procedure to the Prism language is optimized by using the smallest number of states and transitions possible. Groote *et al.* [17] explain how to reduce the size and avoid extremely large models, preventing the occurrence of state-space explosion.

- *Fault handling.* Fault detection and repairing at the component level impacts the overall system reliability. Our approach does not consider explicitly fault handling, although it could be introduced in the model. Specifically, architects can introduce fault detection and repairing by two different methods, either by modifying the generated stochastic model and incorporate the required behavior or by accounting with fault handling directly in the reliability percentage of each module.

6 Conclusions

In this paper we propose a method to automatically perform a sensitivity analysis from an ADL description, regarding its reliability. Our approach makes an extensive analysis on every component reliability and usage profile, identifying bottlenecks and supporting decisions about the required adjustments to adequately evolve a system.

We applied our method to a well-known case study, in which we evolved the architecture by performing small changes in the existent bottlenecks. As a result, the overall system reliability was increased from 0.85 to 0.90, thereby decreasing the unreliability by one third. Lastly, not only was it possible to identify components and connectors that can be modified in order to improve the overall reliability, but we were also able to identify those for which the returns are already diminished.

In future work, we intend to apply our approach to a real case study, extracting real reliability and usage profile values from the deployed system. The generated stochastic model would be able to produce accurate predictions, warning the architect for concrete architectural problems before faults become noticed.

In addition, we plan to represent different failure behaviors besides the probability of failure on demand. Representing time-dependent failure intensities through Continuous-Time Markov Chains would allow to model the dependent execution between components. With these enhancements in our approach it would give support to practitioners and researchers to avoid, prevent and detect undesired or infeasible architectural redesigns which could result in a loss in overall system reliability.

Acknowledgments

This research was supported by a grant from the Carnegie Mellon|Portugal project AFFIDAVIT (PT/ELE/0035/2009).

References

[1] O. Barais, A. L. Meur, and L. Duchien. *Software Evolution*. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008.

[2] L. Bass, P. Clements, and R. Kazman. *Software Architecture in Practice (2nd Edition)*. Addison-Wesley Professional, 2 edition, Apr. 2003.

[3] H. P. Breivold, I. Crnkovic, and M. Larsson. A systematic review of software architecture evolution research. *Information and Software Technology*, 54(1):16–40, Jan. 2012.

[4] 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*, pages 75–84. ACM, 2011.

[5] P. Casanova, B. Schmerl, D. Garlan, and R. Abreu. Architecture-based Run-time Fault Diagnosis. *Engineering*, (September):13–16, 2011.

[6] Y. Chen, X. Li, L. Yi, D. Liu, L. Tang, and H. Yang. A ten-year survey of software architecture. *2010 IEEE International Conference on Software Engineering and Service Sciences*, pages 729–733, July 2010.

[7] R. Cheung. A user-oriented software reliability model. *IEEE Transactions on Software Engineering*, 6(2):118–125, 1980.

[8] J. Franco, R. Barbosa, and M. Zenha-Rela. Automated reliability prediction from formal architectural descriptions. In *2012 Joint Working IEEE/IFIP Conference on Software Architecture (WICSA) and European Conference on Software Architecture (ECSA)*, pages 302 –309, Aug. 2012.

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

[10] D. Garlan, R. T. Monroe, and D. Wile. Foundations of component-based systems. chapter Acme: architectural description of component-based systems, pages 47–67. Cambridge University Press, New York, NY, USA, 2000.

[11] S. Gokhale. Analytical models for architecture-based software reliability prediction: A unification framework. *IEEE Transactions on Reliability*, 55(4):578–590, 2006.

[12] S. S. Gokhale. Architecture-Based Software Reliability Analysis : Overview and Limitations. *IEEE Transactions On Dependable And Secure Computing*, 4(1):32–40, 2007.

[13] S. S. Gokhale and K. S. Trivedi. Reliability prediction and sensitivity analysis based on software architecture. *International Symposium on Software Reliability Engineering*, page 64, 2002.

[14] K. Goseva-Popstojanova, M. Hamill, and R. Perugupalli. Large empirical case study of architecture-based software reliability. In *Proceedings of the 16th IEEE International Symposium on Software Reliability Engineering*, ISSRE '05, pages 43–52, Washington, DC, USA, 2005. IEEE Computer Society.

[15] K. Goseva-Popstojanova and K. S. Trivedi. Architecture-based approach to reliability assessment of software systems. *Perform. Eval.*, 45(2-3):179–204, 2001.

[16] C. M. Grinstead and L. J. Snell. *Grinstead and Snell's Introduction to Probability*. American Mathematical Society, 4 july 2006 edition, 2006.

[17] J. F. Groote, T. W. D. M. Kouters, and A. A. H. Osaiweran. Specification guidelines to avoid the state space explosion problem. *Fundamentals of Software Engineering (FSEN)*, April 2011.

[18] A. Immonen and E. Niemelä. Survey of reliability and availability prediction methods from the viewpoint of software architecture. *Software and Systems Modeling*, 7(1):49–65, Jan. 2008.

[19] ISO/IEC/(IEEE). ISO/IEC 42010 (IEEE Std) 1471-2000 : Systems and Software engineering - Recomended practice for architectural description of software-intensive systems, July 2007.

[20] M. Kwiatkowska, G. Norman, and D. Parker. Prism: Probabilistic model checking for performance and reliability analysis. *ACM SIGMETRICS Performance Evaluation Review*, 36(4):40–45, 2009.

[21] M. Lehman, J. F. Ramil, and G. Kahen. Evolution as a noun and evolution as a verb. In *Proc. Workshop on Software and Organisation Co-evolution (SOCE)*, July 2000.

[22] M. M. Lehman. Programs, cities, students, limits to growth? *Programming Methodology*, pages 42–62, 1978. Inaugural Lecture.

[23] B. Littlewood. Software reliability model for modular program structure. *IEEE Transactions on Reliability*, R-28(3):241 –246, aug. 1979.

[24] 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*, 76(1):3–13, Apr. 2005.

[25] D. Pengoria, S. Kumar, and M. S. Se. A Study on Software Reliability Engineering Present Paradigms and its Future Considerations. *Computing*, 2009.

[26] N. R. Storey. *Safety Critical Computer Systems*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1996.

[27] K. S. Trivedi. *Probability and Statistics with Reliability, Queueing, and Computer Science Applications, 2nd Edition*. Wiley-Interscience, 1 edition, Oct. 2001.

[28] Q. Yang, J. J. Li, and D. Weiss. A survey of coverage based testing tools. In *Proceedings of the 2006 international workshop on Automation of software test*, AST '06, pages 99–103, New York, NY, USA, 2006. ACM.