Exploring Architecture-Based Reliability Analysis of Current Multi-Layered Web Applications

Euler H. Marinho, Alysson A. Mendonça
Department of Computer Science
Federal University of Minas Gerais
Belo Horizonte, Brazil – 31270-010
Email: eulerhm@dcc.ufmg.br, alymenbr@yahoo.com.br

Genaína N. Rodrigues, Vander Alves and Rodrigo Bonifácio
Department of Computer Science
University of Brasília, Brasília, DF - Brazil
Email: genain@ciic.unb.br, valves@unb.br, rbonifacio@ciic.unb.br

Abstract—Web applications architecture evolved from simple web sites add-ons to complex n-layer applications. However, identifying components in this domain is usually a subjective task, as web applications typically comprise web pages, scripts, forms, applets, servlets or simply web objects. As a result of this subjectivity, a component-based life-cycle might reflect on inconsistencies not only on a clear definition of web components, but also on the process development itself. In addition, it is hard to identify which components are more critical according to specific tasks, such that developers could spend more time to improve their design. That quality certainly comprises reliability, availability and security, summing up as dependability attributes. The application of architecture-based reliability analysis techniques in various domains have contributed to solve those problems. However, very little has been done towards the assessment of current web applications in a real-life setting. In this work, we explore the feasibility to apply an architecture-based reliability analysis method in a real-life web application. Our preliminary results show the potential use of this method for the web applications domain, with a considerably accuracy.

I. INTRODUCTION

Architecture-based software reliability assessment has gained increased attention on the last decade [19], but it is usually considered a challenging task. Firstly, the impact of faults on system reliability is directly dependent on how often each part of the system is executed. This is associated to the usage profile of a system, which is usually known after the system is deployed in production environment. Secondly, reliability of software architecture depends on reliability of individual software components. Obtaining failure of the components represents a challenge both on early stages and on late stages of the development. On early stages, the degree of abstraction could be very high, which may lead to imprecise estimation of component reliability. On late stages this is also challenging. For instance, component failure of industrial-strength software is mostly associated to detailed information derived from fine grained bug reports fields, tailored with respect to software implementation and not necessarily to the software architecture. This would not be a problem if commonly present, in practice, the well known phenomena of architectural drift (or erosion) [30]. As a result, there is commonly unclear failure association between the low abstraction level of the software modules implementation and the architecture components. On top of that, complex interactions among components (or between components and the execution environment) may be unknown.

Assessing reliability in web applications is even harder. First, there is a lack of literature work dedicated to objectively and accurately assessing current web applications reliability from the perspective of software architecture. However, from the perspective of existing web logs there are some emerging work [32]. In addition, web applications usually comprise components implemented in different languages (markup, scripting, query, programming languages) and from different sources [25], for example, legacy components, COTS, and so on. According to a conceptual view of this application class proposed by Di Lucca et al. [26], the definition of component for this kind of application is subjective, where it could be Web Pages, Scripts, Forms, Applets, Servlets or simply Web Objects. Besides, with the advent of Rich Internet Applications (RIA) using technologies as Ajax [14], there has been a change in the architecture style of web applications [27]. As a result of this subjectivity, a component-based life-cycle might lead to inconsistencies not only on a clear definition of web application components, but also on the software development process itself. In particular, it is hard to identify critical components to which more project resources developers should allocate to improve the quality of the application. To sum up, the software dependability attributes [3], e.g., reliability, availability and security, may be seriously compromised.

Accordingly, in this paper we explore the suitability of accurately analyzing industrial-strength web applications through our approach to reliability analysis from the perspective of the software architecture [34]. In particular, here we apply our approach by considering a simplified, but common layered-architecture decomposition. Based on this simplification, we are able to assign the fail history of a project to a small, but comprehensive, number of components, even though the implementation might comprise assets implemented in different languages. We perform an evaluation on a real-life web application, developed with current web technologies. This application organizes the activities related to supply contracting of the State Government of Minas Gerais, aiming at increasing the transparency of contract processes.
Similar to our previous work, here we use the Prism Model Checker [23] to perform simulations, in such a way that we can reason about the reliability contribution of the individual components with respect to the whole system reliability. Nevertheless, here we use historical data of a real web application to estimate the components’ reliability and transitions’ probability. The remainder of this work is organized as follows. In Section II we briefly introduce our technique for architecture-based reliability analysis and the tool environment for the quantitative evaluation. In Section III we describe the industrial-strength web system to conduct our evaluation. The quantitative validation of this results is presented in Section IV. Finally, related work is discussed in Section V, whereas in Section VI we conclude and highlight our future directions.

II. Method

Our method for reliability and sensitivity analysis consists of firstly identifying ways to convert the UML models (particularly the Activity and Sequence Diagrams) into the modeling language used in Prism, while preserving the semantic expressed in the UML models. The purpose of modeling in Prism as a probabilistic model checking tool is to make a qualitative and quantitative dependability analysis of the model using sound techniques. Secondly, we annotate the Prism model with variables denoting component reliability and transition probabilities in order to quantify its dependability through PCTL properties. Next section introduces the Prism model checker, whereas the conversion process from UML to Prism models is described in Section II-B. The analysis process is domain-specific and is described in the context of the example in Section III.

A. The PRISM Model Checker

The reliability analysis of the web system in this work is accomplished through a probabilistic model checking tool called Prism [23]. The reason for choosing Prism as the probabilistic state-based model checker in this study was twofold: (1) tool maturity, considering the number of successful case studies that have used the tool [24]; and (2) the richness of the tool environment, which is able to represent various kinds of probabilistic models and their evaluations, as we briefly explain below.

Prism is a tool for formal modeling and analysis of systems which exhibit random or probabilistic behavior. It supports three types of probabilistic models: Discrete-Time Markov chains (DTMCs), Continuous-Time Markov Chains (CTMCs) and Markov Decision Processes (MDPs), plus extensions of these models such as the ability to specify costs and rewards. The tool has three environments: (1) one for system modeling in Prism language, a state-based language derived from the Reactive Modules formalism; (2) one for model simulation; and (3) one for property specification, which uses temporal logic such as the Probabilistic Computational Tree Logic (PCTL) [17], [6] and includes extensions for quantitative specifications and expressions of costs and rewards.

In the modeling environment we model processes, which in Prism are called modules. A model in Prism is composed of a number of modules. Each module has a set of finite-ranged variables, which define the possible states of that module. The final model is the synthesis of all modules through parallel composition. Each module is composed of a set of guarded commands. For example, a DTMC command in Prism takes the form:

\[ \text{[action]} < \text{guard} >\rightarrow< \text{probability} >:< \text{update} >; \]

The guard is a predicate over all variables in the model, and once it is satisfied, the module will make a transition with a certain probability to the update state, where \(0 \leq \text{probability} \leq 1\). The action can be used in order to tag a command that is synchronized with other commands in the same or in a different module. When there is no action label, the command will run asynchronously. An example of a simple Prism command is the following:

\[ \text{[notify]} s = 0 \rightarrow \text{Rel} : (s' = 1) + (1 - \text{Rel}) : (s' = 2); \]

which states that, if \(s\) is 0, it makes the transition to state 1, with probability \(\text{Rel}\), or to state 2, with probability \(1 - \text{Rel}\). Also, note that we use the action notify in order to synchronize with other commands labeled with the same action in accordance to Communicating Sequential Process (CSP) rules [18], which synchronizes all commands with the same action label, once their guard conditions are satisfied.

Once the system modeling is completed using the Prism specification language, Prism reads and parses the language statements and constructs the corresponding probabilistic model, in this case a DTMC (although it can also be used for CTMC and MDPs as well). Prism computes the set of all states reachable from the initial state and checks the model for deadlocks. In the simulation environment, Prism allows the visualization of possible execution traces of the synthesized model.

Another feature in Prism is the ability to specify properties of the probabilistic model. Properties in Prism are expressed using temporal logic (e.g., in PCTL). Prism also performs model checking, determining the quantitative value of each specified property and whether the model satisfies it. For example, in dependability analysis a very useful property is the reachability property expressed as: \(P =? [F(\Phi)]\), which computes the probability of the system to eventually reach a state that satisfies \(\Phi\). In reliability analysis, reachability is an important property to satisfy. It guarantees that the final successful state of the system will be reached, in this case, regardless of the time elapsed to reach it from the initial state.

B. Conversion from UML to Prism Models

This section presents how we map UML models, particularly Activity and Sequence diagrams, to Prism. Initially, each action node becomes a module in Prism. The conversion process consists in first building a state machine model that represents each node from the Activity Diagram (AD). The AD is composed of two types of nodes: decision
and action. Action nodes represent execution scenarios, each represented as a sequence diagram (SD). Each message in the SD is represented as a transition between states, annotated with the probability in which the component performs that message successfully.

Specifically, the conversion from the action node to a PRISM module is accomplished as follows. For each component service execution in the SD that models the action node, there is a corresponding state in the state machine, while the messages exchanged between components in the SD are represented as labeled transitions between states. Therefore, for each state of the state machine model there is a component $C$ processing a service. For this reason, each state is associated with the component reliability $R_C$, i.e., the probability of successful execution of the service, while the probability of the service failure $(1 − R_C)$ is represented by a transition to state $E$. By failure we simply mean that an error is propagated to the service interface causing deviation from correct to incorrect service [3]. This failure can be classified in various modes: domain, detectability, consistency and severity. Our major property of interest in our modeling is the probability of reaching the end state of the system successfully, i.e. reachability.

Differently, decision nodes represent choices and each of their outgoing transitions is represented as a transition between states in the state machine model, annotated with the probability of transitions between scenarios. Precisely, the conversion from decision node to a PRISM module consists of representing the probabilities of transition $PT_{ij}$ from a decision node $i$ to an action node $j$. This information would be normally derived from a system usage profile [28]. Therefore, each decision node in the AD is represented as a state and each outgoing transition in the state machine is labeled with $PT_{ij}$. Thus, from decision node $i$, the sum of the probabilities $PT_{ij}$ for all successor action nodes $j$ is equal to one.

Once the PRISM model is finished, thus consisting of a set of modules representing state machines, the synthesis of the final stochastic model follows the CSP synchronization rules and the probabilistic compositions for the corresponding Markov model of choice. In our case, we focus on DTMC class as it describes a more direct association with state machine models as each state and their respective sets of transitions can be consistently represented as a command line in DTMC, which is not usually true for other kinds of markov models. Therefore, we annotate the models with discrete values of their corresponding probabilities of both component reliability and transition probabilities to the states.

III. EXAMPLE IN THE WEB DOMAIN

We performed the evaluation of the method described previously by analyzing an example in the Web domain. This section describes the example system, which is then evaluated in the following section. Seplag is the target system of our evaluation. It is a portal developed by the Laboratory of System and Software Engineering, which is also CMM level II certified, hosted at the Computer Science Department of the Federal University of Minas Gerais. This industrial-strength web system organizes the activities related to supply contracting of the State Government, aiming to increase the transparency of contract processes. Its size is about 143 KLOC and 1950 classes.

In this study, we apply the methodology described in Section II to estimate the web application reliability and validate our results through a process that comprises the following steps: (1) define a high-level view of the architecture into layers, (2) reverse engineer of the system by defining the scenario executions as method calls between system classes, (3) define the appropriate abstraction level to represent the system components, (4) associate components to their respective architecture layer, (5) estimate reliability of those components with respect to a particular failure behavior, (6) estimate of transition probabilities (in our case between scenarios of execution based on usage profiles), (7) apply the methodology presented in Section II and (8) measure the actual system reliability (according to component failure classification) to compare with estimated results in task 7.

Initially, our knowledge about the system maturity was restricted to a log file, hosted by Bugzilla [7], containing bugs identified during alpha test activities mainly. The log file also had several descriptions of bugs. For our study, we only consider failures reported as critical or blocking, namely failures that prevent the execution of the system.

For tasks 1 to 4, there were no clear component identification and neither a clear component-based architecture view of the chosen web application. However, the software documentation provided high level decisions to divide system in layers. That first obstacle made difficult our work and took us a great deal of time to identify coarse grained components that we could consistently associate with the bug file results. To accomplish this task, it was necessary the assistance of a software engineer in order to identify the components and their respective failure data. We then filtered out those failure of interest, i.e. critical or blocking, in order to carry out task 5. The estimation of transition probabilities in task 6 was based on information extracted from the database as well as an educated guess of the engineers for the frequency of process queries and update. Finally, task 8 was accomplished based on the same bug file used in task 5.

In the next section, we present the outcome of tasks 1 to 6 and an excerpt of our analysis model for the reliability estimation.

A. Software Requirements and Architecture

Considering the magnitude of Seplag, we decided to focus our analysis on the core of that application, described by the Use Case Management of Purchase Process Records. It encompasses one main flow, six subflows and eleven alternative flows. The use case is represented by four alternative flows regarding the CRUD operations (Create, Read, Update, Delete) related to the purchase processes. According to Synergia engineers, these CRUD operations represent the main
functionality of the Seplag Portal, since they are used by most other functionalities and by the largest number of actors.

According to Seplag software engineers, the web application was developed using high level architectural decisions, based on the MVC style. In Figure 1, we depict an abstract view of the web application architecture with the identified components, composed by five layers and their respective dependencies.

In Seplag’s architecture, the boundary layer gathers the classes implementing the user interface; the control layer groups the classes acting as access point to the use case functions; the entity layer gathers the classes implementing information units potentially reusable inside a domain and often persistent; the persistent layer groups the classes that convert data between the entity layer and the physical mechanisms of persistence, as databases or files; the system layer gathers useful classes implementing common services used by other classes.

Following the first steps of our reliability analysis methodology, we first devise the activity diagram of the Seplag Portal, depicted on Figure 2. That diagram encompasses the CRUD operations previously mentioned.

For each action node in the activity diagram in Figure 2, we shall create a sequence diagram in order to model the components interaction within that action. The interactions between the components of each flow were obtained through the method calls analyzed by a software engineer. However, as a clear view of the components was not provided, many classes were identified in this stage as potential candidates for coarse grained components. We then realized that the coupling between the identified components was quite high.

After extensive refinements supported by Seplag software engineers, we managed to find out those classes that would fit as the desired components in a higher abstraction level. The components are represented in Figure 1 inside their respective layers.

An example of a sequence diagram illustrating the realization of an activity is depicted on Figure 3. That diagram presents the components involved in the Query of Purchase Process use case flow. The initial letter of the component name denotes the layer to which the component is related to. Three other sequence diagrams (not shown here for brevity) correspond to the other activities of the activity diagram and represent the steps of the system for creating, deleting and updating the processes in Seplag.

B. Component Reliabilities and Transition Probabilities

Once the components were identified, we needed to associate the components to the log failure data. Once again, we had to make use of Seplag software engineers to make that association. The log file comprehends the period of one year. The main fields of log file are: Bug# is the bug identification; Status and Resolution define and track the life cycle of the bug and Severity classifies the impact of the bug in the overall functioning of the system. In this work, we have considered only blocking or critical failure types as they have a significant impact of the system functioning, as presented on Table I.

The log failure data was then used to calculate the component reliabilities. In order to carry this out, we selected the total number of failures reported as critical or blocking and the number of system executions. The first step consists on associating the failures to their respective components. Compared to the whole process of Seplag’s reliability analysis,
this step was considered one of the most cumbersome, as we needed to match the failure description to their respective components. However, the failure description was either a high level message of the failed services or the raised exceptions. Mostly through the former, which required the assistance of the system specialists to match the service failure message to their related implementation classes. In order to estimate the component reliability, we use the following formula:

\[ R = 1 - \frac{F}{N} \]  

(1)

where \( F \) is the number of component failure, considering blocking or critical failures types that were fixed after detected. \( N \) is the considered number of times the components were executed, following the frequency of the CRUD operations registered on the database. Following Equation 1, the Seplag reliability is 99.6%.

The calculation of the transition probabilities was straightforward, though. The transition probabilities was calculated on the basis of the number of database records, as the context of our experiment focused on the CRUD operations. Finally, after the component reliabilities and the transition probabilities calculated, we could then apply our approach to architecture-based reliability analysis. The results are depicted on Table II. Note that the transitions in Table II follow the names of the transitions in Figure 2.

The last step on the analysis methodology is the conversion of the models in to PRISM in order to proceed with the software reliability estimation. The overall procedure of the conversion process for our study is as follows. There are four action nodes in the activity diagram, each represented by a sequence diagram, and two decision nodes: one for the query-creation decision and one for the query-delete-update decision. Each action and decision node are modeled as a state machine, following the conversion process described in Section II.

The resulting PRISM module of the Query of Purchase Process modeled as the state machine in Figure 4 is presented in Listing 1.

<table>
<thead>
<tr>
<th>Failure Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Blocking</td>
<td>Blocks the continuity of work of the testing and maintenance team.</td>
</tr>
<tr>
<td>Critical</td>
<td>Causes inconsistencies or data losses, affecting the functioning of various parts of system.</td>
</tr>
<tr>
<td>Normal</td>
<td>General system errors that do not render the system outage.</td>
</tr>
<tr>
<td>Minor</td>
<td>Inaccurate messages or errors with very minor or no impact on the system functioning.</td>
</tr>
</tbody>
</table>

### TABLE II

<table>
<thead>
<tr>
<th>Component Reliabilities</th>
<th>Transition Probabilities</th>
</tr>
</thead>
<tbody>
<tr>
<td>B.Action</td>
<td>( T_{\text{query}} )</td>
</tr>
<tr>
<td>E.DAOQuery</td>
<td>( T_{\text{query-loop}} )</td>
</tr>
<tr>
<td>E.Request</td>
<td>( T_{\text{delete}} )</td>
</tr>
<tr>
<td>E.Request</td>
<td>( T_{\text{update}} )</td>
</tr>
<tr>
<td>E.Request</td>
<td></td>
</tr>
<tr>
<td>E.Request</td>
<td></td>
</tr>
<tr>
<td>E.Request</td>
<td></td>
</tr>
<tr>
<td>E.Request</td>
<td></td>
</tr>
<tr>
<td>E.Request</td>
<td></td>
</tr>
<tr>
<td>E.Request</td>
<td></td>
</tr>
</tbody>
</table>

In Listing 1, we declare the constants for the component reliabilities in the Query state machine from line 2 to 6. Note that the failure state modeled in Figure 4 as state 7 (line 26) in module QueryPurchase. Two
important points of note are as follows. Firstly, lines 19 to 24 in Listing 1 are labeled with `query` and `query_end` actions. This is used for synchronizing actions between modules, allowing the modules to make transitions simultaneously according to their respective guard conditions.

Secondly, guards of these commands may refer to variables from other modules, used to indicate that a module starts only once the boolean condition has been satisfied. Listing 1 illustrates this situation, where the boolean variable `query_flag` in line 25 is used to indicate that any of the modules succeeding the query start only once that variable is true. Therefore, that variable indicates the module reached the end state successfully.

The final model synthesized in PRISM contains 58 states and 128 transitions. In PRISM simulation environment, we make extensive runs of the synthesized model in order to check whether it consistently expresses the behavior modeled in the UML specification of the Seplag web application.

![State Machine Representation of the Query of Purchase Process](image)

**Fig. 4. State Machine Representation of the Query of Purchase Process**

As previously mentioned, the architectural view of Seplag was restricted to a layered strategy. But a clear component-view of each layer content was not initially provided. As a result, it initially required us an investigation of the implemented code in order to extract the components. On that regard, there were classes with multiple purposes, where we identified a high degree of coupling. This feature made difficult the clear identification of components (modularization). For example, `ControlManagementPurchaseProcessRecord` is related to five other classes: `ActionDeletePurchaseProcess`, `ActionManagementPurchaseProcess`, `ValidatorPurchaseProcess`, `ManagerPurchaseProcess` and `PurchaseProcess`. Then, the aid of software engineers became fundamental to successfully conclude our experiment and particularly filtering out components and their failures types accordingly.

### IV. Qualitative and Quantitative Analysis

#### A. The Qualitative Analysis

This analysis focuses on the effort to apply our reliability analysis technique on the Seplag portal. Major challenges to apply our approach inherents those of architecture-based reliability techniques in general, discussed below.

First of all, as for the system usage profile, we extracted this information directly from the database. This step was facilitated due to the scope of the CRUD operations of the use case. The significance of the extracted information can be reasoned by the fact that all the 88,000 processes in the database of the Seplag system were registered via this CRUD use case.

As for the system components, the portal was implemented through different frameworks such as Struts [2], Spring [35] and Hibernate [31]. Thus, some classes analysed were dependent on their design (e.g. the Action component of Boundary layer stem from the application of Struts). Consequently, the problem of components with heterogenous nature [29] and distinct abstraction levels and granularity was observed during our study.

Following Equation 1, the Seplag reliability is 99.6%. Note that we only consider blocking or critical failure as other bugs would not deviate the system from its correct service execution [3].

The quantitative analysis of Seplag reliability is carried out in two parts. First, we compute the reliability of the software system in PRISM and compare with the actual system reliability. In order to estimate the actual system reliability, we use the same Formula 1, where, in this case, \( F = 21 \) is the number of system failure, considering blocking or critical failures types that were fixed after detected. \( N = 5,156 \) is the considered number of system execution, including test cases. We should point out that the number of system executions were derived from the number of inclusions, exclusions and queries of the processes registered on the database. In particular, the number of queries were an educated guess over the number of processes included and excluded. We assumed that, for each process included and excluded, a query was carried in order to verify the success of their operation.

Following Equation 1, the Seplag reliability is 99.6%. Note that we only consider blocking or critical failure as other bugs would not deviate the system from its correct service execution [3].

We follow the methodology presented in Section II to compute the Seplag model reliability in PRISM. The model consists of obtaining the system reliability as the probability of reaching the terminal state from the initial state [9]. In PRISM, we use the following PCTL formula to obtain the
Seplag system’s reliability:

\[ P = \mathbb{P}[F(\text{end})] \]  

(2)

Considering that we have the initial value of the reliability of each component, we obtained an approximate single reliability value for the Seplag system: 98.7%. This result considers the component reliability values and transition probabilities presented in Table II. Comparing the actual system reliability to the one computed using our methodology, the error is roughly 1%, which shows quite a high accuracy between computed and observed result.

One of the most useful ways to use our architecture-based methodology is to explore the PCTL query to compute and plot the values of system reliability obtained from the query by treating each individual component reliability value and transition probabilities as independent variables in order to identify those parts that have a higher impact on the overall system reliability. This is accomplished in the following section.

1) The Sensitivity Analysis: The most useful way to analyze the model and to gain insights into its dependability, mainly reliability, is to compute and plot the values as some parameters are varied, i.e. to perform a sensitivity analysis. Plotting the value of system reliability while varying the reliability of the components will reveal which components have higher impact on the overall system reliability. Therefore, focusing on enhancing the reliability of those components with proper allocation of project resources will be paramount to enhance the reliability of the overall system.

We structure the sensitivity analysis in two parts. Firstly, we consider the general scenario without taking into account any particular system usage profile. Secondly, we conduct sensitivity analysis based on the most frequent AAL system usage profile. For the quantification of the analysis we use the same PCTL reachability statement in Equation IV-B.

According to this statement, we want to obtain the probability to reaching the final end state without failing. We have the reliability of the components as input parameters and vary the reliability one component at a time, from 0 to 100% and fix the other components to 100%. Note that, for this analysis, we only consider the long-run system reliability, i.e. we do not take into account the amount of time it takes for the system to reach its successful end. The outcome of this first part of the sensitivity analysis is depicted in Figure 5.

The Y-Axis in Figure 5 represents the reliability of the system, while the X-axis represents the reliability of a component. The analysis includes six components of the Seplag web system. We order the legend from top to bottom, from the least to the most significant component.

According to the plotted results, the steeper the slope for a component, the more significant is its impact on system reliability. This way, the most critical components are the Control and Process, while the Request and Query are the least critical. Indeed, this can reasoned by the fact that Control and Process are central to the Control and Entity layers respectively. Also, this result shows the Seplag system relies on these two components for this part of the system we analyzed, showing high coupling between these components with the rest of the components. Indeed, as we are analyzing the operations with the process itself, the Process component plays an important role in this system. Therefore, this design decision is consistent with the outcome of this first sensitivity analysis.

The results in this analysis have shown to be potentially useful for the testing team in particular. Their tests are mostly oriented by use-cases execution and does not provide a straight relation between the developed components and their impact on the overall system reliability.

2) Analyzing the Sensitivity of the Transition Probabilities: The second part of our sensitivity analysis aims at focusing on the scenario transitions impact. The mapping of the information of the influential transition is modeled as the outgoing transitions, presented in Figure 2.

We should note that, for the scenario transitions sensitivity analysis we had to use lower values for the components reliability to 90%. Otherwise, the variation of the transition probabilities would not be noticed. Figure 6 depicts the outcome of our analysis. From the mentioned Figure, we notice that \( T_{\text{update}} \) and \( T_{\text{delete}} \) transitions present a steeper curve, compared to \( T_{\text{query}} \).

Considering the CRUD operations, those that play a major role in the communication with the database are indeed the operations of creating, updating and deleting. In our case, the operations of updating and creation were described by the same scenario, according to the development team of Seplag. Therefore, it is natural to conclude that \( T_{\text{update}} \) and \( T_{\text{delete}} \) would be the transitions with higher impact on the system reliability. This outcome is, therefore, consistent with the results obtained through the sensitivity analysis.

C. Threats to Validity

1) Construct Validity: Construct validity concerns establishing correct operational measures for the concepts being studied. According to the methodology (Section II), the PCTL property assessing the system’s reliability relies on the reliability of its components and on the probability of transitions among modules. In the example analyzed (Section III), log failure data was used to calculate component reliabilities, and the transition probabilities was calculated on the basis of the
number of database records. We focused on critical or blocking failures.

2) Internal Validity: Internal validity concerns establishing a causal relationship, whereby certain conditions are shown to lead to other conditions. We noticed that the variation of the reliability of individual components and their probabilities of transition do impact the final results computed using PRISM. In particular, the sensitivity analysis well illustrates this impact and the identified critical components correspond to their key role in the architecture. However, since we carried out an analysis by means of an example, in which control of variables is limited, further empirical assessment is necessary for validating the proposed method.

3) External Validity: External validity concerns establishing the domain to which a study’s findings can be generalized. In particular, the conversion phase could be applied to other domains, whereas the analysis phase is domain specific. DTMC was used for system modeling; despite assuming discrete-time probability input values, we noticed that the model parametrization in PRISM and the action annotations used to synchronize PRISM modules could be applicable in other contexts. Indeed, in previous work we showed the applicability of the method in the Ambient-Assisted Living domain [34]. Nevertheless, we are aware that the current study was based only on one single system—albeit an industrial-strength one—and that there is a myriad of more complex systems in different domains. At the moment we are planning to conduct further empirical studies to investigate the generalization of the applicability of this approach.

4) Conclusion Validity: Conclusion validity concerns whether it is possible to draw correct conclusions from the results, e.g., reliability of the results. First, according to Section IV-B, the method is sound within the accuracy of 1%. Second, in particular sensitivity analysis showed that improving component reliability has a positive impact on system reliability and also to which components system reliability is more sensitive (e.g., Process and Control). This is consistent with the design of placing such components at the core of the system’s architecture. Nevertheless, since this is an exploratory case study, in which control of variables is limited, further empirical studies should be carried out to further assess these claims.

5) Repeatability: Repeatability concerns demonstrating that the operations conducted in a study can be repeated with the same results. We expect that replications of our study should offer results similar to ours. Indeed, the characteristics of the specific system architecture may differ from the one used in the current study, but the underlying reference architecture should remain unchanged.

V. RELATED WORK

Software architecture provides a means to achieve the system qualities over the software life cycle [4]. Nevertheless, the architecture, by itself, is unable to achieve qualities [4]. Methods for software architecture evaluation such as SAAM [21] or ATAM [20] analyze it to show the satisfaction of certain properties. An analysis of these methods was performed in [10].

The identification of architecture components was approached by DiLucca et al. through a reverse engineering technique to reconstruct UML diagrams providing distinct views of the web applications [26]. Belletini et al. use class and state diagrams obtained through static and dynamic web application analysis to support the test case generation [5].

From the perspective of existing web logs there are some emerging results [32]. However, the focus of their analysis is not on the higher abstract level of software components. Some approaches to reliability assessment of web applications do exist [1]. However, they differ from our work in various points: the accuracy of the model estimation, consideration to current web development technologies and identification of critical components.

There are several work by Ghezzi group related to probabilistic model checking for reliability analysis [11], [13], [12], [15]. Most of their work have focused on run-time/dynamic analysis, including monitoring requirement properties on their KAMI approach [11] as well as optimization techniques for evaluating the satisfaction of reliability requirements at run time [13]. Particular to the latter, they consider both design and run time verification. A more recent work, KAMI has been weaved into the QoSMOS approach, where a set of tools, including PRISM have been weaved into the realization of a comprehensive dynamic QoS management in service-based systems [8]. Our approach for dependability analysis is complementary to their approach in the sense that our scenario-based requirements approach translated into DTMC models at design time and therefore, can be straightforwardly integrated. Moreover, in this work we focus on a component-based view of the web system activities (comparatively to services) so that our analysis, particularly the sensitivity analysis, may directly reflect the impact of the components failure on the overall system reliability.

In previous work, Rodrigues et al. applied the architecture-based reliability analysis used in this work on the realm of Ambient-Assisted Living domain [34]. The focus on that work consisted on exploring critical issues on the reliability of AAL system prior to implementation, when no component reliability or system usage is known. In this work, we have focused on
exploring the technique’s accuracy as well as the feasibility of applying the technique on current industrial-strength web applications, where the components clear definition (and respective failure) is not usually provided. Also, Rodrigues et al. [33] proposed a Message Sequence Chart (MSC) based reliability prediction technique and had the technique automated by using the Label Transition System Analyser Tool (LTSA) [22]. Though it was a sound work, the LTSA tool was not originally designed for probabilistic model checking, such as PRISM. On the other hand, PRISM does not provide the scenario-based modeling facilities such as the MSC plugin provided by LTSA [36]. This study explored the potential of the PRISM tool using a model-driven approach to accurately conduct a component-based reliability analysis in the complex realm of developing multi-layered web applications.

VI. CONCLUSION AND FUTURE WORK

The purpose of this work was to explore the feasibility of applying our previous work [34] to the domain of multi-layered web applications, as a real-life case study. Additionally, applying our methodology showed that such analysis can highlight gaps in the software development cycle, particularly on the software architecture definition and the clear component separation. Despite the limitations, the application of our technique showed quite an accurate result for the estimated web application reliability: relative error of roughly 1%, compared to real data. Also, we carried out a sensitivity analysis in this study to identify system components that have the highest impact on the dependability of the web application. The outcome of this analysis may be of great use, for instance, to enhance the quality of tests with respect to prioritizing most critical components.

For future work, we plan to scale the quantitative analysis to various large web systems. One such example could be the analysis of the various other parts that constitute the web application we used in this work. Also, we plan to analyse other specific web technologies (e.g. AJAX) that has changed the traditional web development paradigm from multipage to single page applications. Finally, we also plan to evaluate the potential of our technique to augment the quality of test cases from the sensitivity analysis results, since it reveals those components that are relevant from the system reliability perspective.

ACKNOWLEDGMENT

We would like to acknowledge the Synergia Lab at DCC-UFMG and, particularly, Clarindo Padua without whom none of our work would be realized.

REFERENCES


