




ORIGINAL RESEARCH

Bayesian Network analysis of software logs for data-driven software maintenance

Santiago del Rey¹  | Silverio Martínez-Fernández¹  | Antonio Salmerón² 

¹Universitat Politècnica de Catalunya, Barcelona, Spain

²Department of Mathematics & Center for the Development and Transfer of Mathematical Research to Industry (CDTIME), University of Almería, Almería, Spain

Correspondence

Santiago del Rey
Email: santiago.del.rey@upc.edu

Funding information

Junta de Andalucía, Grant/Award Number: P20-00091; AEI, Grant/Award Number: PID2019-106758GB-C32/AEI/10.13039/501100011033; Spanish project, Grant/Award Number: PDC2021-121195-I00; Spanish Program, Grant/Award Number: BEAGAL18/00064

Abstract

Software organisations aim to develop and maintain high-quality software systems. Due to large amounts of behaviour data available, software organisations can conduct data-driven software maintenance. Indeed, software quality assurance and improvement programs have attracted many researchers' attention. Bayesian Networks (BNs) are proposed as a log analysis technique to discover poor performance indicators in a system and to explore usage patterns that usually require temporal analysis. For this, an action research study is designed and conducted to improve the software quality and the user experience of a web application using BNs as a technique to analyse software logs. To this aim, three models with BNs are created. As a result, multiple enhancement points have been identified within the application ranging from performance issues and errors to recurring user usage patterns. These enhancement points enable the creation of cards in the Scrum process of the web application, contributing to its data-driven software maintenance. Finally, the authors consider that BNs within quality-aware and data-driven software maintenance have great potential as a software log analysis technique and encourage the community to deepen its possible applications. For this, the applied methodology and a replication package are shared.

KEYWORDS

Bayes methods, software maintenance, software quality

1 | INTRODUCTION

Software quality management is a key activity for the maintenance of software systems. Indeed, the budget that companies invest in software quality management is estimated to increase in the next years [1]. Software systems can generate large amounts of data both from its internal state and from how it is being used. The maintenance of such systems can benefit from the analysis of the available data to get better insights of the system, thus, performing data-driven software maintenance. For instance, companies can analyse the user experiences of their software systems [2] in order to detect possible defects, such as performance downgrades or unexpected errors. To improve the user experience, it is key to understand the users' usage patterns when navigating a software system. Understanding these patterns enables companies the ability to enhance the design of

their software systems [3] as well as provide personalised information depending on the user [4]. In this regard, software logs are a popular resource to monitor.

Developers place Application Programming Interface calls to a logging framework (e.g., SLF4J [5] or Log4j [6]) to record events that describe the internal state of the system as it runs in production. The analysis of software logs has been proved useful in improving system comprehension [7, 8], performance analysis [9, 10], finding usage patterns [11, 12], and anomaly detection [13, 14]. Nevertheless, this analysis is not free from its own challenges [15].

Regarding software logs analysis, the introduction of Bayesian Networks (BNs) based on behaviour data to improve performance and users' experience has not been studied yet in real-life contexts. This is an opportunity that motivates our research, as BNs have turned out to be a useful tool to achieve

This is an open access article under the terms of the Creative Commons Attribution License, which permits use, distribution and reproduction in any medium, provided the original work is properly cited.

© 2023 The Authors. *IET Software* published by John Wiley & Sons Ltd on behalf of The Institution of Engineering and Technology.

actionable and interpretable results [16] and have been previously proposed in other SE areas [17]. The potential advantage of BNs with respect to other software logs analysis techniques is twofold. On one hand, BNs can be regarded as predictive models like classification trees or artificial neural networks, turning out to be competitive in terms of accuracy while remaining significantly less costly in terms of model size and learning time, which makes them especially appropriate in high-dimensional problems with large volumes of data. On the other hand, BNs can also be employed as a means of extracting knowledge from data. This is achieved by encoding the relationships between the variables in the network structure in such a way that just by exploring the structure of the learnt model, it is possible to know if two variables are relevant to one another in a given context with no need to carry out any numerical analysis and regardless of the values of the estimated parameters. In addition, we find that they provide very intuitive results, which can be easily understood by non-experts compared to other approaches proposed in the literature [18, 19]. This can be beneficial in the industry where there can be members without the knowledge of BNs. Moreover, by using BNs, we are able to process a large amount of log data at a relatively low computational cost compared to other techniques, such as classification trees or Deep Neural Networks.

Therefore, to enable interpretable data-driven software maintenance activities like managing application performance and usage patterns, we propose the BN analysis of software logs. More specifically, we explore a data-driven construction of BNs, rather than expert-based ones, as they are scarce but possible with the huge amounts of behaviour data available. To fix the scope of the study, we focus our research on the context of log analysis in web applications. We hypothesise that BNs constitute an appropriate tool for modelling and analysing web performance and usage pattern issues due to their probabilistic and temporal capabilities. More precisely, we propose the use of BNs for representing the relationships between the variables analysed. In this way, the uncertainty related to the interaction between the variables is naturally represented by the factorised probability distribution encoded by the network.

In this context, we have designed and conducted an action research study in a web application project to study the effectiveness of BN analysis of software logs for (a) evaluating application performance and (b) discovering usage patterns. To this aim, we create three models: (i) performance model, (ii) error model, and (iii) temporal model.

In this paper, we show the potential of using BNs for software log analysis to detect performance deficiencies, errors, and usage patterns. The main contributions of this work are:

- We propose and show the use of BNs for software logs analysis to detect performance deficiencies and errors in a web application. With our approach, we are able to detect a set of enhancement points, such as pages with irregular load times or unexpected exceptions.
- We propose and show the use of BN analysis of software logs to discover new usage patterns in a web application. By doing so, we detect recurrent patterns that led to the improvement of the user interface.

- We propose and show the usefulness of BN analysis of software logs for data-driven software maintenance. Specifically, we show several Scrum cards created after performing the BN analysis.
- A replication package of our study, whose objective is twofold: to enable the reproducibility of the used BNs and to support other researchers and practitioners to apply this technique in other web applications.

This paper is structured as follows. Section 2 describes related work. Section 3 presents the research methodology, consisting of an action research, and a process to analyse software logs with BNs. Section 4 presents the results of the action research, together with the validation of the Bayesian models, and the deployment into Scrum cards. Section 5 discusses the results of the study and lessons learnt. Section 6 reports threats to validity. Finally, Section 7 concludes the paper and anticipates future work.

2 | RELATED WORK

In this section, we describe related work regarding managing software quality and Bayesian analysis in Software Engineering (SE).

2.1 | Managing software quality based on software logs

Software quality assessment can be performed at almost all the phases of software development – from project inception to maintenance – and at different levels of granularity – from source code to architecture. For data-driven software quality and maintenance, software logs are a good source of data due to their capacity to inform about the state of a system. Indeed, log analysis in web applications has grown in popularity among the SE research community [20]. Current research ranges from log analysis for performance and error detection to usage patterns detection.

Log analysis for performance. Logging is a well-known and extended practice to obtain different types of performance metrics. For instance, Nagaraj et al. propose DISTALYZER, an automated tool to support the developer for the investigation of performance issues in distributed systems. They use machine learning (ML) techniques and the log data available to present a set of patterns to help developers to detect the root cause of the performance issues [9]. Syer et al. propose an automated approach using hierarchical clustering that combines performance counters and execution logs to diagnose memory-related performance issues [21]. Chow et al. use a causal model to analyse the end-to-end performance in distributed systems by observing log traces [22].

Log analysis for error detection. As web applications grow in size and popularity, the number of errors occurring could also grow due to a higher diversity of their usage (e.g., different devices and locations). With thousands or millions of users accessing every day, it is of utmost importance that developers

can quickly locate these errors and identify their source to reduce the fixing time as much as possible. One way to achieve this is to analyse the different logs that a web application generates. However, log files tend to grow rapidly in size and contain irrelevant information making their manual exploration an impossible task. For this reason, it is important to find other methods that can extract for us the information needed without much intervention. Indeed, this type of analysis has already been explored for other topics like anomaly detection. Du et al. propose DeepLog, which utilises a Long Short-Term Memory deep neural network to model a system log as a natural language sequence [18]. Similarly, Zhang et al. utilize an attention-based Bi-LSTM model to capture the contextual information in the log sequences and automatically learn the importance of different log events. In this way, they are able to handle unstable log events and sequences for anomaly detection [23]. Yang et al. propose PLELog, a semi-supervised approach to log-based anomaly detection to get rid of time-consuming manual labelling and incorporate the knowledge of historical anomalies via probabilistic label estimation. PLELog is robust against unstable logs via semantic embedding and detects anomalies by leveraging the attention mechanism [19].

Log analysis for usage patterns detection. Understanding how users make use of a web application can be very useful in the elicitation of new requirements. By detecting their usage patterns, developers can detect multiple enhancement points, such as unused or popular features and error-inducing sequences of actions. Wang et al. propose the use of a compact path traversal graph to represent the web navigation patterns and an efficient graph traverse algorithm to find throughout-surfing patterns [24]. Gadler et al. use Hidden Markov Models to mine use logs and automatically model the use of a system [25]. Barifah et al. explore the hidden usage patterns of a large-scale digital library by analysing log files and classifying them by using K-Means [26]. Pettinato et al. propose a semi-automated method for reconstructing sequences of tasks of a system [27]. Liu et al. analyse log registries of an appstore-service to understand usage patterns with respect to aspects, such as app popularity, app management, and device-specific preferences [12]. Jindal et al. propose a pruning-based Markov model called Pruned all- K th Modified Markov Model (PKM3) for predicting web navigation sequences [28].

de Sousa et al. provide an overview of 20 years of research in BNs applied to software project management [29]. They claim that most of the studies using BNs are performed in a laboratory environment rather than in real contexts. Moreover, to the best of our knowledge, there are no studies on how to analyse software logs to detect multiple software quality deficiencies while providing intuitive results. To address these issues, we propose the use of BNs for log analysis to detect performance issues, errors, and common usage patterns, leveraging usage data and system logs of a web application already in production. Ultimately, we want to provide a tool to accurately detect multiple software quality deficiencies for large volumes of data that is easy to understand for technical and non-technical users.

2.2 | Bayesian network analysis in software engineering

Bayesian Networks [31] are a statistical technique able to represent Bayesian hierarchical models. Besides the quantitative modelling of the relationship between the variables under analysis, BNs provide interpretable qualitative information about the model structure. The Bayesian approach usually comes along with flexible analysis techniques that yield results that are intuitive and easy to interpret. The importance of properly interpreting the results of statistical analysis is a topic of interest within the statistical community, especially given the misuse of statistical concepts like the p -value when making decisions [32].

Bayesian Networks are factorised representations of the joint distribution over a set of random variables. Therefore, they can be used in a wide variety of tasks, being one of the most prominent ones in the classification task. From the point of view of BNs, classification consists in determining the most probable value of a so-called *class* variable, given that the values of some predictive variables are known. For instance, in a web application, one can be interested in predicting whether a page is going to show problems when loading given some information about the user, like web browser, device, location etc. The performance of BNs for classification, also called BN classifiers, has been compared to other popular ML techniques in terms of different measures, like the Area Under the Receiver Operating Characteristic Curve and the H-measure [33].

Bayesian Networks have already been applied to some aspects of SE. Examples are the exploration of the relationships between software metrics and defect proneness (distinguishing between the most and less effective metrics on defect proneness) [34], defect causal analysis [35], and code quality [36]. Other studies have focussed on assessing software quality, particularly considering the prediction of quality strategic indicators [37], and the requirements engineering phase [38–40]. They have also been used as a tool for making decisions in evidence-based [17] and value-based [41] SE. In a recent study, Russo et al. use Bayesian analysis to study the relationship between software engineers' personality traits and their tendency to engage in and enjoy effortful thinking. In another recent study, Britto et al. use Bayesian analysis to evaluate the impact of bug priority, code-churn size in bug fixing commits, and links between bug reports on the bug-fixing time [42].

In a previous work by Hassan et al., BNs were used to learn and predict page categories visited in first N positions, type of visit (short or long), and rank of page categories visited in N first positions [30]. We complement their study by applying BNs not only to make user behaviour predictions but also to assess application performance and detect errors.

We summarize related works regarding the application of BNs in software log analysis in Table 1. In this study, we propose the use of BNs to perform such analysis. We will approach this goal through an action research oriented to improving the software maintenance process of a web application using BNs. In this paper, we introduce two novel

TABLE 1 Summary of relevant related works to our study.

| Study | Open data | Analysis approach | Description of the log analysis approach | Aim of the study |
|-------|-----------|---|--|--|
| [9] | No | Statistical tests and dependency networks | Extract a small set of features from logs. Then, use statistical tests to identify features that most distinguish two sets of logs. Next, use dependency networks to learn the joint distribution among the statistics and the performance. Finally, combine the last two steps to automatically identify a set of interrelated variables that most diverge across the logs and most affect overall performance. | Performance issues detection |
| [22] | No | Causal models | Generate a causal model from a collection of logged events that are first transformed to a collection of segments. The model looks for three types of relationships: Happens-before, mutual exclusion, and pipeline. Then, perform several performance analyses on the model: Find the critical path, quantify slack for segments not on the critical path, and detect performance anomalies | Performance issues detection |
| [19] | Yes | Semi-supervised learning | First, obtain a semantic embedding of the logs in three steps: Log parsing, word embedding, and TF-IDF-based aggregation. Then, use probabilistic label estimation to estimate the labels of unlabelled log sequences. Finally, build an anomaly detection model with an attention-based Gated Residual Unit neural network and train it with the estimated labels. | Anomaly detection |
| [30] | No | Bayesian networks | Learn Bayesian models by using maximum likelihood estimation on the training data. To predict short and long visit sessions train a Naive Bayes classifier. To find patterns in the sequence of pages visited, the range of number of page views, and the ranking of the pages visited train standard Bayes classifiers. | Web navigation patterns detection |
| [25] | No | Hidden Markov models | Build an Iteratively generated HMM (IHMM) by adding one hidden state at each iteration, starting from a state containing all the event symbols. Two approaches: An Interactively generated IHMM, in which at each iteration a practitioner is asked for the symbols to add from a given list and an automatically generated IHMM where the top k θ -interesting sequences generated by the current IHMM are automatically selected. | System usage modelling |
| Ours | Yes | Bayesian networks | Obtain a small set of features from multiple log sources. Then, train a Naive Bayes classifier to predict performance issues and use a standard BN to detect errors. To detect usage patterns, first, transform the data into a sequence of actions. Then, train a BN to detect the usage patterns. | Performance issues, error and usage patterns detection |

Abbreviation: TF-IDF, Term Frequency-Inverse Document Frequency.

aspects: (i) the data-driven construction of BNs from software logs data and (ii) their use to reason about multiple quality requirements like performance efficiency and usability.

3 | RESEARCH METHODOLOGY

In this section, we describe the goal and Research Questions (RQs), the study design, the data collection and preparation procedures, and the steps to construct BNs for analysing software logs.

3.1 | Goal and research questions

To define the goal of our empirical study, we used the Goal Question Metric approach [43]. Therefore, we firstly specified the goal of the study, then traced those goals to the questions that lead us to know if we meet those goals, and finally provide

a set of efficiency metrics that answer those questions. The goal of our study is to:

Analyse BNs for software logs for the purpose of maintaining software systems with respect to its usefulness for detecting performance problems, unexpected errors, and usage patterns from the point of view of the developers in the context of a web application.

Bearing this goal in mind, we define two RQs:

- **RQ1: Is the BN analysis of software logs effective to detect performance deficiencies and errors in a web application?**

By performance deficiencies, we refer to the quality requirement of performance efficiency, that is, performance relative to the number of resources used under stated

conditions. More precisely, we refer to the time behaviour sub-category, defined as the degree to which the response and processing times and throughput rates of a product or system, when performing its functions, meet the requirements [44].

With **RQ1**, we want to explore if BNs can help in small-sized software projects in answering questions about performance deficiencies and error proneness since these are two key quality attributes when maintaining a web application due to their implications in the user experience, and in the end, in user retention. To answer this question, we measure the performance by using the page load time metric, which is divided into three categories: high, medium, and low. In addition, we also measure the effectiveness of BN analysis to identify the likelihood of a type of error in the web application and the causing class.

- **RQ2: Is the BN analysis of software logs effective to detect users' usage patterns in a web application?**

By detecting usage patterns, we aim at assessing the quality requirement of usability, that is, the degree to which a product or system can be used by specified users to achieve specified goals with effectiveness, efficiency, and satisfaction in a specified context of use. Specifically, we intend to evaluate the learnability sub-category, defined as the degree to which a product or system can be used by specified users to achieve specified goals of learning to use the product or system with effectiveness, efficiency, freedom from risk, and satisfaction in a specified context of use [44].

With **RQ2**, we want to study the use of BNs to help practitioners detect anomalous usage patterns in web applications. For example, it is not uncommon for users to follow unexpected or rare sequences of actions, under different contexts (e.g., device and language) leading to errors in the application. For this purpose, we created a system to collect usage and error data to help us detect sequences of actions leading to the error page as well as identify the most common patterns of usage. To measure how well our findings fit the previous question, we manually explore the generated model to find patterns that can help us answer this question.

3.2 | Study design: action research

At the inception of this study, we discussed the appropriate empirical research method. We chose to design an action research study, given the realistic research setting of our web application [45]. Action research is the “empirical research that investigates how an intervention, like the introduction of a method or tool, affects a real-life context” [46]. Its application has been successful in other SE studies [47, 48]. Please note the main difference between action research with case study research, as in the former one, the investigator takes an active role in the case. As defined by Wohlin [49], “A case study is an empirical investigation of a case, using multiple data collection methods, to study a contemporary phenomenon in its real-life context, and with the investigator(s) not taking an active role in the case investigated”.

3.2.1 | Context: a web application

In this action research, we investigate how to improve the software qualities of a web application named ChessLeague [50]. We selected this case because it is developed and maintained by our research group and we can indeed conduct data-driven software maintenance (i.e., applying the intervention of using Bayesian analysis of software logs to manage performance and anomalous usage patterns). Chessleague is an online chess game that simulates team competitions where the users become managers of a chess club [51, 52]. Players need to perform tasks, such as the management of their team lineup or the buying and selling of chess players, to accumulate points throughout the season to win the league. Currently, it has over 200 daily users.

The web application is built using Java EE 6 and is deployed in a GlassFish 4.1.2 server. It is structured as a three-layer application, where the presentation layer is developed using Java Servlet Pages, the business layer is implemented using Enterprise Java Beans, and the database layer uses MariaDB v10.4.19. This technology stack is a common choice for enterprise software [53].

The web application is developed and maintained following the Scrum methodology where we have sprints of 1 week.

3.2.2 | The action research cycle

Inside an action research project, we find different actors involved. There is an action team, who is responsible for planning, executing, and evaluating the research. In our study, this team was composed of the first two authors. The first author took the roles of both practitioner and researcher, while the second took the role of a researcher. There is a reference group that is responsible for the advice and feedback for the action team. This group was composed of the third author.

Each action research cycle has five phases [45]. In Figure 1, we can see an overview of the structure of this cycle in our study. As inputs to the action research cycle, we have the goals we wanted to achieve first as developers of the application and second as researchers. In the same way, the outputs of the action research project represent what we obtained after applying the five phases during one cycle. We describe our action research cycle below following the existing guidelines [45, 54]:

1. *Diagnosing the problem:* Since the action team included the application developers, the diagnosing phase was completely oriented towards the improvement of the application from a software quality point of view. Thus, the action team identified the need of improving multiple software quality requirements, such as performance efficiency and usability (anomalous usage patterns). In other words, the developers in the action team wanted to use log analysis to detect performance deficiencies, run-time errors, and usage patterns inside the web application to improve the software quality. As a result, we defined our goal and RQs, which have already been described in Section 3.1.

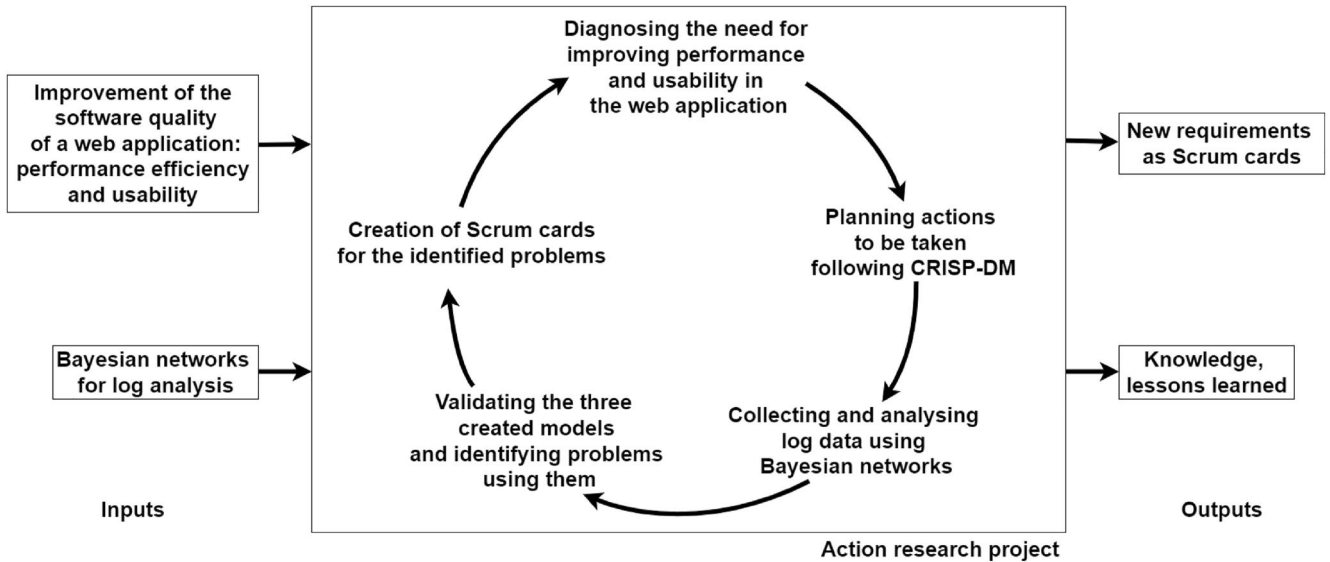


FIGURE 1 Action research in our study (adapted from [45]).

2. **Planning actions:** To cope with the diagnosed problem in the web application, we first started by reviewing the work that had been done in the fields of software quality management based on logs and the BN analysis for SE. After this step, we considered that what we intended to achieve with this study had not been done yet. We planned our actions following CRISP-DM [55], which has been used in SE projects [56]. First, we needed to understand the needs of the product. This was already solved in the previous phase. Second, we planned to collect the log data and understand it to be able to extract knowledge from it. Third, we needed to process the data to avoid interferences such as missing values or redundancy and to construct new features from it. Fourth, we needed to apply the BN analysis to the processed data to create three data-driven models. Fifth, we planned to validate the models created by using cross-validation and evaluating the accuracy. Finally, we intended to deploy the interpretations from the models into the development process.
3. **Taking actions:** As stated in the planning, we first started by collecting the data. For this, we used three different sources that will be described in Section 3.3. After the data collection, we proceeded to its preparation in order to extract the relevant features for our study. Finally, we produced three BNs to analyse the data obtained to discover novel insights.
4. **Evaluating actions:** To evaluate our results, we took two actions. First, we validated two of the three networks by measuring their quality using their accuracy estimated by cross-validation. For the third one, we manually explored it to search for useful patterns for our goal. Second, we interpreted the BNs, finding out unknown performance anomalies. From these insights, we derived actionable Scrum cards.
5. **Learning and theory building:** After conducting the action research in three sprints, the action team appreciated the value of BN analysis of software to exploit implicit

feedback and improve the software quality. The team identified a set of problems that were turned into cards in a Scrum sprint [57]. The developers of the web application decided to conduct this software quality management periodically to discover and address performance efficiency and usability issues.

Section 3.1 describes the first phase (diagnosis). Phase 2 (planned actions) is described in Sections 3.3, 3.4, and 3.5. Finally, phases 3, 4, and 5 are described in Section 4.

3.3 | Data collection procedure and dataset

In order to cope with the aforementioned goal of this study, we collected data from three different sources. First, we used the usage logs to get the application performance and to see how the application is being used. Second, we used Google Analytics (GA) behaviour data to extend the previous information in order to gain better insights into what causes performance deterioration. For this, we registered the browsers and devices used by the users as well as the country from where they connected. Third, we used the logs provided by the server to get information about the errors occurring in the application. The former one was created from scratch to carry out this study and the two latter were available before this action research.

Usage logs: To collect the usage data, we first defined the list of variables needed to answer both RQ1 and RQ2. To keep the usage logs simple and not overload them, we chose to collect a reduced number of variables (see Table 2), which provided us with all the necessary information to answer the questions presented in Section 3.1. Since we were interested in user interaction and page performance, we registered when a page started and finished loading as well as some of the actions a user could do on a page.

Google Analytics behaviour data: GA offers a great number of metrics and behaviour data from which we selected a small relevant subset shown in Table 3. This data was then used to enrich the usage logs to detect the causes of performance deficiencies.

Server logs: The server itself provided us with logs generated at runtime, which allowed us to answer the second part of RQ1 by analysing the errors thrown by the application. As in the previous case, we chose a subset of the variables we considered relevant for our goal (see Table 4). Due to the multiple formats used inside the server logs, we used Logstash [58] to extract the relevant data and write it to a CSV file, which we later used to build the models.

3.4 | Data preparation procedure

Before creating the different models, we needed to prepare the raw data. To this end, we created a group of Python scripts (available in the replication package) that have been evolving

through various iterations. These scripts were used to clean the raw log data and engineer new variables. Moreover, we did not use any type of random data during the study.

During the first iteration, we started by cleaning the usage data. We performed the following cleaning activities. First, duplicated rows were dropped in order to avoid redundancy. (ii) Usernames were turned into a lower case since the same user could appear with its username in upper and lower cases. (iii) The user was unknown until it logged into the application, so we back-propagated the username to those rows that we knew that belonged to a user. Once we had cleaned the data we proceeded to construct new variables using the usage data. The variables created were: day of the week, page load time, time on page, and action duration. The variables representing durations were obtained by grouping the data per session and calculating the time difference with the previous row. The last step of the first iteration was to integrate the usage data with the GA data.

During the second iteration, we modified the scripts to create new variables that we thought could affect the page load time. To this aim, we created a new dataset where we pivoted

TABLE 2 Variables obtained from the usage logs.

| Variable | Description | Type | # Levels | Values (examples) |
|------------|---|-----------|----------|---------------------------------------|
| Timestamp | Instant when the action is being done | Timestamp | – | 2021-03-18T12:46:00.216Z |
| Language | Language used to visualize the page | Nominal | 4 | es, en, fr, de |
| Username | User that does the action | String | – | Santiago |
| Session ID | Session ID where the action is being done | String | – | 56693043c07c6016237def2f0620 |
| Page | Page where the action is done | Nominal | 47 | Bid, Cron, Help, Play, Team, Start... |
| Action | The action that is being done | String | – | Load page |

TABLE 3 Relevant variables obtained from Google Analytics (GA).

| Variable | Description | Type | # Levels | Values (examples) |
|-----------------|---|-----------|----------|--|
| Session ID | Session ID where the action is being done | String | – | 56693043c07c6016237def2f0620 |
| Timestamp | Instant when the action is being done | Timestamp | – | 2021-03-18T12:46:00.216Z |
| Country | Country of the connection | Nominal | 12 | Spain, Portugal, Germany... |
| Browser | Browser used by the user | Nominal | 10 | Chrome, Firefox, Safari, Edge, Samsung Internet... |
| Device category | Category of the device used | Nominal | 3 | Desktop, mobile, tablet |

TABLE 4 Relevant variables obtained from the server logs.

| Variable | Description | Type | # Levels | Values (examples) |
|-------------|---|-----------|----------|---|
| Timestamp | Instant when the log entry was registered | Timestamp | – | 2021-03-18T12:46:00.216Z |
| Severity | Severity of the entry | Ordinal | 3 | SEVERE, WARNING, INFO |
| Message | Detail of the entry | String | – | Servlet.service() for servlet jsp threw exception |
| Thread name | Name of the thread where the incidence occurs | String | – | http-listener |
| Error | Error that has occurred | String | – | java.lang.NullPointerException |
| Class | Class that throws the error | String | – | org.apache.jsp.ErrorPage_jsp |
| Method | Method where the error has occurred | String | – | _jspService |

the server logs using the error type and then computed the number of errors every 5 seconds. This grouping was done due to the difficulty of relating an error to a particular user since the only common value they had is the timestamp. In addition, we used the usage logs to compute the number of pages accessed every 30 seconds. The final step of the second iteration was the integration of the new data with the dataset from the first iteration. This gave us an approximation of the number of errors that were happening for every row. Additionally, we took the opportunity to change the granularity of some variables that were too fine-grained, such as the user-name or the action.

3.5 | Construction of Bayesian networks for analysing software logs

To facilitate the use of BN analysis in other web applications or other software applications in general, we provide a detailed explanation of how we built the BNs in our study.

In what follows, we will use uppercase letters to denote random variables and lowercase letters to denote a value of a random variable. In this context, the random variables will be the variables of interest in our action research. For instance, *page load time*, *time on page*, and *action duration* (see Section 3.4). Boldfaced characters will be used to denote sets of variables. The set of all possible combinations of values of a set of random variables \mathbf{X} is denoted as $\Omega_{\mathbf{X}}$. A BN [31] with variables $\mathbf{X} = \{X_1, \dots, X_n\}$ is formally defined as a directed acyclic graph with n nodes where each one corresponds to a variable in \mathbf{X} . Attached to each node $X_i \in \mathbf{X}$, there is a conditional distribution of X_i , given its parents in the network, $Pa(X_i)$, so that the joint distribution of the random vector \mathbf{X} factorises as

$$p(x_1, \dots, x_n) = \prod_{i=1}^n p(x_i | pa(x_i)), \quad (1)$$

where $pa(x_i)$ denotes a configuration of the values of the parents of X_i .

An example of a BN representing the joint distribution of the variables $\mathbf{X} = \{X_1, \dots, X_5\}$ is shown in Figure 2. It encodes the factorisation

$$p(x_1, x_2, x_3, x_4, x_5) = p(x_1)p(x_2|x_1)p(x_3|x_1)p(x_5|x_3)p(x_4|x_2, x_3).$$

One advantage of BNs is that the factorisation simplifies the specification of large multivariate distributions that are replaced by a set of smaller ones (with a lower number of parameters to specify). For example, the factorisation encoded by the network in Figure 2 replaces the specification of a joint distribution over 5 variables with the specification of 5 smaller distributions, each one of them with at most 3 variables. Another advantage is that the network structure describes the interaction between the variables in the model, in a way that can be easily interpretable, according to the d -separation criterion [31]. As an example, the structure in Figure 2 determines

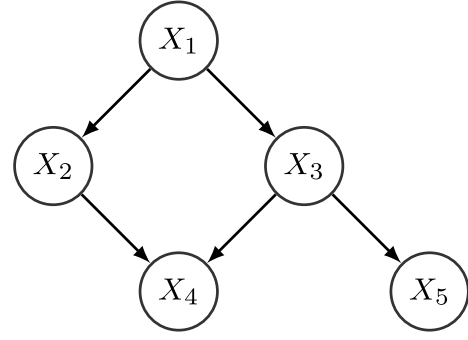


FIGURE 2 An example of a Bayesian network (BN) structure with 5 variables.

that variables X_1 and X_5 are independent if the value of X_3 is known, and likewise, X_2 and X_3 are independent if X_1 is known.

From the perspective of our study, by using BNs, we will be able to model the probability distribution of the variables of interest, given all the other available variables, and use it to make predictions (for instance, predicting the load time of a given page with the network in Figure 3). Besides the prediction capabilities, we will also be able to represent the interaction between the variables that determine that an error happens (see, for instance, Figure 4).

The process of building a BN from data involves two tasks, namely learning the network structure and then estimating the conditional distributions corresponding to the selected structure. Assuming that all the variables in the network are discrete or qualitative, the estimation of the conditional distributions can be obtained from the relative frequencies in the data of each combination of possible values of the variables involved, which correspond to the so-called *maximum likelihood estimation*. The structure learning task is usually approached as an optimization problem, where the space of possible network structures is traversed trying to maximize some score functions that measure how accurately a given structure fits the data. More precisely, we have used the Bayesian Information Criterion (BIC) score, which is a typical choice in the literature [59], defined as

$$BIC(M|D) = \sum_{l=1}^N \ln p(\mathbf{x}_l | \hat{\boldsymbol{\theta}}) - \frac{1}{2} d \ln N, \quad (2)$$

where $D = \{\mathbf{x}_1, \dots, \mathbf{x}_N\}$ is the dataset, M is the network under evaluation, and $p(\mathbf{x}_l | \hat{\boldsymbol{\theta}})$ is the joint distribution corresponding to network M with parameters $\hat{\boldsymbol{\theta}}$ estimated by maximum likelihood. The idea of using the BIC score is to obtain networks that fit the data accurately, but at the same time, give preference to simple networks. That is why the number of parameters, d , necessary to specify the probability distributions in the network, is used as a penalty factor. Other popular choices are the Akaike Information Criterion (AIC) and Bayesian Dirichlet equivalent (BDE) scores (we refer the reader to [59] for their detailed definition).

Alternatively, it is also possible to choose a given network structure beforehand and only estimate the conditional

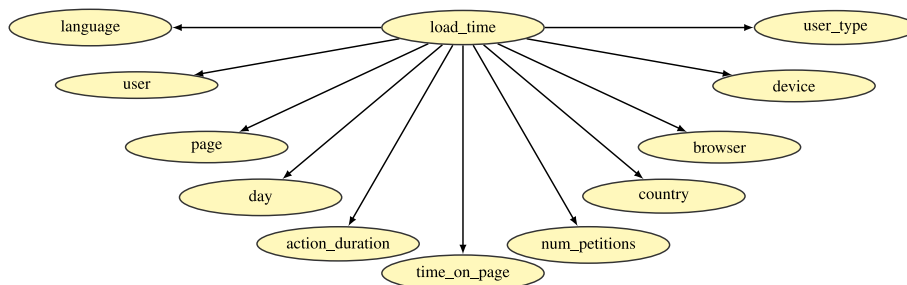


FIGURE 3 Structure of the performance network.

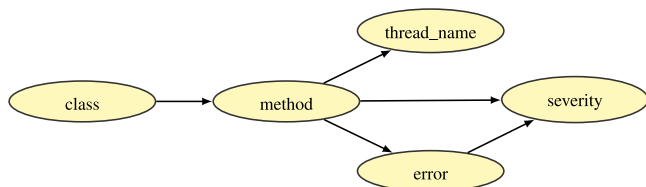


FIGURE 4 Structure of the error model learnt using the Bayesian Dirichlet equivalent score.

distributions from data. This is particularly interesting if the network is going to be used for prediction purposes, where one is interested in the value of a target variable rather than in the interactions between the other variables. An example of such a fixed structure is the so-called Naive Bayes (NB) [60], where the variable whose value we want to predict is the root of the network and the only existing links go from that variable to the rest of the variables in the network (see Figure 3 for an example of such structure).

Adopting an NB structure means a strong independence assumption (all the variables are conditionally independent given the root variable C), but in practice, it is compensated by the low number of parameters that need to be estimated from data. Notice that, in this case, the factorisation encoded by the network results in

$$p(c, x_1, \dots, x_n) = p(c) \prod_{i=1}^n p(x_i | c), \quad (3)$$

meaning that n one-dimensional conditional distributions must be specified instead of one n -dimensional conditional distribution.

The process we have adopted to build the BNs for software logs is shown in Figure 5. Specifically, it involves the following steps:

- Step 1: Collect the performance and error data from the different sources and process it, if necessary, to extract the relevant variables for the study and to integrate the different datasets.
- Step 2: Load the data in the R statistical software [61] and select the relevant variables for the model, and then manually discretise the variables that need custom thresholds or values. Next, transform all character values to

factors and all integers to numeric using, for example, the *dplyr* [62] package. Finally, use the discretise function from the *bnlearn* [63] package to discretise all numeric values.

- Step 3 (needed for temporal models): To simulate a temporal model, use the *fold_dt* function from the *dbnR* [64] package. However, when using this function, keep in mind that to correctly simulate the sequence of actions we need to previously have the data ordered in a descending action order.
- Step 4: Learn the network structure from data, using the function *hc* from the *bnlearn* package, and then use the *bn.fit* function to estimate the conditional distributions corresponding to the learnt structure. The function *hc* performs a hill-climbing search over the space of possible network structures, attempting to optimise a score function as in Eq. (2).
- Step 5 (needed for NB model validation): Randomly split the data into K folds and go to step 7.
- Step 6 (needed for model comparison): We can use the *bn.cv* function from the *bnlearn* package to obtain the Log-Likelihood loss, which we can use to compare the networks obtained from using different scores.
- Step 7 (needed for NB models): If instead of learning the network structure we want to force the model to have an NB structure, we can use the function *naive.bayes* from the *bnlearn* package and then estimate the corresponding conditional distributions using the *bn.fit* function on $K - 1$ folds.
- Step 8 (needed for NB models): We can use the model learnt in step 6 to predict the performance of the data in the fold reserved for testing. For this, we use the *predict* function from *bnlearn*, which returns the corresponding performance for each row. Then, we can compare the true labels with the predicted ones and assess the model quality.
- Step 9 (needed for NB models): Repeat steps 7 and 8 for each of the K folds.
- Step 10 (optional): We can plot the network structure using the *graphviz.plot* function from the *Rgraphviz* [65] package. Also, we can use the *bnlearn* package to write the network to a BIF, NET, DSC, or DOT file and then explore it with a tool like Hugin [66].

The replication package, structured following CookieCutter Data Science [67], is available on <https://doi.org/10.5281/zenodo.6823268>.

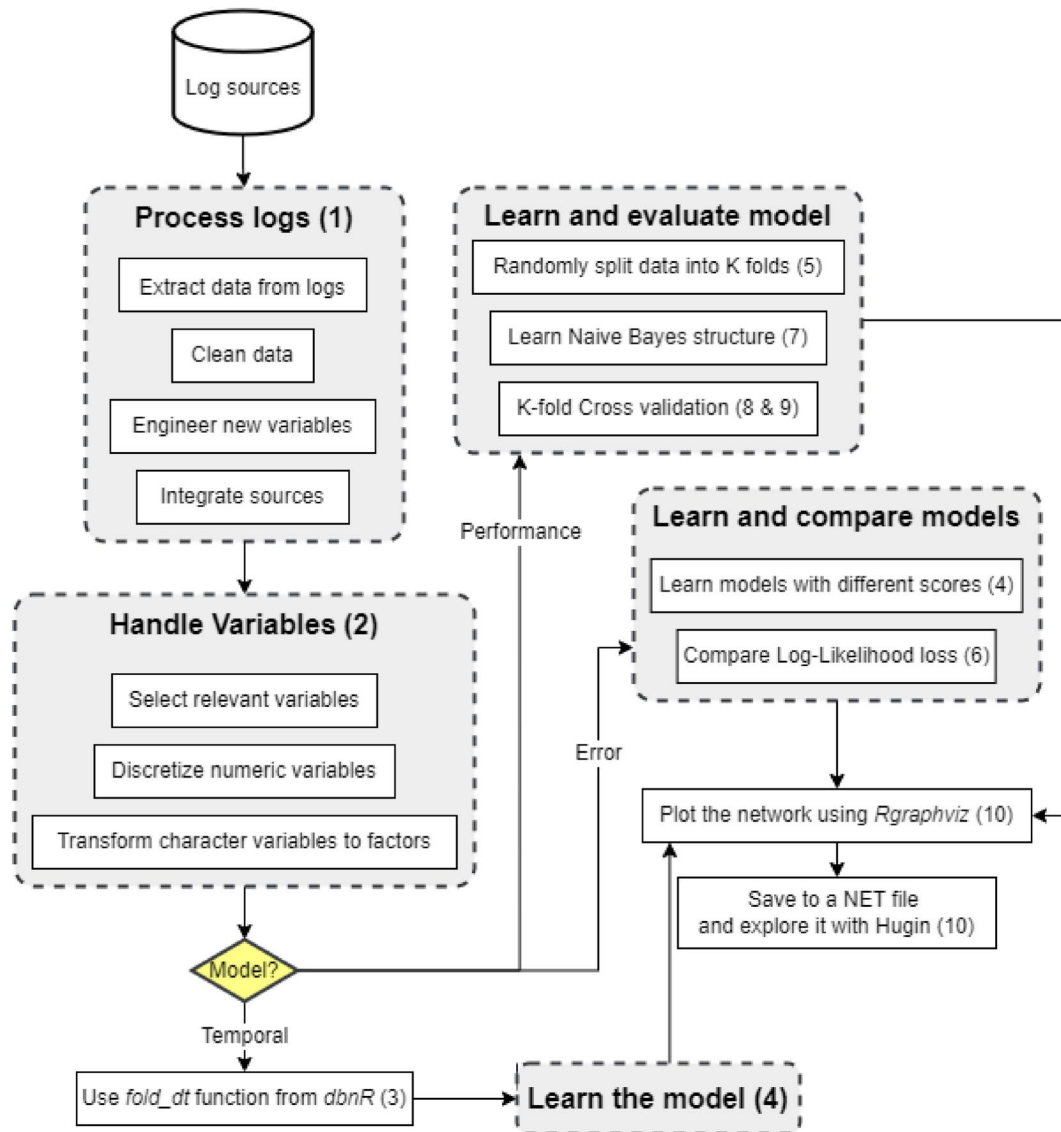


FIGURE 5 Overview of the steps we take for the creation of the Bayesian Networks (BNs). Numbers in brackets refer to the steps described in Section 3.5.

4 | RESULTS AND INTERPRETATION

In this section, we give a brief description of the process we followed to create the models (applying the steps defined in Section 3.5) used to answer each RQ and analyse the results. We, respectively, answer RQ1 and RQ2. In Figure 6, we show a summary schema of the study.

4.1 | RQ1: Is Bayesian network analysis of software logs effective to detect performance deficiencies and errors in a web application?

4.1.1 | Performance model

We created a model to analyse how the variables we had collected affected the load page time. For this, we created a BN following the procedure described in Section 3.5. We first

converted all the variables to factors or numerical values depending on their type. Then, we discretised the numerical variables since we wanted to know when their values exceeded the performance thresholds that the action team had set rather than their particular value. Among the discretised variables, we highlight three of them: *load_time*, *time_on_page*, and *action_duration*. For the *load_time* and *action_duration*, we used the following categories: optimal, low, medium, and high. And for the *time_on_page*, we used the categories: very low, low, medium, and high.

After learning the network structure, we noticed that contrary to our first thoughts, *action* was not connected to any other variable. We suspected this due to the high granularity of the variable since it captured too specific actions. We also realised that *the user* and *page* were not related, which went against our initial thoughts. To solve these problems, we decided to group fine-grained actions into more general actions and to group users into two

categories depending on their role: admin and not admin. Finally, we detected that the variables representing time measures were not distributed in a balanced way since almost all of their values fell below the first level (see Figure 7a,b).

We used the second iteration to solve all the aforementioned problems. In particular, we redefined the thresholds for the distribution of the variables *load_time* and *action_duration*. To define the new intervals, we decided to use the values of the median and the third quarter as reference points. The new distributions can be seen in Figure 7c,d. Additionally, two more variables were added: the number of errors and weekly sessions per user. To discretise the number of errors, we used the following categories: none, low, medium, and high. The

weekly sessions per user were used to divide the users into the following categories: occasional, regular, active, and very active.

As a result of this second iteration, we obtained the network shown in Figure 3. Since we wanted to analyse the application performance and how all the variables affected it, we decided to use an NB structure, which focusses on a variable of interest, in this case, the *load_time*, forcing the creation of a tree structure with the *load_time* as the root and the other variables as its children. With this structure, we can then fix the value of *load_time* and see how the probabilities in all the other variables change. In the same way, we can fix the value for one or more leaf nodes (e.g. *action_duration*, *page*, ...) and see how these combinations affect the value of the *load_time*.

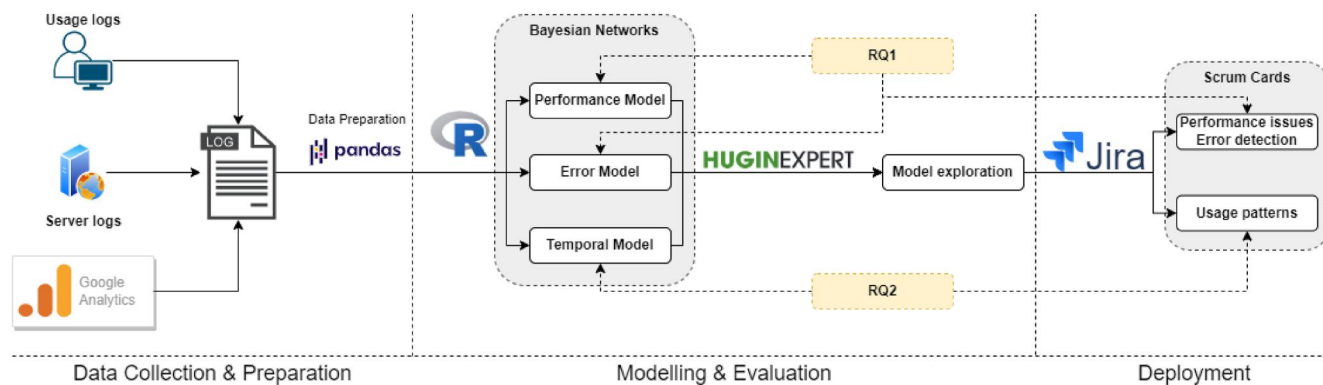


FIGURE 6 Schema of the empirical study.

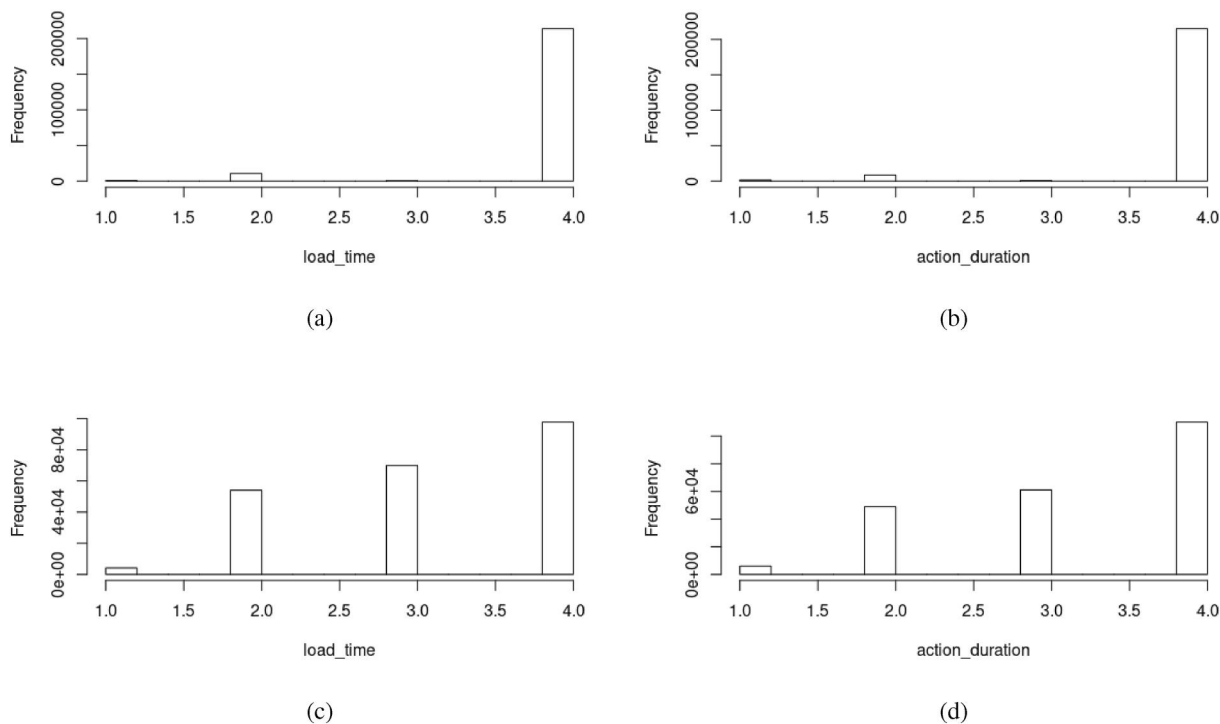


FIGURE 7 Data distribution for the features *load_time* and *action_duration*. (a) and (b) are the distributions obtained from the first iteration, while (c) and (d) are the distributions obtained after the second iteration. The categories are represented as follows: high (1.0), medium (2.0), low (3.0), and optimal (4.0).

In order to validate the accuracy of the model, we performed 10-fold cross-validation and computed the average accuracy. We obtained an average accuracy of 0.75. The best model reached an accuracy of 0.76. We show the confusion matrix of the best model in Figure 8 after column normalisation. We see how the model achieves a True Positive Rate over 75% on the *low* and *optimal* instances. However, its performance downgrades on instances belonging to the *high* and *medium* classes. Moreover, they tend to be mislabelled as *low*. In Table 5, we show the per-class accuracy. Here, we see how the model achieves very good values of accuracy for each class. In particular, we observe that the model is able to discern instances belonging to the *high* class very well. This is critical in our context since we are most interested in detecting pages with high load times to find the web application bottleneck. Note that we do not have a base model to compare with in terms of accuracy. We attempted to build a classification tree [68] using R package *rpart* [69], but after several hours of computation, no result was obtained. This shows how the BN approach can efficiently process huge amounts of data more efficiently than other common approaches.

4.1.2 | Error model

The purpose of this model was to analyse the errors occurring in the server so that it allowed us to locate the classes where the error was most probable to be thrown and their main cause. To this end, we used the following variables from the server logs: error, class, method, severity and thread name.

To learn the model structure, we used four different scores: AIC, BDE, log-likelihood, and BIC. The reason to use four

scores was to compare the different structures generated and check their similarities to select the most appropriate one.

The most similar structures are the ones learnt using AIC and BDE as can be seen in Figures 4 and 9. In both structures, the root is the *class*, which has a direct dependency with the *method*, as we would normally expect, and then the *method* forms a divergent connection with *error* and *thread_name*, and additionally with the *severity* in the BDE. This implies that once we know the method, the class is not relevant for knowing which error has been thrown as well as the severity, or the thread name. The main difference between these two structures is that for the one learnt with AIC, the error, and the severity become independent once we know the method. However, for the structure learnt using BDE, it is the error and the thread name that are independent when knowing the method, and the error and severity are always influencing each other.

Figure 10 displays the structure learnt using the log-likelihood score. In this case, the overall structure resembles the two previous ones. However, instead of having the *class* as the root, we have the *error*, which directly influences the *class*. Then, we have that the remaining variables (i.e. *method*, *severity*, and *thread_name*) are d-separated by the *class*. This implies that once we know the *class*, these three variables become independent.

The most dissimilar structure is the one learnt using the BIC score. As shown in Figure 11, there is a dependency relation between the error and the method that throws it as we would expect. The interesting thing is that the method does not directly imply the class where the error is thrown, but conditions the error severity, and it is this that gives us information about the class throwing the error as well as the thread name.

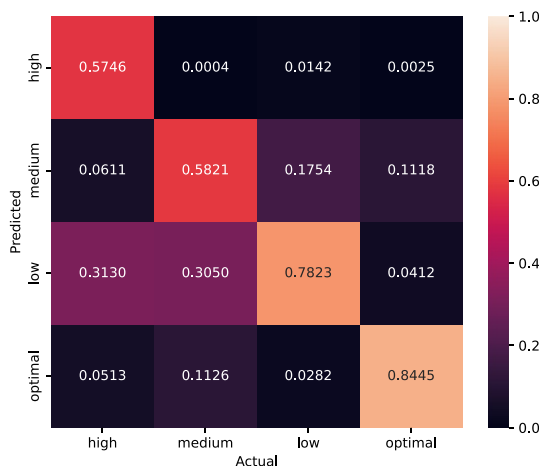


FIGURE 8 Confusion matrix for the load time normalised by columns.

TABLE 5 Per-class accuracy.

| | High | Medium | Low | Optimal |
|----------|-------|--------|-------|---------|
| Accuracy | 0.987 | 0.796 | 0.836 | 0.896 |

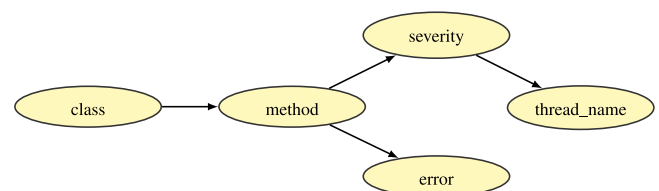


FIGURE 9 Structure of the error model learnt using the Akaike Information Criterion score.

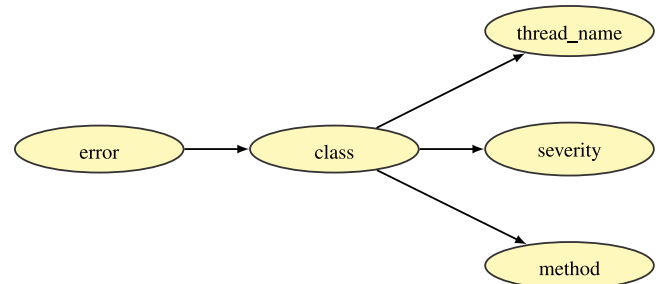


FIGURE 10 Structure of the error model learnt using the log-likelihood score.

In order to select the most appropriate network, we used 10-fold cross-validation to compare how well each of them fitted our data. As seen in Table 6, the network using the BIC score was the one performing worst among the four. The other three obtained very similar results. Our final decision was to use the network learnt with BDE since it performed slightly better than the others.

4.1.3 | Deployment

Once the two models were created and validated, we incorporated them into the agile development process of the web application. During the duration of one sprint [57], we used the models to increase the software quality of the application. The models were presented to the action team through the Hugin software. By exploring and interpreting the resulting models, the action team discovered several issues in the web application. As a result, eight Scrum cards were created from the use of these models.

From the performance model, we created cards, such as the improvement of the performance of the Start, Market, and ViewOffers pages, since they were the most probable pages to suffer from bad performance. With the error model, we

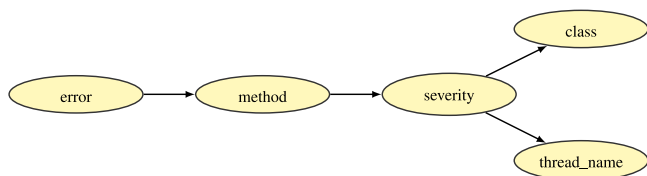


FIGURE 11 Structure of the error model learnt using the Bayesian Information Criterion score.

TABLE 6 Log-Likelihood loss of the four error models created obtained from cross-validation.

| | AIC | BDE | Log-likelihood | BIC |
|---------------------|-------|-------|----------------|-------|
| Log-likelihood loss | 1.145 | 1.144 | 1.146 | 1.183 |

Abbreviations: AIC, Akaike Information Criterion; BDE, Bayesian Dirichlet equivalent; BIC, Bayesian Information Criterion.

detected several unexpected exceptions. For example, we created a card (Figure 12) to solve the NullPointerExceptions appearing in the ErrorPage.

Answer to RQ1. With the creation of the two data-driven BN models described in this section, we have been able to efficiently detect multiple points of bad performance in the application. By using the performance model, we are now able to predict the web application performance, with an average accuracy of 0.75, based on the collected data. Furthermore, the manual exploration of the error model has enabled us to detect the most frequent errors in the application and its sources.

4.2 | RQ2: Is Bayesian network analysis of software logs effective to detect users' usage patterns in a web application?

4.2.1 | Temporal model

The purpose of this model was to analyse the behaviour of the users inside the application throughout multiple temporal instants. In particular, we wanted to get an idea of the sequence of pages the users visit inside the application so that we could discover specific patterns leading to errors or performance deficiencies.

To keep the model simple and easy to understand at first sight, while having the necessary data to fulfil our goal, we decided to select a small subset of variables from the usage logs. The variables selected were: page, page load time, time on page, and number of sessions per week. The latter was used to define the type of user depending on how many times they accessed the application during the week, which was then assigned to each row belonging to the corresponding user.

The first step was to discretise the numerical variables using the same intervals as in the performance model. The

Resolve NullPointerException originating from ErrorPage

Attach Add a child issue Link issue

Description

As a developer

I want to reduce the number of errors originating from the error page

In order to improve the quality of the application.

Acceptance criteria:

- The number of `NullPointerException` within the `ErrorPage` has to be reduced by 70%.

Done Done

Pinned fields

Estimated effort 5

Details

Assignee SR Santi del rey

Labels None

Development Branch

Reporter SR Santi del rey

FIGURE 12 Jira card extracted from the analysis of the error model.

next step was to create the variables representing the different temporal measures. Given that the number of events collected per session varied from one another, we decided to consider all the available data as a unique session and replicate this data to represent the different instants. For this, we used the R package *dbnR* [64]. We only need to specify the number of temporal measurements and the package performs the data replication as many times as we specify. Note, however, that the generated replications are not identical, but instead, they are shifted by one row as described in Table 7. Hence, to have the accesses correctly arranged, we had to reorder the dataset by session ID, which ensured that all consecutive accesses belonged to the same session, and then by the descending order of access to ensure that after the replication, we had the correct sequence of actions.

Finally, we learnt the network structure using the Hill-Climbing method, which resulted in the network seen in Figure 13 in the appendix. As we can see, the network has a clear structure. On one hand, we see how all the variables of the same type follow a serial connection in their natural order, except for the load time that is reversed. On the other hand, we see how the page creates a divergent connection with the rest of the variables in the same temporal instant; this means that, for the same temporal instant, the variables *load_time*, *time_on_page*, and *user_type* are conditionally independent given *page*.

The two connections mentioned above give rise to a series of convergent connections, where all the variables, except for the page, from *t1* onward work as an intermediate node connecting the same variable in the previous instant with the page that works as the nexus in the divergent connection. It should be noted that this process is reversed for the load time. This implies that whenever we know the value of an intermediate node or any of its children, both their parents become dependent. In Figure 14, we can see an example of one of these convergent connections. In this example, the variables *time_on_page_t_0* and *page_t_1* are independent unless we

know the value of *time_on_page_t_1* or any of its children, in which case they would become dependent.

4.2.2 | Deployment:

As with the models described in Section 4.1, we deployed this model into the development process for the duration of one sprint. Again, we presented the model to the action team who used the Hugin software to explore and interpret the model. After this process, the action team was able to create four Scrum cards that were incorporated into the Product Backlog. These cards referred to the detection of common usage patterns and anomalous patterns previously unknown. For example, we detected that the most accessed page after logging in was the Results page. Thus, we added a new card (Figure 15) to include the results of the last match on the Start page. Also, we detected a short sequence of pages, starting from Offer-Player that led to the error page. Consequently, we created a new card to investigate this issue.

Answer to RQ2. With the creation of the data-driven temporal BN model, we have been able to explore the different sequences of pages the users follow inside the application. By manually exploring the model, we have detected several sequences of pages that end in the error page, which have proved useful to detect errors in the application. Furthermore, we have been able to detect common patterns of usage, which helped us to understand how the application was being used.

TABLE 7 Example of the replication of a table using *dbnR* to represent three temporal instants.

| access_order | | |
|--------------|--|--|
| 6 | | |
| 5 | | |
| 4 | | |
| 3 | | |
| 2 | | |
| 1 | | |

Replication ↓

| access_order_t_0 | access_order_t_1 | access_order_t_2 |
|------------------|------------------|------------------|
| 4 | 5 | 6 |
| 3 | 4 | 5 |
| 2 | 3 | 4 |
| 1 | 2 | 3 |

5 | DISCUSSIONS

As seen in this action research, BNs have great potential to become powerful analytic techniques in data-driven software maintenance. Although there is a need to explore the models manually, if done by a domain expert (e.g., development or operations team members), they can provide valuable insights into the application that can help in the decision-making process in agile practices. Another strong point of BNs as an analytic technique is their capability to represent sequences of actions over time, which can help us discover usage patterns. This is especially useful when looking for sequences of actions leading to errors since software testing can only help in the measure of what the developers expect to be the possible sequences of actions a user can make. However, users often take developers by surprise by trying to use the application in unexpected ways. It is in situations like this that BNs are useful since they represent these anomalous behaviours, enabling developers to analyse such patterns to perform data-driven software maintenance. As an example, in this action research,

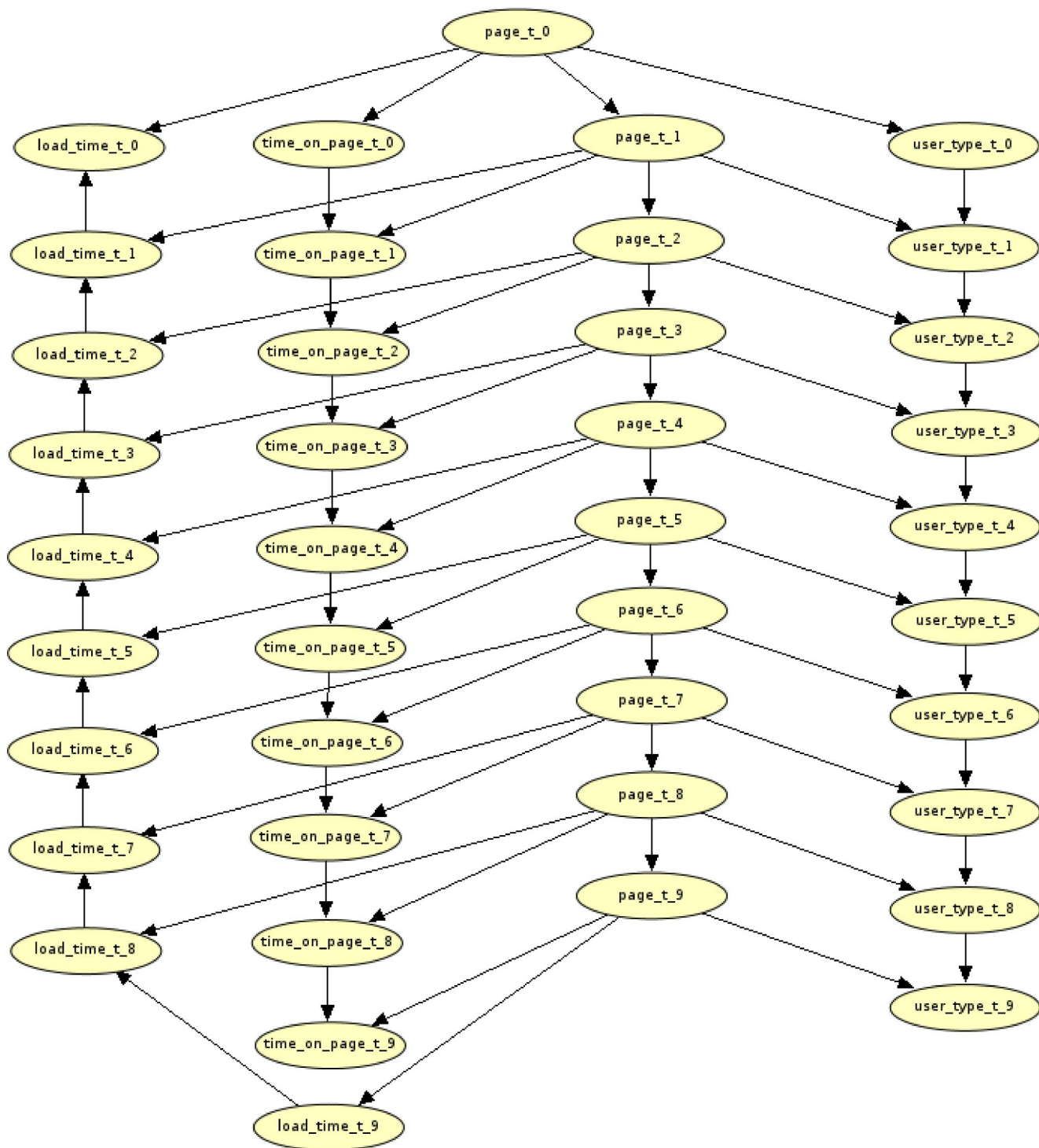


FIGURE 13 Structure of the temporal model.

we exploited this characteristic to improve the user interface of the Start page and the performance of several pages.

In previous works, we have seen other approaches for detecting performance issues, errors, and usage pattern behaviours' leveraging software logs, such as semi-supervised learning [19], causal models [22], or dependency networks [9]. However, all these approaches only focussed on detecting

one of the three problems mentioned. In this work, we have observed the flexibility that BNs provide for solving multiple problems of the same domain without requiring additional effort apart from the need for data preparation, which is required for any other approach. Furthermore, we agree with Hassan et al. in that BNs provide more intuitive results compared to other more complex approaches [30]. This allows

results obtained from BNs to be more easily interpreted by non-expert users, such as stakeholders, which can be beneficial in the industry. Moreover, we observe that our performance model obtains very similar accuracy to the one used by Hassan et al. for detecting long and short sessions [30]. Both try to model how much time is spent in the web application although at different levels of granularity, ours at the page level, and theirs at the session level. By modelling the time at the page level, we are able to detect bottlenecks in the web application.

Also, we would like to discuss what is needed for a software team to apply the BN analysis of software logs for data-driven maintenance of web applications. First, we consider that there must be a domain expert who can interpret the analysis results and their implications in the development process. Second, we believe that knowledge in data processing (e.g., data cleaning) is of utmost importance since is one of the key factors when creating a reliable model. Third, it would be advisable that at least one member of the team is familiar with BNs to understand their structure and how to analyse them. However, if this is not possible we find that the book by Nagarajan [70] is a good starting point to understand and implement a simple BN in R to have a base model running.

Finally, we would like to give our thoughts about the action research methodology. After applying it to our small project, we consider that this methodology fits quite appropriate for small-sized projects that want to combine research and agile development methodologies. Since both share a cyclic structure that repeats over several iterations, it becomes easy to adapt one to the other. This brings the possibility of

introducing small research teams that communicate with the development and management teams to improve different aspects of the development process, such as the introduction of new techniques or tools to increase productivity.

6 | THREATS TO VALIDITY

This section explores the main threats to validity of the action research. As global mitigation action, we have considered in our study the Association for Computing Machinery SIG-SOFT Empirical Standards [46]. In particular, we have ensured to comply with all the 12 essential specific attributes for action research. In Section 3.2, we cover the first 6 attributes (e.g. describe the context). We also explain the different phases throughout this work as explained in Section 3.2. Attributes from 8 to 11 are explained in Section 4. Finally, in this section, we cover the last attribute by exploring the possible threats to validity of our work. In addition, we have ensured to avoid the four anti-patterns that apply to action research (e.g., forcing interventions that are not acceptable to participants).

Construct validity: To avoid investigating only one aspect of the usefulness of BNs for software logs analysis, we apply the BN analysis to three different problems: performance issues detection, error detection, and usage patterns detection. Moreover, we meticulously defined our RQs and how we planned to measure our results to avoid vague definitions of the constructs.

Conclusion validity: To ensure the reliability of the analysis, the analysis procedure was documented in detail, including a replication package. Furthermore, for the BNs analysis, we involved three researchers, being one of them external to the software implementation of the web application. Also, to avoid low statistical power, we used three different software log sources. Each one adds new knowledge to our analysis.

Internal validity: The main threats remain in the web application used and the logging frameworks we used. To mitigate the first one, we tried to generalize as much as possible without forgetting that we worked with a single project that was not representative of the entire software industry. For the second one, we used popular libraries in the software

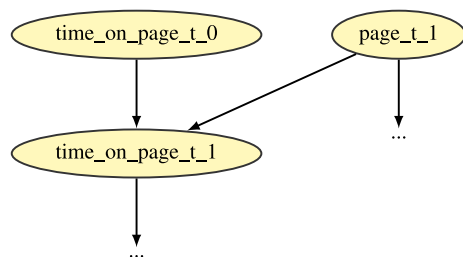


FIGURE 14 An example of a convergent connection.

Add last round results to the Start page

Attach Add a child issue Link issue ...

Description

As a user

I want to be able to see the results of my last round on the homepage

In order to see how the last round went without having to navigate to the results page.

Acceptance criteria:

- A table with the results of the user's last day must be displayed.
- The performance of the page must not decrease by more than 10%.
- The table must be available in all offered languages.

Done

Done

Pinned fields

Estimated effort 3

Details

Assignee SR Santi del rey

Labels None

Development Branch

Reporter SR Santi del rey

FIGURE 15 Jira card extracted from the analysis of the temporal model.

community. Furthermore, we identify two confounding factors: traffic generated by bots and the appearance of usage patterns due to an exploration phase. To mitigate the first one, we use a reCAPTCHA in the sign-up process and the users are required to access the web application using a password. The second confounding factor is mitigated by collecting data over several months. This gives the users enough time to learn how the application works and makes almost negligible the number of patterns generated due to an exploratory phase of the application. In addition, several errors and performance issues of the web application identified with the BNs were addressed as cards in our Scrum development. Therefore, we could explore the relationship between using this analysis with software quality. However, an in-depth study continuously updating and monitoring the BNs remains future work.

External validity: Although most of the data used in this study should be present in almost every web application, it differs greatly from one another. Consequently, the model structures resulting from this action research cannot be generalized to other applications since they have been learnt from the data generated by the application used in this study. Therefore, we described the setting of the action research and the process for creating the models. Our results are tied to the context of our web application. However, we prepared a replication package that can be executed in other web applications with a similar context, leading to new data-driven BNs to address performance efficiency and usability.

7 | CONCLUSIONS AND FUTURE WORK

We have explored the use of BNs as a log analysis technique to discover indicators of poor performance in a system and to explore usage patterns that cannot be found without a temporal analysis. We have done this in the context of a web application developed and maintained by our research groups. In this context, we have presented an action research where we used BNs as a technique to analyse software logs to improve the software quality. We show in a demo video, available on <https://youtu.be/X2bnexyIqZM>, examples of new cards in Scrum elicited by exploring and interpreting the BNs.

The results show that using BNs we can detect indicators of performance deficiencies and errors in a web application. We believe that the information gained from these models can have a beneficial impact on the data-driven software maintenance process by helping developers to detect users' usage patterns leading to errors in the application that could be hard to discover by other means (e.g., testing). Indeed, in this action research, the identified issues ended up in cards in the product backlog for the development process.

Although the preparation process for the construction of a BN requires considerable effort, mainly due to the need to generate, collect, and process the data that will be the base to learn the model, we consider that its benefits outweigh its disadvantages. For example, they can reveal hidden relations

between different variables that may not be apparent at first sight, which can help developers to better understand how the application behaves. Furthermore, we could use these networks to simulate different situations and observe how an application would perform, allowing the developers to detect issues that otherwise could remain hidden for a long time. For this reason, we believe that BNs have the potential to become powerful techniques to understand how an application operates and encourages the rest of the community, both researchers and practitioners, to deepen their possible applications in the SE field.

Moreover, we describe an action research in a small-size software project in order to improve the software quality of a web application. The application of action research was positive in this study, and we encourage others to perform it in small software projects as well.

Future work spreads in several directions. First, we plan to enhance the data collection with software quality data by static analysis of the code repositories. Then, we plan to study the correlation of these software quality metrics with usage errors from logs. We also plan to update the described networks with continuous monitoring and software log collection. Furthermore, we plan on studying the economic benefits and Return on Investment of applying this technique in the long term. Finally, we are working on hands-on materials to help practitioners to create and enhance the BNs for data-driven software maintenance, building on top of the replication package.

AUTHOR CONTRIBUTION

Santiago del Rey: Conceptualisation, Methodology, Data Curation, Formal Analysis, Investigation, Resources, Software, Validation, Visualisation, and Writing – Original Draft, writing – Review and Editing. **Silverio Martínez-Fernández:** Conceptualisation, Methodology, Funding Acquisition, Project Administration, Resources, Supervision, Validation, and Writing – Original Draft, writing – Review and Editing. **Antonio Salmerón:** Conceptualisation, Methodology, Funding Acquisition, Formal Analysis, Resources, Supervision, Validation, and Writing – Original Draft, writing – Review and Editing.

ACKNOWLEDGEMENTS

This research is funded by the Junta de Andalucía, grant P20-00091, the Spanish project PDC2021-121195-I00, and the “Beatriz Galindo” Spanish Program BEAGAL18/00064. This research is part of Project PID2019-106758GB-C32 funded by MCIN/AEI/10.13039/501100011033, FEDER “Una manera de hacer Europa” funds.

CONFLICTS OF INTEREST STATEMENT

All authors declare that they have no conflicts of interest.

DATA AVAILABILITY STATEMENT

We provide a demo video of the Bayesian analysis of software logs, available at <https://youtu.be/X2bnexyIqZM>. Additionally, we provide an open package in <https://doi.org/10.5281/zenodo.6823268>.

ORCID

Santiago del Rey  <https://orcid.org/0000-0003-4979-414X>
 Silverio Martínez-Fernández  <https://orcid.org/0000-0001-9928-133X>
 Antonio Salmerón  <https://orcid.org/0000-0003-4982-8725>

REFERENCES

- Oriol, M., et al.: Data-driven and tool-supported elicitation of quality requirements in agile companies. *Software Qual. J.* 28(3), 931–963 (2020). <https://doi.org/10.1007/s11219-020-09509-y>
- Feng, L., Wei, W.: An empirical study on user experience evaluation and identification of critical ux issues. *Sustainability* 11(8), 2432 (2019). Available from: <https://doi.org/10.3390/su11082432> <https://www.mdpi.com/2071-1050/11/8/2432>
- Verma, N., Singh, J.: Improved web mining for e-commerce website restructuring. In: 2015 IEEE International Conference on Computational Intelligence & Communication Technology, pp. 155–160 (2015)
- Moreno, M.N., et al.: Web mining based framework for solving usual problems in recommender systems. a case study for movies' recommendation. *Neurocomputing* 176, 72–80 (2016). <https://doi.org/10.1016/j.neucom.2014.10.097>
- QOS. CH: SLF4J (2022). <https://www.slf4j.org/>
- The Apache Software Foundation: Log4j – Apache Log4j 2 (2022). <https://logging.apache.org/log4j/2.x/>
- Wieman, R., et al.: An experience report on applying passive learning in a large-scale payment company. In: 2017 IEEE International Conference on Software Maintenance and Evolution (ICSME), pp. 564–573 (2017)
- Harty, J., et al.: Logging practices with mobile analytics: an empirical study on firebase. In: 2021 IEEE/ACM 8th International Conference on Mobile Software Engineering and Systems (MobileSoft), pp. 56–60 (2021)
- Nagaraj, K., Killian, C., Neville, J.: Structured comparative analysis of systems logs to diagnose performance problems. In: 9th USENIX Symposium on Networked Systems Design and Implementation (NSDI 12), pp. 353–366. USENIX Association, San Jose (2012). <https://www.usenix.org/conference/nsdi12/technical-sessions/presentation/nagaraj>
- Yao, K., et al.: Log4perf: suggesting logging locations for web-based systems' performance monitoring. In: Proceedings of the 2018 ACM/SPEC International Conference on Performance Engineering, pp. 127–138 (2018)
- GalOz, N., Gonen, Y., Gudes, E.: Mining meaningful and rare roles from web application usage patterns. *Comput. Secur.* 82, 296–313 (2019). <https://doi.org/10.1016/j.cose.2019.01.005>
- Liu, X., et al.: Understanding diverse usage patterns from large-scale appstore-service profiles. *IEEE Trans. Software Eng.* 44(4), 384–411 (2017). <https://doi.org/10.1109/tse.2017.2685387>
- Xu, W., et al.: Detecting large-scale system problems by mining console logs. In: Proceedings of the ACM SIGOPS 22nd Symposium on Operating Systems Principles. SOSP '09., pp. 117–132. Association for Computing Machinery, New York (2009). <https://doi.org/10.1145/1629575.1629587>
- Fu, Q., et al.: Execution anomaly detection in distributed systems through unstructured log analysis. In: 2009 Ninth IEEE International Conference on Data Mining, pp. 149–158 (2009)
- Oliner, A., Ganapathi, A., Xu, W.: Advances and challenges in log analysis. *Commun. ACM* 55(2), 55–61 (2012). <https://doi.org/recursos.biblioteca.upc.edu/10.1145/2076450.2076466>
- Mihaljević, B., Bielza, C., Larrañaga, P.: Bayesian networks for interpretable machine learning and optimization. *Neurocomputing* 456, 648–665 (2021). <https://doi.org/10.1016/j.neucom.2021.01.138>
- Tosun, A., Bener, A.: Bayesian networks for evidence-based decision-making in software engineering. *IEEE Trans. Software Eng.* 40(6), 533–554 (2014). <https://doi.org/10.1109/tse.2014.2321179>
- Du, M., et al.: DeepLog. In: Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security. ACM (2017)
- Yang, L., et al.: Semi-supervised log-based anomaly detection via probabilistic label estimation. In: 2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE), pp. 1448–1460 (2021)
- Cândido, J., Aniche, M., van Deursen, A.: Log-based software monitoring: a systematic mapping study. *PeerJ Computer Science* 7, e489 (2021). <https://doi.org/10.7717/peerj-cs.489>
- Syer, M.D., et al.: Leveraging performance counters and execution logs to diagnose memory-related performance issues. In: 2013 IEEE International Conference on Software Maintenance, pp. 110–119 (2013)
- Chow, M., et al.: The mystery machine: end-to-end performance analysis of large-scale internet services. In: 11th USENIX Symposium on Operating Systems Design and Implementation (OSDI 14), pp. 217–231. Broomfield, CO: USENIX Association (2014). <https://www.usenix.org/conference/osdi14/technical-sessions/presentation/chow>
- Zhang, X., et al.: Robust log-based anomaly detection on unstable log data. In: Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering. ESEC/FSE 2019, pp. 807–817. Association for Computing Machinery, New York (2019). <https://doi.org/10.1145/3338906.3338931>
- Wang, Y.T., Lee, A.J.T.: Mining web navigation patterns with a path traversal graph. *Expert Syst. Appl.* 38(6), 7112–7122 (2011). <https://doi.org/10.1016/j.eswa.2010.12.058> <https://www.sciencedirect.com/science/article/pii/S0957417410014211>
- Gadler, D., et al.: Mining logs to model the use of a system. In: 2017 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM). IEEE (2017)
- Barifah, M., Landoni, M.: Exploring usage patterns of a large-scale digital library. In: 2019 ACM/IEEE Joint Conference on Digital Libraries (JCDL), pp. 67–76 (2019)
- Pettinato, M., et al.: Log mining to re-construct system behavior: an exploratory study on a large telescope system. *Inf. Software Technol.* 114, 121–136 (2019). <https://doi.org/10.1016/j.infsof.2019.06.011>
- Jindal, H., Sardana, N.: Pkm3: an optimal markov model for predicting future navigation sequences of the web surfers. *Pattern Anal. Appl.* 24(1), 263–281 (2021). <https://doi.org/10.1007/s10044-020-00892-7> <https://link.springer.com/10.1007/s10044-020-00892-7>
- de Sousa, A.L.R., de Souza, C.R.B., Reis, R.Q.: A 20-year mapping of Bayesian belief networks in software project management. *IET Softw.* 16(1), 14–28 (2022). <https://ietresearch.onlinelibrary.wiley.com/doi/abs/10.1049/sfw2.12043>
- Hassan, M.T., Junejo, K.N., Karim, A.: Learning and predicting key web navigation patterns using Bayesian models. In: Gervasi, O., et al. (eds.) *Computational Science and its Applications – ICCSA 2009*, pp. 877–887. Springer Berlin Heidelberg, Berlin (2009)
- Pearl, J.: *Probabilistic Reasoning in Intelligent Systems*. Morgan-Kaufmann (San Mateo) (1988)
- Wasserstein, R.L., Lazar, N.A.: The asa statement on p-values: context, process, and purpose. *Am. Statistician* 70(2), 129–133 (2016). <https://doi.org/10.1080/00031305.2016.1154108>
- Dejaeger, K., Verbraken, T., Baesens, B.: Toward comprehensible software fault prediction models using bayesian network classifiers. *IEEE Trans. Software Eng.* 39(2), 237–257 (2013). <https://doi.org/10.1109/tse.2012.20>
- Okutan, A., Yildiz, O.T.: Software defect prediction using Bayesian networks. *Empir. Software Eng.* 19(1), 154–181 (2014). <https://doi.org/10.1007/s10664-012-9218-8>
- Kalinowski, M., Mendes, E., Travassos, G.H.: An industry ready defect causal analysis approach exploring Bayesian networks. In: International Conference on Software Quality, pp. 12–33. Springer (2014)
- Furia, C.A., Torkar, R., Feldt, R.: Applying Bayesian analysis guidelines to empirical software engineering data: the case of programming languages and code quality. *ACM Trans. Software Eng. Methodol.* 31(3), 1–38 (2022). <https://doi.org/recursos.biblioteca.upc.edu/10.1145/3490953>
- Manzano, M., et al.: A method to estimate software strategic indicators in software development: an industrial application. *Inf. Software Technol.* 129, 106433 (2021). <https://doi.org/10.1016/j.infsof.2020.106433>
- del Águila, I.M., del Sagrado, J.: Bayesian networks for enhancement of requirements engineering: a literature review. *Requir. Eng.* 21(4), 461–480 (2016). <https://doi.org/10.1007/s00766-015-0225-3>

39. del Sagrado, J., del Águila, I.M.: Stability prediction of the software requirements specification. *Software Qual. J.* 26(2), 585–605 (2018). <https://doi.org/10.1007/s11219-017-9362-x>
40. Wiesweg, F., Vogelsang, A., Mendez, D.: Data-driven risk management for requirements engineering: an automated approach based on Bayesian networks. In: 2020 IEEE 28th International Requirements Engineering Conference (RE), pp. 125–135. IEEE (2020)
41. Mendes, E., et al.: Using bayesian network to estimate the value of decisions within the context of value-based software engineering: a multiple case study. *Int. J. Software Eng. Knowl. Eng.* 29(11n12), 1629–1671 (2019). <https://doi.org/10.1142/s0218194019400151>
42. Britto, R., Souza, L.: Bayesian analysis of bug-fixing time using report data. In: ESEM '22: Empirical Software Engineering International Week (2022)
43. Basili, V., Caldiera, G., Rombach, H.D.: The Goal Question Metric Approach (1994)
44. ISO/IEC 25010:2011: Systems and Software Engineering — Systems and Software Quality Requirements and Evaluation (SQuaRE) — System and Software Quality Models. International Organization for Standardization, Geneva (2011)
45. Staron, M.: Action Research in Software Engineering. Springer (2020)
46. Ralph, P., et al.: Empirical Standards for Software Engineering Research (2021)
47. Braun, S., et al.: Tackling consistency-related design challenges of distributed data-intensive systems: an action research study. In: Proceedings of the 15th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM), pp. 1–11 (2021)
48. Choraś, M., et al.: Measuring and improving agile processes in a small-size software development company. *IEEE Access* 8, 78452–78466 (2020). <https://doi.org/10.1109/access.2020.2990117>
49. Wohlin, C.: Case study research in software engineering—it is a case, and it is a study, but is it a case study? *Inf. Software Technol.* 133, 106514 (2021). <https://doi.org/10.1016/j.infsof.2021.106514>
50. Martínez Fernández, S.: Chessleague.es, juego en línea de simulación de ligas de ajedrez' (2022). <https://www.chessleague.es>
51. Martínez Fernández, S.: Chessleague [Manuscrito]: juego en línea de simulación de ligas de ajedrez (2010). https://indaga.ual.es/permalink/34CBUA_UAL/1f96lk/alma991001193459704991
52. del Rey, S.: Desarrollo dirigido por datos: Un ejemplo práctico con ChessLeague. [B.S. thesis]. Universitat Politècnica de Catalunya (2021)
53. Daschner, S.: Architecting Modern Java EE Applications: Designing Lightweight, Business-Oriented Enterprise Applications in the Age of Cloud, Containers, and Java EE 8. Packt Publishing Ltd (2017)
54. DosSantos, P.S.M., Travassos, G.H.: Action research can swing the balance in experimental software engineering. In: *Advances in Computers*, pp. 205–276. Elsevier (2011)
55. Chapman, P., et al.: CRISP-DM 1.0: Step-by-step Data Mining Guide, vol. 9, pp. 13. SPSS inc (2000)
56. Ebert, C., et al.: Data science: technologies for better software. *IEEE software* 36(6), 66–72 (2019). <https://doi.org/10.1109/ms.2019.2933681>
57. Deemer, P., et al.: The Scrum Primer (2012). https://www.infoq.com/minibooks/Scrum_{_}Primer/
58. Elasticsearch: Logstash (2022). <https://www.elastic.co/logstash/>
59. Neapolitan, R.E.: Learning Bayesian Networks. Prentice Hall (2003)
60. Friedman, N., Geiger, D., Goldszmidt, M.: Bayesian network classifiers. *Mach. Learn.* 29(2/3), 131–163 (1997). <https://doi.org/10.1023/a:1007465528199>
61. R Core Team: R: A Language and Environment for Statistical Computing. Vienna (2021). <https://www.R-project.org/>
62. Wickham, H., et al.: dplyr: A Grammar of Data Manipulation (2022). <https://dplyr.tidyverse.org>
63. Scutari, M.: Learning Bayesian networks with the bnlearn R package. *J. Stat. Software* 35(3), 1–22 (2010). <https://doi.org/10.18637/jss.v035.i03>
64. Quesada, D.: dbnr: Dynamic Bayesian Network Learning and Inference (2022). r package version 0.7.5. <https://CRAN.R-project.org/package=dbnR>
65. Hansen, K.D., et al.: Rgraphviz: Provides Plotting Capabilities for R Graph Objects (2022). <https://bioconductor.org/packages/release/bioc/html/Rgraphviz.html>
66. Hugin Expert (2022). <https://www.hugin.com/>
67. DrivenData: Cookiecutter Data Science (2022). <https://drivendata.github.io/cookiecutter-data-science/>
68. Breiman, L., et al.: Classification and Regression Trees. Chapman & Hall/CRC (1984)
69. Therneau, T., Atkinson, B.: Rpart: Recursive Partitioning and Regression Trees (2019). r package version 4.1-15. <https://CRAN.R-project.org/package=rpart>
70. Nagarajan, R., Scutari, M., Lèbre, S.: Bayesian Networks in R: With Applications in Systems Biology. Use R!. Springer-Verlag (2013)

How to cite this article: del Rey, S., Martínez-Fernández, S., Salmerón, A.: Bayesian Network analysis of software logs for data-driven software maintenance. *IET Soft.* 1–19 (2023). <https://doi.org/10.1049/sfw2.12121>