

Development of a method for software maintenance using event logs

R Murdoch

 **orcid.org 0000-0003-1673-2081**

Dissertation accepted in fulfilment of the requirements for the degree *Master of Engineering in Computer and Electronic Engineering* at the North West University

Supervisor: Dr J Marais

Graduation: May 2020

Student number: 24161624

ABSTRACT

Previous studies have shown that software maintenance can comprise between 60% and 80% of a software system's overall cost due to the time spent on maintenance. It is also important to note that in any software system, it is vital to make software maintenance a priority to encourage efficient use of the resources available to the software system. A need, therefore, exists to reduce the time spent on software maintenance.

From the literature it could be seen that event logs are vital for software maintenance in systems, but a gap exists regarding which method should be followed on how to log events within the systems. There are currently no methods showing or explaining how to log events and how these event logs should be structured for implementation.

Therefore, this dissertation will develop a generic method for software maintenance which uses structured event logs as a base which can be implemented on any software system. This method will aim to reduce the time spent on software maintenance, which can correlate to reducing the overall cost of a software system's life cycle. The current literature has shown that event logs can also contain valuable information regarding a system's overall performance.

This method will ensure that a software system has enough information in its event log structure to be able to run analyses on the event logs for interpretation. Another advantage to the design of this method is that it will enable effective filtering and make the event logs available to software maintainers. It will also enable the event logs to be easily accessible and stored in one central location. Lastly, this method will ensure that every event log is effectively structured in such a way that it can be implemented and interpreted on any software system.

Using this method, a hypothesis is drawn that an event log that contains all the information – as designed from literature – can be used to recognise patterns and event signatures over time and compare them before and after the method has been implemented. Using this method, the time it takes to identify error and critical error hotspots in event logs can also be reduced. Where these hotspots can be identified, a quicker reaction time to them can be expected, which in turn reduces the time spent on software maintenance.

There are three distinct systems which were identified on which the designed method for software maintenance were implemented. These systems are all unique with different functionalities which include a system which translates data, a system which extracts data, and a system which sends out reports of the processed data.

These systems were used as case studies to prove that the designed method for software maintenance can be used and implemented on any software system to reduce the time spent on software maintenance. The time spent filtering through and finding unstructured event logs compared to the structured event logs in the method after implementation in these case studies was reduced by almost half the time.

Keywords: Event logs; Event signatures; Method for software maintenance; Reduce time spent on software maintenance; Software maintenance; Structured event logs

TABLE OF CONTENTS

ABSTRACT	I
TABLE OF CONTENTS.....	III
LIST OF TABLES	IV
LIST OF FIGURES.....	V
LIST OF ABBREVIATIONS	7
ACKNOWLEDGEMENTS	8
CHAPTER 1 INTRODUCTION.....	9
1.1 Preamble	9
1.2 Background	9
1.3 Literature study	11
1.4 Need for the study	26
1.5 Study objectives.....	28
1.6 Summary	28
CHAPTER 2 DEVELOPMENT OF A METHOD FOR SOFTWARE MAINTENANCE USING EVENT LOGS.....	29
2.1 Preamble	29
2.2 Analysing the requirements.....	29
2.3 Functional design of the method	30
2.4 Design of the event log structure	35
2.5 Summary	46
CHAPTER 3 IMPLEMENTATION AND RESULTS	48
3.1 Preamble	48
3.2 Implementing the method for software maintenance using event logs	48
3.3 Verification through case studies	54
3.4 Validation.....	69
3.5 Summary	75
CHAPTER 4 CONCLUSION	76
4.1 Summary	76
4.2 Recommendations for further study	78
BIBLIOGRAPHY	80

LIST OF TABLES

Table 1: Derived event log structure component breakdown	37
Table 2: Preliminary event log components	38
Table 3: Preliminary detail event log components.....	38
Table 4: Preliminary exception event log	39
Table 5: Final header event log components	42
Table 6: Final detail event log components.....	43
Table 7: Final exception event log components	44
Table 8: Online survey - Structured event logs access timestamps	69

LIST OF FIGURES

Figure 1: The software development life cycle (SDLC) [10]	11
Figure 2: Software development life cycle costs according to the research [13], [16]–[19]	12
Figure 3: Maintenance categories derived from literature [14], [18]	14
Figure 4: ETL Process.....	20
Figure 5: ETL functionality and LookUp operation	21
Figure 6: Process mining types	23
Figure 7: High-level software maintenance method	31
Figure 8: Software system and event logs	32
Figure 9: Event log viewer	33
Figure 10: Event log aggregation.....	33
Figure 11: Event log maintenance	34
Figure 12: Derived event log structure	36
Figure 13: Preliminary event log structure	37
Figure 14: Preliminary event log implementation algorithm.....	40
Figure 15: Final event log structure	42
Figure 16: Final event log implementation algorithm.....	45
Figure 17: Example of unstructured event logs.....	49
Figure 18: Example of structured event logs.....	50
Figure 19: Code snippet of an event being logged.....	51
Figure 20: System logs interface – Detailed event information	52
Figure 21: System logs interface – Event escalation overview.....	52
Figure 22: System logs interface – Event log analysis	53
Figure 23: Data extraction system	55
Figure 24: Case study 1 – Data extraction maintenance task	56
Figure 25: Software maintenance on the data extraction system	57
Figure 26: Average time spent on maintaining the data extraction system	58
Figure 27: Data translation system	59
Figure 28: Case study 2 – Data translation maintenance task	60
Figure 29: Software maintenance on the data translation system.....	61
Figure 30: Average time spent on maintaining the data translation system.....	62
Figure 31: Data reporting system	63
Figure 32: Case study 3 – Data reporting maintenance task	64
Figure 33: Software maintenance on the data reporting system	65
Figure 34: Average time spent on maintaining the data reporting system.....	66
Figure 35: Software maintenance using unstructured vs. structured event logs.....	67
Figure 36: Average time using unstructured vs. structured event logs.....	68

Figure 37: Ease of use of unstructured vs. structured event logs.....	70
Figure 38: Event log structuring using old method	71
Figure 39: Event log structuring using new method	71
Figure 40: Accessibility of unstructured vs. structured event logs	72
Figure 41: Data translation system throughput analysis.....	73
Figure 42: Reporting system errors after a deployment	74

LIST OF ABBREVIATIONS

API	Application programming interface
BI	Business Intelligence
CI	Communicated Information
CSV	Comma Separated Values
DSS	Decision Support System
ELK	Elasticsearch, Logstash, Kibana
ETL	Extract, Transform, Load
ID	Identification
IEEE	Institute of Electrical and Electronics Engineers
ISO	International Organization for Standardization
ITS	Issue Tracking System
LPA	Log Processing Applications
MLE	Mandatory Log Events
NoSQL	Not-only Structured Query Language
PAIS	Process Aware Information System
PDF	Portable Document Format
REST	Representational State Transfer
SDLC	Software Development Life Cycle
SM	Software Maintainer
SQL	Structured Query Language

ACKNOWLEDGEMENTS

A heartfelt thank you to ETA Operations (Pty) Ltd for providing me with financial support during this study as well as time and resources.

Furthermore, I would like to thank the following people:

- My Heavenly Father, for giving me the privilege of completing this study and blessing me with opportunities to further my studies.
- My parents, for their love, sacrifices and motivation.
- My girlfriend, for supporting me during this study.
- Dr. J. du Plessis and Dr. J. Marais, for their insight, support and guidance.
- Prof. E.H. Mathews and Prof. M. Kleingeld, for providing me with the opportunity to continue my studies at CRCED Pretoria.

CHAPTER 1 INTRODUCTION

1.1 Preamble

This section provides an overview of the relevant literature needed for the study as well as background on why the research was done. The literature study will highlight the need for this study, and finally, the introduction chapter will conclude with the shortcomings identified in the literature and what can be taken from the literature into the design phase.

1.2 Background

Software maintenance is a process that is implemented after a new system has been deployed and is responsible for keeping software up to date and in working order. It usually consists of upgrades to the software, bug fixes, new features requested by a client or a user, and changes in the software environment. The reliability of the software as well as its performance and adaptability can be significantly improved by implementing software maintenance procedures [1].

However, software maintenance has been identified as the largest contributor to the software development life cycle (SDLC) and the time and costs related to this phase in the SDLC are often underestimated [2], [3]. When developing software there are requirements in place to reduce the costs of the overall development as it is always increasing. Because the maintenance phase is known for being the largest contributor to the costs of development, there is always a need to find ways to reduce these costs whilst improving the reliability and maintainability of the software [1].

Since the software must be maintained and its costs are so significant in the SDLC, it is necessary to have proper methods in place when approaching software maintenance. One of the specific tasks that system maintainers are responsible for is finding and fixing issues within an existing system. Generally the software maintainers would need some technical background on the system to know how to do fault finding and how to fix those issues [2].

Therefore, software system maintainers often need more information on issues that occur. This can be achieved by means of event logs to gain better insight when investigating these issues [2]. The event logs which can be used for software maintenance usually contain a large

amount of extracted information regarding the execution of business processes, also known as process mining [4], [5].

The amount of these event logs generated by a software system containing important information regarding the execution of processes, can be considered as Big Data [2], [6]. This is a term which is very broadly defined and has multiple definitions. It can however be summed up as large information sets which can only be processed by non-traditional data tools for analysis and storage. This means that to analyse the large data sets, complex computational platforms are necessary [7].

The event logs used by software maintainers which are considered as Big Data have to be stored in a database which is able to manage Big Data before approaching software maintenance. Therefore, it is necessary to make the right decisions when choosing a database to manage large volumes of event logs. The event logs stored in these Big Data databases are generally stored for aggregation purposes to gain valuable insights for Business Intelligence (BI) [8].

The event logs are usually stored by properly designed extract, transform, and load (ETL) procedures. The extract process is responsible for accessing the raw data, after which the transformation procedure manages the raw data and modifies it into structured, filtered and aggregated data. Lastly, the load procedure is responsible for loading the structured data into the database [9].

However, event logs are not always loaded into a database using a generic and structured method, which can lead to complex analysis procedures for extracting valuable information from them. Furthermore, research has shown that event logs can be of great significance when software has to be maintained [2], although this is not always the case as they are often only used for gaining insight into issues or errors within a system. Instead, event logs have the potential to provide considerable value to a business if they can be more generic and structured, which in turn provides more detailed information, not only when errors occur, but likewise during a system's runtime which can aid software maintainers.

1.3 Literature study

1.3.1 Existing software maintenance procedures

It is central to this study to note that software maintenance is a key component of the SDLC. As shown in Figure 1, the SDLC consists of different phases defined as the requirements, design, testing, implementation, and the maintenance phases [10], [11]. This dissertation will focus mainly on the software maintenance phase.

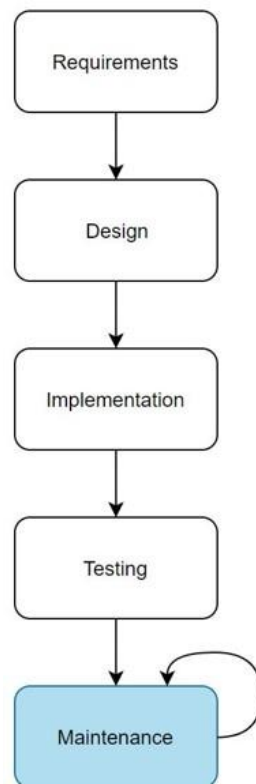


Figure 1: The software development life cycle (SDLC) [10]

Figure 2 shows the SDLC which was constructed from the research conducted in this study. From this chart it can be seen that software maintenance is a large contributor to the costs of the SDLC [12].

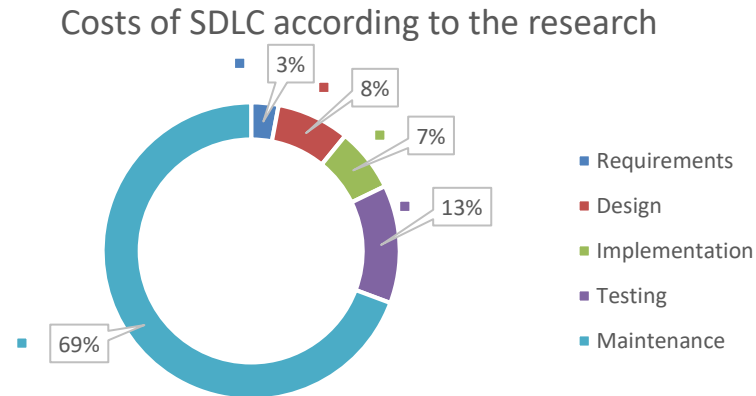


Figure 2: Software development life cycle costs according to the research [13], [16]–[19]

Since it is understood that software maintenance forms the largest part of the SDLC, it is important to understand the elements of software maintenance. According to the Institute of Electrical and Electronics Engineers (IEEE), software maintenance can be defined as the process of constantly changing a software system's components after the software has been delivered to a client or user of the software. This is done to improve on software performance as well as other aspects such as fault correction and adapting to constant environmental changes [13].

Marounek [13] has a different view on how software maintenance should be defined, which is in contrast to the definition provided by the IEEE. Marounek [13] rather defines software maintenance as all the activities that are necessary for a software system to be supported in a cost-effective manner. This means that even before a software system goes into operation, maintenance should be implemented on the system. Therefore, maintenance should typically start when the new development of a software system begins [13].

During the SDLC, the last phase of the life cycle – also known as the software maintenance phase – is identified as a critical phase of the life cycle. Environments that are continuously changing around software emphasise the fact that there is a real necessity to make this phase a priority [2], [3], [14], [15] .

The maintenance phase is recognised as the most costly component of any project in software development, thereby rendering this phase the main contributor to time, effort and costs involved in a project [2], [3], [14]. Research indicates [13], [15]–[19] that this phase can contribute between 60% and 80% of the total project cost due to the time spent on maintenance during a software development project.

Software is developed over time and this development phase can require up to two years of the total software development process. Depending on the type of application, the maintenance phase can, however, require much more time, consuming up to ten years of the total software development process, which is usually when software is either discontinued or replaced with new and updated software [10].

There are key components of software maintenance that should be highlighted. Firstly, control should be maintained on the system's daily running functionality. Secondly, control should be maintained over any changes that are made over the system. All existing functions that are acceptable should be perfected. Lastly, the performance of the system should be closely monitored to prevent degradation [14].

In 2004 proper documentation for maintenance processes within an organisation were introduced and implemented, ensuring the system's maintainers were knowledgeably aware of all the elements involved. This is still applicable today [14].

Software which is efficient, effective, reliable, and that can be maintained is often difficult to develop; therefore, it is important to focus on software maintenance as there will always be a need for improvement in this area [20]. Existing software which develops over time and adapts to constant change usually increases in complexity, and issues which are directly related to maintenance on the software also tend to increase. The system's state can, however, be improved by taking preventative measures [19].

April et.al. [14] have considered the ISO14764 international standard for software maintenance and identified two categories, namely, preventative and corrective maintenance on software. The former is the identification of errors before a system becomes prone to total failure, and the latter reacts on existing errors which impacts a production system's operation by eliminating the error conditions. Sometimes errors go unnoticed for a period of time and, therefore, corrective maintenance should be applied [11], [14], [21].

According to Shray et. al. [18], software maintenance can be split up into different parts, as stated before by [14]. Shray et. al. [18] adds that adaptive and perfective maintenance can also be conducted depending on the nature of the software delivered.

Adaptive maintenance on software ensures that the software is kept up to date and that it is modified, enabling it to run as trends occur over time. Perfective maintenance ensures that software can operate over an extended period. This is done by applying updates and new

features as well as doing modifications to the software, ensuring that the software product is reliable and dependable over time [11], [18].

Figure 3 is derived from literature [14], [18], and shows the different categories into which software maintenance can be separated.

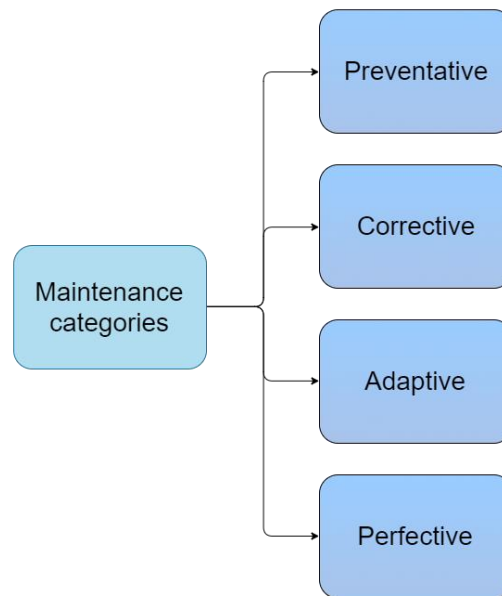


Figure 3: Maintenance categories derived from literature [14], [18]

Another method of maintaining software is explained by [22], where it was found that the amount of repairs on software declined and thus needed less frequent attention. This was due to more modern methodologies that were used during development. Changes that happened over time on maintaining the software using these methodologies resulted in software that increased in reliability [22].

Usually when a system has high complexity, it introduces increased maintenance costs. This can, however, be counterweighted by a system which, in turn, provides more functionality since it is maintained. In the initial deployment stage of new software and its maintenance, the time spent on sustaining the system usually increases in relation to complexity. From the study conducted by [22] it is clear that maintenance plays a significant role during a software system's lifetime.

With the knowledge that software maintenance is critical, time consuming and costly, a need arises for innovative ways it can be approached [23]. One such way is the use of event logs, as they can contain important and useful information regarding the software application or system [12]. Event logs can be used for system diagnostics and issue detection. However, these logs can become extensive complex structures, or have no structure at all [23]–[25].

Therefore, it is necessary to note how event logs have previously been implemented for software maintenance.

1.3.2 Event logs used for software maintenance

In any software system it is important to understand how the system operates. To achieve this the system needs to be studied and observed in a production environment during runtime [26]. In present-day systems there are various sources that generate data. Significant information can be sourced regarding a system and its processing activities by specifically looking at event logs that are generated by software systems. The event logs can, therefore, be used for approaching and implementing software maintenance procedures [24], [27], [28].

A process followed by [2], when software systems need to be maintained and issues in the system persist, is aimed at developers or maintenance personnel who should consider event logs on the issue. These event logs typically contain information from the system such as which action was performed when the issue occurred, the criticality of the issue, and how it affects the system's overall functionality [2], [23], [29]. Using this information stored in an event log, an investigation can be done on the issue that occurred by the personnel responsible for maintaining the system. After the issue is resolved, the event log can be removed [2].

Different systems exist to manage event logs, but have scalability issues, can be costly and require technical skills to be managed. An ecosystem namely the ELK (Elasticsearch, Logstash, Kibana) stack can be specifically used for storing, managing and analysing event logs and aims to address some of the disadvantages of using such systems [30]. Elasticsearch is used to store and index the information gathered and structured by Logstash, which is an event log collection pipeline. Kibana then provides visualisations and analyses on the stored data structures [30]. The ELK stack is typically hosted on a cloud computing platform or physical hardware both requiring self-deployment and management. Alternatively, paid services are available for hosting the ELK stack [31].

Event logs are generally utilised as an input for processing complex events, optimising systems, and for process mining techniques that rely on it [6], [26], [29]. The costs regarding the challenges faced by the software maintenance control are far more than simply the cost reduction by error detection applied at the beginning of the software maintenance life cycle or by evading errors [19].

The information contained in an event log can also be referred to as Communicated Information (CI), since it is known as system information that has been automatically communicated during a system's execution. Another type of log is called a trace log, but this is typically generated manually by analysts during the execution of a system [32].

Event logs may also be used for predicting future failure in a software system during early stages and can ensure that the software system is robust, which will subsequently minimise the costs spent on software maintenance in terms of time and cost [33].

Runtime analysis and the prediction of failures could possibly extend a system's functionality to react on potential failures automatically and apply corrective actions before these failures occur [33]. The study conducted by [33] focusses on error log records generated by a system to craft error-spread signatures during the runtime of the system, and then summarises the error counts. However, although the method that is used by [33] is excellent for failure prediction, it is limited to explaining what will be done with the event logs and not how the event logs should be structured or how they should be logged. Emphasis is also only put on errors for failure prediction and not on any other system features.

The event log information of a software system can hold large amounts of significant data around the runtime execution of a system and its error states [34]. These event logs that have been generated and which can be used to predict future failures can assist a software maintainer who has little or no background in the understanding of a system or how it was designed. Additionally, there is no need for special software instrumentation to achieve this [33].

Error log variables are used by [33] to generate an error distribution together with error parameters that are available during runtime signatures. The variables in the error logs typically contain either the type of error together with a message, or only parameters within the software system's logs [25], [33].

A study done by [35] looked at a Process Aware Information System (PAIS) where event log mining was performed from the logs that were generated by this system. This was used to determine a process model as well as system enhancement and maintenance. Juneja et. al. [35] explains that a minimum of four fields are necessary to create an event log for an event that occurred during an algorithm called "process model discovery."

These fields should include a case ID, timestamp, activity, and an actor. The fields that are included in the event log are structured according to the case ID and in an increasing order for the timestamp. Juneja et. al. [35] applies sequential data clustering and as a result the data is transformed into a sequential format.

This could mean that for parallel processes in a software system, the sequential data clustering may not be the ideal way of transforming the data. The analysis of unstructured event logs that have been mined from real-world applications might be difficult due to the complexity of the process models that are produced and the fact that these models are spaghetti-like. To aid process analysts in understanding these models, [35] contributed by trying to simplify the complex process models through clustering.

The structure that was proposed by [35] could be used for a more generic structure, which can be implemented on various software systems. Not much detail has been given around a method or a structure for event logs that might aid software maintenance, and the focus of the event logs were mostly on how they will be used after they have been mined.

A software system that was implemented by [36] has event logs that included the date each log was created, where it was created, the machine on which it was created, and who caused the event, since it was an application that had human interaction. Lastly, the event log had a type, state and description which provides more information regarding it.

Categorising event logs that are unstructured pose a number of challenges which have undergone research over the years according to [25], [37], because problems can be predicted and determined. Therefore, a specific study on a cloud-based system has been done. This system had event logs that comprised four parts. The first part had the event log status in terms of its severity. The second part had the time that the event happened. The third part consisted of the location where the event took place [37]. Lastly, a description is added to the event log by software maintainers.

Lastly, some more detail around the event log is added for use by software engineers. From this study it has been shown that event logs should be designed to have a constant structure and most of the event logs that do currently exist in systems are mostly unstructured [37].

There are two event log types that have been introduced by [26] that were used during software maintenance. The first type of event log is called a “flat” event log. Here the event log typically explains what event occurred when inside of the software, and each event is part

of a set which is grouped to form traces. This is the entry point for process mining techniques [26].

Every process comprises traces which correspond to the process during its execution, and there are different kinds of features that belong to an event log, of which it is characterised by. Examples of the features that form part of a “flat” event log could be a timestamp, which corresponds to a point in the process; a representative start and end event, which points to the entry and exit point of a process; a reference to where in the process the event occurred, etc. [26].

A “flat” log can then be extended to have many different activities linked to it, which makes it a hierarchical event log – this is the second type of event log. The hierarchical event log describes each “flat” event log’s activity in more detail at every level according to its granularity [26].

To have a better understanding of what communicated information is, [32] explains that it usually consists of main activities inside of a system, for example events that occurred within a system. Along with such an event there is also some context, for example the time stamp when the event took place. The communicated information is then used by the software system administrators or developers to help understand what the overall behaviour of the system is, as well as fault-finding, system diagnostics, and bug fixing.

Zanoni et. al. [32] continues to distinguish between tracing information and communicated information. The former mainly occurs at a lower level to generate information that is not as descriptive as the latter, which is specifically put in place by software developers who aim to gather valuable information from a system to use in practice [32].

As communicated information is specifically designed by software developers and software engineers, the abundance of information contained in the event logs have given birth to event log processing applications (LPAs). These applications can be used to aid large software systems with maintenance and further development [32]. LPA’s are known for being developed in-house and use the communicated information available from processing the event logs; therefore, the communicated information shouldn’t be modified since the LPA’s are heavily reliant on it [32].

Processing event logs from an execution- and code-level uncovers communicated information which can be used for generating BI. In every system that implements event logs,

communicated information that has been gathered for each system should come from an event logging mechanism that is constant over the different releases of a system. This will aid software maintainers to keep records of trends in the software system over time [32].

The event logs that are currently used for uncovering communicated information usually do not have a set format, but rather have an inconsistent format. The use of execution event logs are preferred above an API that monitors a system since these logs contain valuable information that is more simplistic and light-weight for monitoring a system [32].

While event logs are a vital part of software maintenance, it is sometimes evident that the event logs lack quality [28], [38]. Event logs that are missing are also cause a problem. This can be due to systems that make use of manual recordings that may have been omitted or systems that have failed critically, also omitting event logs [6].

For characterisation of the patterns and features of erroneous states in software systems, [37] also proposes that event logs can be used to obtain critical information. This can then be used to connect the low-level events to broader, high-level error events. Error event logs can be analysed to allow characterisation of the impact that the error has had on the system, and patterns can be identified [37].

It is important to note that systems that operate on large scale datasets could have a logging mechanism which generates large amounts of error event logs that could be abstruse and redundant. Event logs that are redundant and abstruse could possibly obscure pattern identification of errors in a software system, making software maintenance problematic. This could also cause the analysis process to be obsolete since it could lead to incorrect error detection [37].

The study done by [5] extracted knowledge around processes in a business with event logs as an input for a method called frequent itemset mining. Djenouri et. al. [5] also proposed that three main steps could be used to extract business process knowledge. The first step states that analysis can be performed on data when event logs are prepared for it. Secondly, attribute-based strategy databases are created as well as events-based databases that contain the characteristics of the event logs for it to be analysed, and different insights around the data can be found. Lastly, frequent item sets are extracted as a third step.

From the literature, clear shortcomings were identified with regards to event logs and their structure for software maintenance. The focus on how event logs should be structured was

irrelevant in most studies. Where there was some information regarding event log structures, they were not properly explained. There was also a lack of knowledge around where event logs should preferably be stored for accessing at a later stage. A large number of the studies mainly focussed on how event logs should be used after they are already available, whether for manual analysis by software maintainers or automatic aggregation processes for gaining knowledge into a software system's execution. Therefore, there is a need to develop a generic method for doing software maintenance using structured event logs as the foundation.

1.3.3 Extract, transform and load procedures

Extract, transform and load procedures (ETL) are put into categories that consist of a synchronised set that comprises different tasks. For analysis purposes, the extraction (E) procedure accesses one or many different databases to fetch the raw requested data [9][39]. The transforming (T) procedure consists of standardising the data by following a large amount of filtering and aggregation tasks. The last procedure is the load (L) procedure which takes the transformed data and stores it in the database or data warehouse [9], [39], [40].

Figure 4 shows the ETL process and the interaction between these processes as previously explained. Bala et. al. [9] presented an approach that focusses on improving the performance of these ETL processes when Big Data is being handled. The context of the work that [9] has done on Big Data systems was decision support systems (DSS).

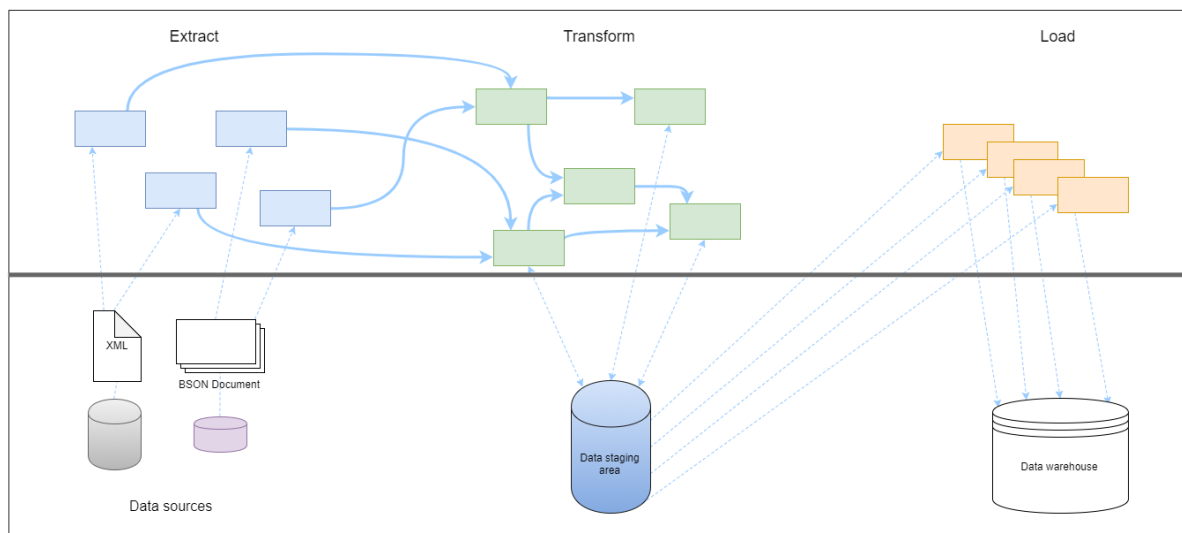


Figure 4: ETL Process

This type of system has a data warehouse in a central repository which is used to store valuable operational data or information for business activities [9]. For important information to be delivered to decisionmakers, the data warehouse is used by the DSS.

The ETL process is used and integrated to make the information available in the data warehouse that is used by the DSS. Therefore, the raw data is extracted, transformed by standardising the data and finally loaded into the data warehouse for use by the DSS for analysis, which can be interpreted by decisionmakers [9].

In order to increase the speed of the ETL process, parallelisation and distribution methods were used. The first method results in faster processing speeds by allowing several independent processing units to run simultaneously. The latter method uses a cluster that has multiple nodes and allows data partitions to be assigned to each of these nodes [9].

As seen in Figure 5, the functionality of the ETL process is illustrated by means of the lookup operation. The source data is represented by the origin of the data, the destination is represented by the target into which the data will be transferred, and the mapping is done between the source and the destination rows with the master lookup table. Most ETL processes are designed with the implementation and organisation of ETL functionalities such as LookUp [9].

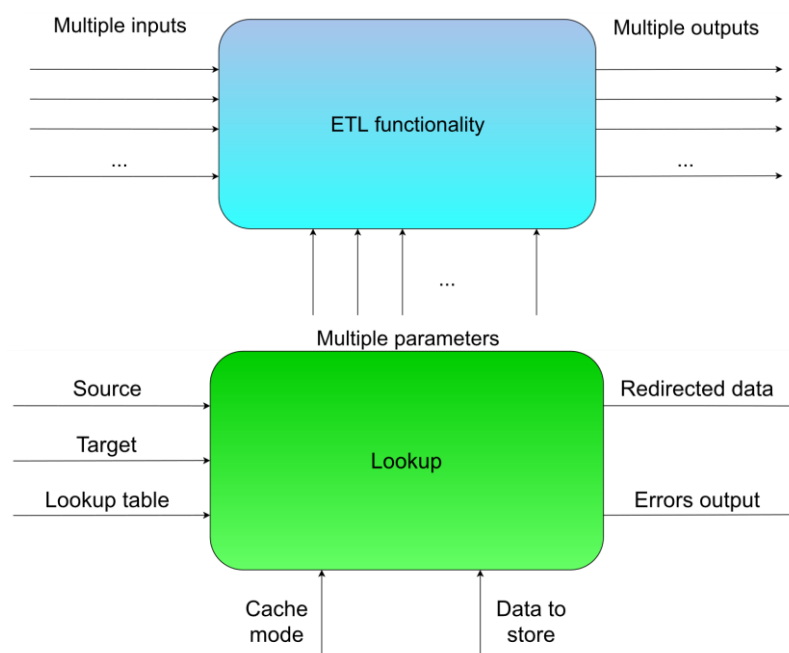


Figure 5: ETL functionality and LookUp operation

Since software systems need to be maintained, ETL processes are applicable regarding the way Big Data is handled, specifically regarding event logs. It is important to understand how a system can mine event logs and how the Big Data generated by the system can aid software maintenance.

1.3.4 Big Data and event log mining

This section will focus on event logs and the different ways in which these event logs can be stored as set out in the literature. It will also outline some information on what event logs are in the context of Big Data and how a large collection of event logs should be handled since most modern software systems have some form of trace logs.

To better understand event logs, it is necessary to understand process mining. Event logs represent information that is gathered in a business process or system and this method of doing is called process mining [41], [42]. When a process is implemented, it is characterised by event logs that are logged by an information system automatically [43].

It is required to have an understanding around information regarding the process. This is where the process mining discipline can be used to analyse the actual processes, since the event logs contain information regarding everything that happened in a system or business process [41], [44].

An example of process mining is an algorithm to handle any kind of noise by implementing a heuristic miner, which also puts focus on frequency quantification. Depending on the amount of traces, event logs can either be simple or very complex. This means that event logs can be structured in a simple way and still contain valuable information from traces [41].

A process model is built by analysing different activities' dependency values corresponding to the event logs. The heuristic miner, mentioned previously, analyses these event logs to determine their performance [41].

According to [41] and shown in Figure 6, there are different process mining types available. These are called enhancement, discovery, and conformance checking. The type of process mining which is of interest for this study is the enhancement type. This type focusses on previous process models to create a new process model, or renew or improve an existing process model [41].

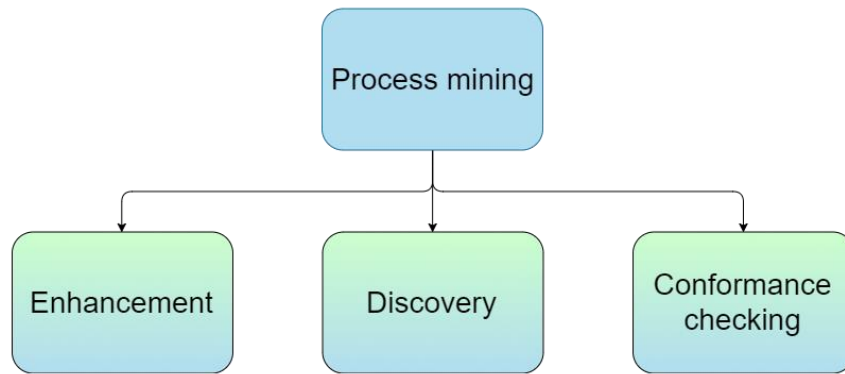


Figure 6: Process mining types

This is based on attributes which have been previously unidentified, which may include users, time, etc. [41]. This study applied event logs to three different case studies. From each case study it was clear that event logs need different event types or activity types. These event types helped categorise event logs for a better input for the study on the heuristics miner [41].

Event logs are typically generated for analysis at a later stage to extract meaningful information from them. Usually, event logs are produced in real-time and, therefore, according to [6], it can be seen as Big Data.

The analysis of different systems can be very process-intensive and may require a lot of computational power. According to [45], event logs are usually unstructured and are generated by various systems. Process mining uses these unstructured event logs to try and create models or definitions of real-life processes that are contained within them [45]. Since the event logs are unstructured, it makes it more complex to create these models.

The following study by [46] had the objective of performing forensic analysis through the identification of mandatory log events (MLEs), making use of a method that is heuristics-driven. Industry standard methods have also been researched and were compared to their heuristics-driven method to understand whether there are improvements for software engineers in being able to identify MLEs. This was conducted to aid security analysts [46].

Often, event log mechanisms which are responsible for recording events in a system as well as what is important to be logged, are inadequately and inconsistently implemented by software engineers. Therefore, the method that the study proposed is a method where MLEs are expressed as tuples of <verb, objects>, where the verb is defined as the user's actions that are performed and the object is the part on which the user has performed the action [46].

All of the industry standard methods researched by [46] only targeted certain aspects of event logs, namely error diagnostics and debugging. The method that this study proposes only seeks advancements in logging user activity. This, in turn, will provide eloquent forensic analyses of all the event logs recorded right after a breach in privacy or security has occurred [46].

Different techniques in process mining can be used to develop process models for a system which can be analysed and improved by extracting the system's event logs. There are multiple methods used to obtain event logs which can display a system's behaviour [26].

Event logs and their signatures were used in an approach by [33] to predict the possibilities for potential failures. All the event logs were stored in PostgreSQL, which is an open source and commonly used database system. For this approach to work, a single event that occurred could possibly not provide enough information to predict these failures, but multiple events stored in a database containing information about previous events may.

The event logs could possibly be of value to managers on various systems to identify underlying issues or alert that immediate action should be taken [36]. It is important that event logs should be available, especially historic event logs, which can be used at a later stage for making predictions based on historic information.

For efficiency to increase in the way business operations occur, business process analysis activities can be used. To achieve this, there are different methods in data mining that can be used that were specifically designed to extract patterns from event logs that could be of interest [5].

Many organisations have started to see process mining as a vital activity to analyse business processes automatically and to extract massive amounts of data which contain important information regarding business processes. To do this, information systems inside of organisations have valuable information logged about business processes during runtime and from this, different techniques have been applied to extract this information [5].

The information is transformed into event logs which are available in an applicable form for analysis. Firstly, an event log is created and then altered into an event log which is complete with more explanatory information around the initial event log [5].

Inside of modern data centres, log data are generated at an increasingly rapid pace, which creates technical challenges when these logs need to be managed. Usually, the interpretation

of the logs is done by humans and their verdicts on these logs are used. This causes an issue where the verdicts can be erroneous and inefficient [47].

Using a MySQL database, [35] endeavoured to achieve an implementation that uses a technique called trace clustering. This technique is grounded on applying groups of sequential data on an open source issue tracking system project, which has complex and large log-based datasets. As a first step, Issue Tracking System (ITS) data is extracted by using Bugzilla REST API for this Firefox project, and then the extracted data is saved to a MySQL database. The log-based dataset is, however, not too large as it can be handled in MySQL [35].

1.3.5 Database options

Choosing the right database can make a significant difference when large volumes of data need to be stored in the form of event logs. When data mining needs to be performed, data aggregation is a crucial feature which should be available to use in a database. Not-only-SQL (NoSQL) are databases which can handle large amounts of data [8].

NoSQL was introduced as the result of a need to deal with large repositories containing large datasets which needed to be managed and analysed. A pre-processing step is usually required when data mining must be done on a large volume of data. Complexity increases as the large amounts of data that must be manipulated are resource and time intensive [8].

Furthermore, the pre-processed aggregated data must also be stored. For meaningful information to be extracted in the future, [8] used MongoDB as the preferred database since it is reliable and fast for aggregating and retrieving data. It also has a built-in MapReduce framework which can handle large datasets of repositories as it can perform parallel computations [8], [48].

Another two advantages which NoSQL has over SQL is that it is scalable and vast amounts of data can be processed in an environment that is distributed [48]. Data aggregation in such databases are performed at a faster speed, and the read-write operations are optimised, although consistency is relinquished to achieve this [8].

This means that read and write errors are avoided. Even though consistency is not a priority, it should still be taken into account when designing software which will make use of these databases. Another strength that can be seen in NoSQL is that these databases are designed to have no schema, which is a model that simplifies updates to the database [8].

Different types of NoSQL databases exist and [8], [49] explains these as key-value stores, graph oriented-, column oriented- and lastly, document-oriented databases. In key-value stores, the data is represented in an unstructured or structured way and is particularly used for content that should be cached [8].

Data which are stored in a tabular manner are used in databases which are column-oriented and is usually used for frameworks which are write-intensive. Usually in social media networks, databases which are graph-oriented are used since graph-oriented data is stored. Lastly, databases which are organised to have collections that contain documents are document-oriented [8].

According to [8], MongoDB is a NoSQL database which has a lot of advantages when considering the amount of data it can handle. One of the main reasons to consider MongoDB is the fact that in the future, where BI and aggregation might be necessary, MongoDB provides a read-intensive architecture which is ready to handle both. There are many well-documented resources available for MongoDB since it has a helpful community and is open-source [8].

The database can also be directly queried since MapReduce can be used as a command. Most programming languages can integrate MongoDB's drivers which can also be used to build queries that are much more complex when compared to traditional SQL queries. This means that data can be retrieved with a better interface for queries and is also combined with the scalability which NoSQL is known for, which is very advantageous [8].

1.4 Need for the study

From the literature it can be seen that software maintenance is crucial. Therefore, event logs and process mining can be used to obtain communicated information which will support software maintenance.

It has been identified, however, that event logs lack quality and are mostly unstructured, unavailable or not meaningful enough. This means that the analytics which can be extracted at a later stage can be affected and no BI can be made available [6]. It is necessary for event logs to be structured in such a way that data analytics could later be implemented on the event logs that could lead to uncovering patterns and possible system failures. This will aid administrators responsible for software maintenance in circumventing any critical states as well as uncovering root causes when failures in the software systems occur [47].

Also, since event logs can be very dynamic, it is possible for developers to easily advance event logs to attend to issues and changes in a system more rapidly [32]. This means that there is a possibility for event logs to be structured and more generic. Therefore, the problem arises that there is also a need for a method on how software maintenance can be done by using event logs that are structured and generic.

The research has also shown that important information can be analysed by groups of LPA's to monitor software and improve software maintenance by being more time-efficient, but for this to occur it is important to have event logs available for gaining intelligence on a system or communicated information [32], [36]. To mine these event logs, they need to be in a structured form, otherwise they obscure the intelligence that can be extracted from them [36].

According to [50], event logs play such a vital role that it is sometimes the only method that can be used to determine and analyse the health and performance of a software system in production. It is essential for event logs to be accurate since the trust levels depend on it for analysing failures.

Inaccurate perceptions of a system's performance could be made if the event logs are unreported, or when there is no knowledge of a system, the maintenance administrators might make misinterpreted conclusions [50]. Most systems only use event logs for error logging and for detecting failures, but far more information could be extracted from event logs if they contain more detailed and relevant information while a software system is running in production.

Current event logs are also vulnerable to changes in a software system which can alter the integrity of them. This may include upgrades, configuration updates, and other changes which could be necessary during the lifetime of a system [36]. Therefore, the influence that these changes should have on event logs and their integrity and character should be minimised. This can possibly be done if event logs are structured towards being more generic.

It is, therefore, a problem that event logs are either unavailable or unstructured, which causes the necessity to have a method for software maintenance which integrates structured event logs. This will enable systems to support event log-based analysis, because currently, maintenance administrators need some background knowledge of the system to understand the unstructured event logs. It is not always the case that maintainers have a full background on the software systems and the analysis is a burden since it is a very manual process, because proper event log structures and methods are not in place [50].

1.5 Study objectives

The literature mentioned available systems which can be used to transform unstructured event logs into structured event logs. The advantages to using such systems include rapid expandability, low maintenance and cost effectiveness. However, there is also a possibility that such systems might have additional costs involved when it must be deployed on hosted cloud services or physical hardware for enterprise use. Therefore, the study aims to take it into consideration.

From the literature it could also be seen that there is a need for structured event logs that can be integrated into a method for software maintenance which is generic for implementation in software systems. This will aim to reduce the time spent on software maintenance by software maintainers being able to obtain access to the structured event logs that are always available and accessible in a central location. It will also aim to enable software maintainers to analyse the structured event logs to improve the overall software systems' performance.

This can be obtained by capturing software maintainers' experiences using a survey to compare the use of a legacy system which implemented unstructured event logs versus a system implementing the new method which uses structured event logs. The survey must track the time it takes to perform maintenance tasks and use the time as the measurable variable to verify whether there is a time reduction. The survey must also allow software maintainers to compare the legacy- and new system in terms of ease of use, structure and accessibility of the event logs.

1.6 Summary

From the literature in Section 1.3, shortcomings were identified. It also provided information that can be used as a basis of knowledge for the design of a method for software maintenance using event logs. These shortcomings led to a need for event logs to be structured for integration into a generic method for software maintenance.

The literature and the need for the study is used in the following chapter to identify the objectives regarding the method for software maintenance as well as the event logs. It will also discuss the design and development of the method for software maintenance which integrates structured event logs.

CHAPTER 2 DEVELOPMENT OF A METHOD FOR SOFTWARE MAINTENANCE USING EVENT LOGS

2.1 Preamble

This chapter aims to address the need for the study – to design a method using structured and more generic event logs to gain meaningful insight into a system during software maintenance.

Section 2.2 will describe how the requirements were obtained as well as the objectives of the study. This will be used to design the method as well as the event log structure. In Section 2.3, the functional design will be discussed. This will provide the functional phases which should adhere to the requirements discussed in Section 2.2. After the functional design, the event log structure will be designed which will be used in the method for software maintenance. Two designs will be discussed – a preliminary design and a final design, which will be used and implemented. This chapter concludes with a summary in Section 2.5.

2.2 Analysing the requirements

Analysing the requirements for the method for software maintenance which implements event logs begins by understanding that event logs will be used as the basis for the method used by software maintainers. From the literature and the need for the study, it was evident that software maintainers spend the most time on the maintenance phase. Therefore, to do software maintenance, the following requirements and objectives are identified:

1. The method for doing software maintenance **should reduce the time** spent on maintenance by rapidly identifying issues using event logs as the basis.
2. The event logs **should be available** at any given time.
3. The event logs **should be organised** in such a way that they can be retrieved easily.
4. The event logs **should be stored in a central place** for all the software maintainers to rapidly access the event logs.
5. The event logs **should provide enough information** regarding the execution of a software application or system for analysis purposes.

From the requirements and objectives set out, it is evident that there is a necessity to have event logs that are:

1. **Generic** – For use by software maintainers to implement in any software system or application. This means that the event logs should be able to conform to any new software system's requirements.
2. **Structured** – It should be designed in this way for simplifying the process of retrieving information from the event logs.
3. **Available** – After it has been structured, it should always be stored in a centralised location.
4. **Accessible** – Software maintainers should be able to have access to the abovementioned location of the event logs.

2.3 Functional design of the method

The literature highlighted that event logs can be used as the basis for doing software maintenance by ultimately using the information contained within to gain intelligence into the software system.

The literature states that event logs usually contain CI which is automatic information about the software system collected during the runtime [32]. This information has to be stored in a NoSQL database which will ensure that all the event logs are stored structured [2]. This will also ensure that event logs are available and easily accessible to software maintainers.

Having easy access to the event logs and having it available in a central location, i.e. NoSQL database, will ensure that the time spent on maintenance can be minimised. This will already be achieved by the fact that software maintainers can query the database and extract the necessary information since the event logs will be structured. However, in order to achieve this and before the event logs can be analysed, a method for software maintenance needs to be in place to understand where the event logs will fit into the overall picture of software maintenance.

As seen in Figure 7, *the high-level process of the method for software maintenance*, incorporating event logs within the software maintenance phase is shown. Firstly, a software system which must be maintained is shown at 1.0 in Figure 7. This software system can either be a system that has no event-logging mechanism or a system that does implement event logs, but that are unstructured.

Using the proposed method for software maintenance, the software maintainers will implement the new event log structure. The software system will generate event logs that are stored in a central database for software maintainers to use for maintaining a software system.

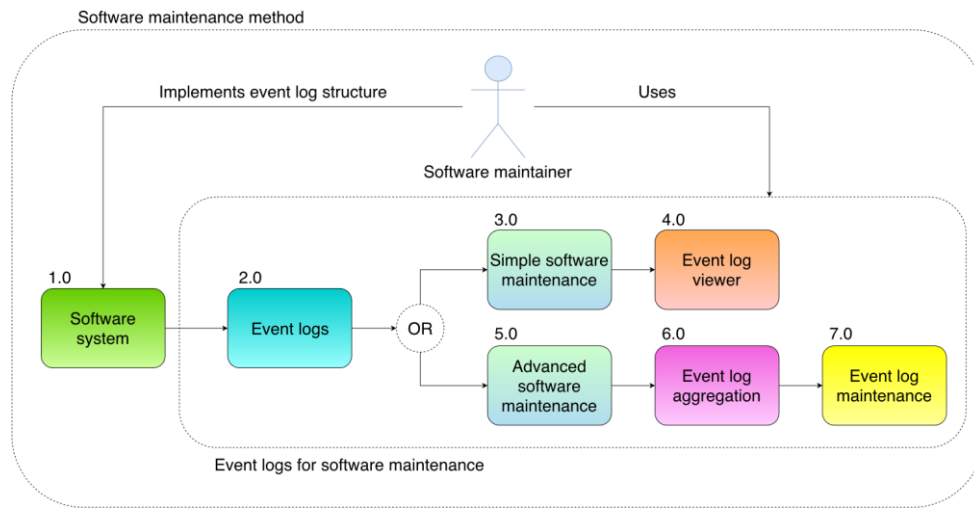


Figure 7: High-level software maintenance method

The event logs will be used by software maintainers for two types of maintenance – simple and more advanced software maintenance, as shown at 3.0 and 5.0 in Figure 7, respectively. The former by simply viewing the event logs that are stored in a central database, as shown at 4.0, and the latter for aggregation to eventually gain meaningful insight into the system's operations by performing aggregation on the event logs, as shown at 6.0. By doing aggregation, it allows for the event logs to also be maintained, as shown at 7.0.

Figure 8 expands upon Figure 7, showing an example of a typical software system from 1.0 in Figure 7, which has processes executing and transferring data to and from a database. The software system requests data from the database and the data is returned for processing by the software system. However, when the data is processed by the software system, there is no insight into the system regarding the execution of processes inside of the software system while it is running. For instance, if a process fails and event logs are not available, the fault-finding procedure for software maintainers to follow will be tedious since no event log traces of what occurred in the system are available.

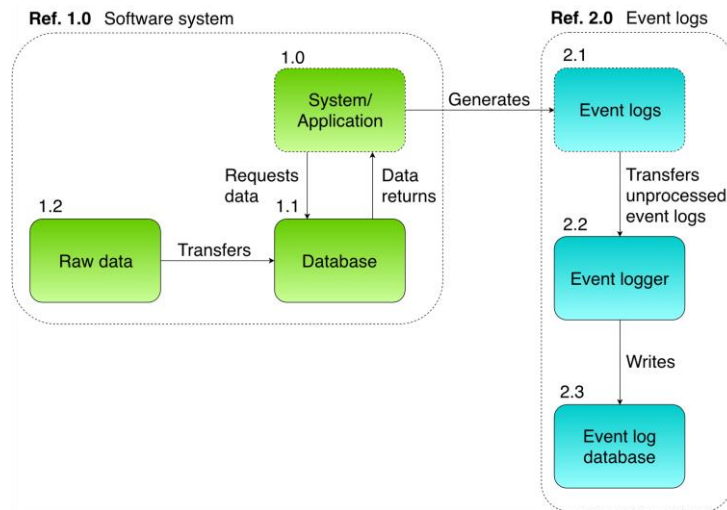


Figure 8: Software system and event logs

The new proposed generic method for doing software maintenance, as explained in Figure 7, will implement the new event log structure that starts at 2.1 in Figure 8. This shows how the event logs will be processed and how they will function within the software system. The software system which implements the event logs will generate new event logs which are still raw and unprocessed.

The unprocessed event logs are transferred to the event logger, which will be responsible for structuring the unprocessed raw event logs. This will structure the event logs in the correct format as specified by the event log structure design, which will be explained in Section 2.4. The event logger will be a middle component between the raw event logs and the event logs that are structured in the event log database, and will follow an ETL process.

The event logs are extracted from the software system generating the event logs which acts as the source of data. The transform procedure is where the event logs will be structured to be ready for loading into the data warehouse which will store the event logs in its structured form. When the event logs are structured and in the event log database, it will enable querying at a later stage, allowing software maintainers to access the event logs.

Simple software maintenance can be done by software maintainers who will be able to access the event logs by viewing them for troubleshooting. Figure 9 references part 4.0 of Figure 7 and shows the proposed event log viewer that will be able to use the structured event logs. Since the event logs will be structured in a database, an event log viewer can be created. The event log viewer can be used to scope into the system which is of interest and use it as a tool to reduce the time spent on maintenance.

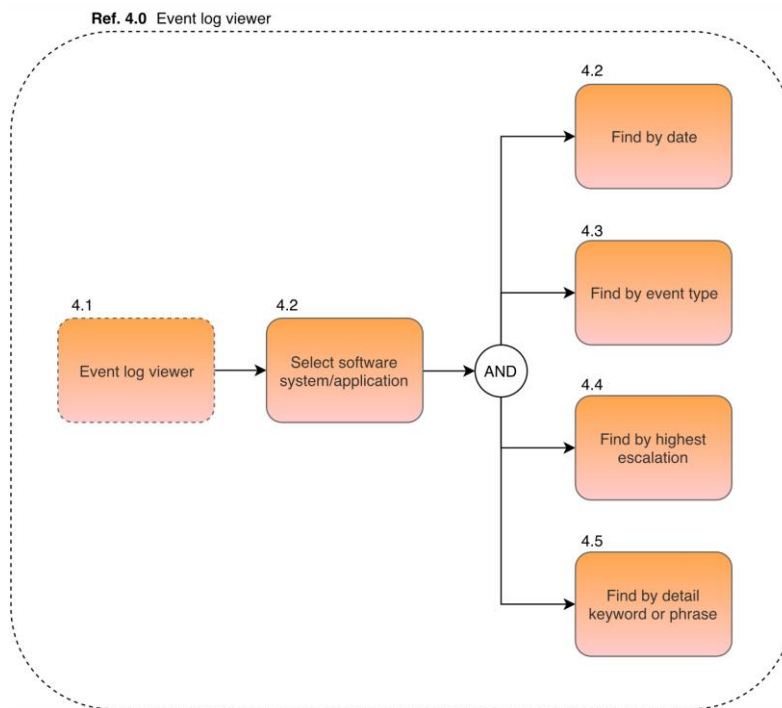


Figure 9: Event log viewer

As seen in Figure 9, the event log viewer will typically have filters according to which the event logs can be filtered. If a system is split up into multiple parts or smaller applications, the event logs should cater for linking to an application. The event log viewer will also need to filter the event logs by date, event type, highest escalation, and detail.

The filters will assist software maintainers gaining the fastest access to the event logs. Since a software system will generate large amounts of event logs each day, this viewer can create a way of viewing the logs without having to access a database to filter through the event logs.

Considering utilising more advanced software maintenance using aggregation methods as shown in Figure 10, more knowledge can be retrieved out of the event logs than what can be retrieved by the event log viewer. Not only will the advanced software maintenance be used when errors occur, but it will also provide other metrics which can be dependent on each specific software system.

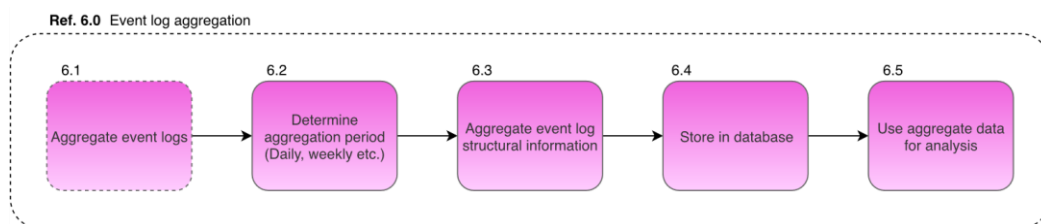


Figure 10: Event log aggregation

When doing aggregation on the event logs that are available, the aggregation period should be specified to determine the resolution at which the aggregate event log data will be made available. After the resolution has been determined, the aggregation can be processed on the event logs depending on the structural information contained within them. Since the event log database will contain a large amount of data, the aggregated data of the event logs should also be stored. This will enable analyses on the aggregate data which is available in the database.

Aggregating the event logs that are in the database allows detail event log data to be removed or backed up from the database. In effect, this will mean that the detailed information of the event logs is lost, but the most important information about the event logs is stored. This aims to reduce the amount of detail event logs to reduce the size of the database as well as remove any event logs that may no longer be relevant.

After all the event logs have been aggregated, the event logs must be maintained to avoid event log build-up in the database. The maintenance of the event logs must be an automatic process since software maintainers' time spent on overall software system maintenance needs to be reduced. Therefore, Figure 11 shows the process that will be followed to ensure that the event logs are maintained. This will also ensure that the relevant event logs are kept in the database and the storage of the event logs does not build-up.

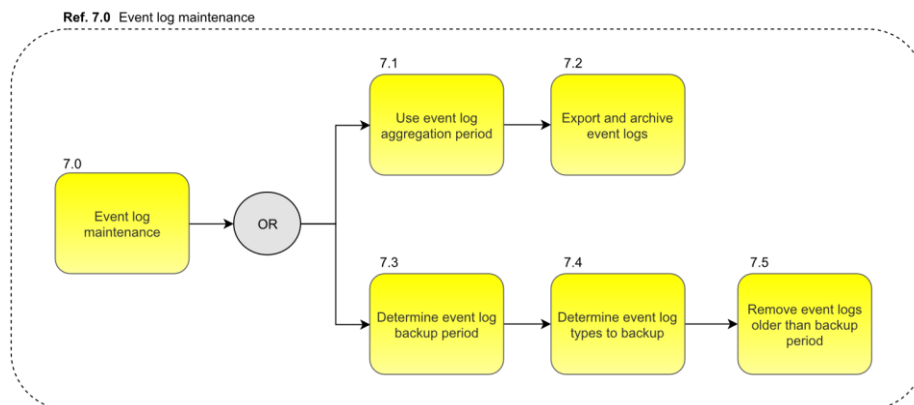


Figure 11: Event log maintenance

Since all the relevant aggregated information has been extracted from the event logs, the event logs have been used for this aggregation, as shown previously, which leads to residual event logs in the database. Therefore, the event logs that have been aggregated can be archived, as shown in 7.1 – 7.2.

This is only necessary if software maintainers find that there is a need to backup event logs for restoring at a later stage in time. Typically, it can be necessary to do backups if the storage space in the database is under pressure or if there is a need to evaluate event logs at an earlier stage in time. If this is necessary, the period of the aggregation will be used to archive the event logs. The backed-up event logs can, for instance, either be stored on the local storage that the software system is running on, or it can be stored using cloud-based storage.

Shown in Figure 11 at 7.3, the event logs will also be reserved in the database it is stored in for a certain period. Depending on the type of software system where the software maintenance method will be implemented, the period for which the event logs will be reserved may differ. This can depend on factors such as the average time it takes for system error identification or the relevance of the information contained within the event logs over time. When the period has been determined, the type of the event logs to reserve in the database will be determined. Finally, the event logs older than the backup period will be removed.

The event logs will be reserved in the database for viewing according to the most recent, as shown in Figure 9. The aggregation will ensure that trends on previous event logs are available. Therefore, it is not always necessary to reserve event logs for longer than a predetermined period. Historic event logs may also not be applicable anymore, since software maintainers have already solved an issue that no longer exists.

The functional design of the software maintenance method in this section focussed on how the implementation of event logs can be used to perform software maintenance on a software system. However, the event logs need to be in a certain structure to be used in the method. In the next section, the design of the event log structure will be discussed.

2.4 Design of the event log structure

The method for doing software maintenance using event logs requires that the event logs are structured for the purpose that useful knowledge can be extracted when the software maintenance method is implemented in a software system. Therefore, the focus of this section is to design the structure of the event logs. At first, the event log structure is derived from the literature and therefrom the preliminary event log structure design is created. The final event log structure, which expands 2.0 in Figure 7, will follow-up on the preliminary design.

The final design aims to simplify the preliminary design. Therefore, a preliminary and final design is considered. Another reason for considering these designs is to optimise the querying speed by minimising the complexity of the event log structure.

2.4.1 Preliminary design

Preliminary event log structure

As previously mentioned, there are different structures that have been used in software systems' event logs. Using this knowledge, a new event log structure can be designed. The purpose of designing the new event log structure is to ensure that the event logs can solve the problem as discussed before and to adhere to the requirements set out.

An event log typically contains a timestamp of when an issue occurred and how critical the issue was [2], [35]. Error event logs should also contain the type of error, together with a message or the parameters that are of interest [33]. The event log should be identifiable with an id as well as ordered by the id and timestamp [35]. Lastly, the event log should contain a form of location where the issue occurred [4], [36], [37]. Figure 12 shows a visual representation of the design of an event log structure that has been derived from previous studies [2], [4], [33], [35]–[37].

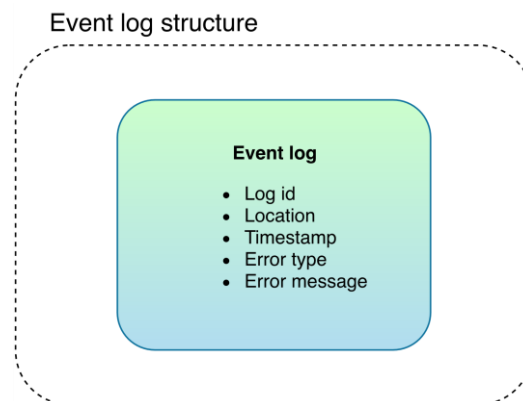


Figure 12: Derived event log structure

Table 1 explains the event log structure derived from the literature, as previously discussed. This shows what each event log consists out of and what each event log component means. From the event log structure design that has been derived from previous studies, it can be concluded that the focus is only on an event log which contains errors and details surrounding the error.

Table 1: Derived event log structure component breakdown

Event log component	Description
Log id	Unique id for each event log created
Location	Where in the process the software error occurred
Timestamp	The point in time when the error occurred
Error type	The error type determined by software maintainers
Error message	A message containing more information about the error event

As previous studies focussed mostly on event logs that contain errors for doing software maintenance, the event logs do not fit a generic method and do not contain additional information which could be useful if metrics other than software system issues need to be considered.

Therefore, a preliminary design of an event log structure is proposed in Figure 13 for a structured and generic event log. This contains error information as well as additional information that can be logged from a software system.

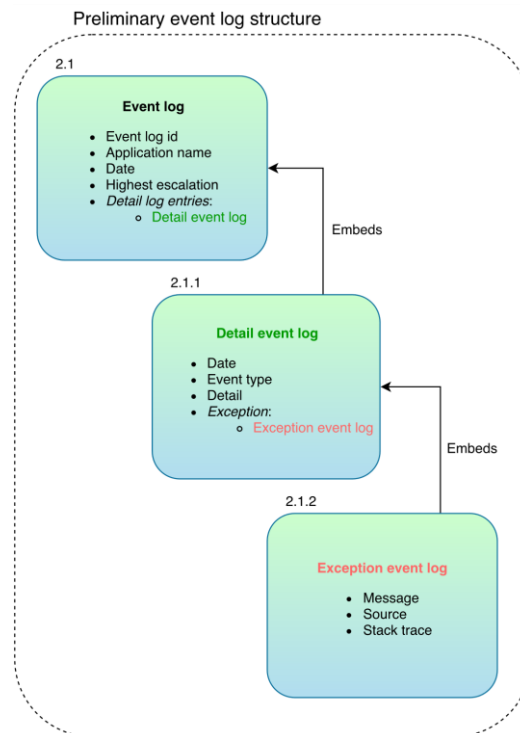


Figure 13: Preliminary event log structure

As seen in Figure 13, the proposed event log structure is a folded or a nested structure. As mentioned in previous studies, a main event log can be created and then altered into a complete event log which contains more explanatory information around the main event log [5]. Therefore, for the preliminary design, there is a main event log which has a detailed event

log embedded into it. Embedded into the detail event log is an exception event log. The main event log, shown at 2.1, has the following components in its structure, explained in Table 2.

Table 2: Preliminary event log components

Event log components	Description
Event log id	A unique event log id which is an identifier for the main event log
Application name	The location where the event took place
Date	A timestamp when the event log was created
Highest escalation	Escalation level calculated from the detail log entries
<i>Detail log entries</i>	Detail log

Table 3 explains the embedded detail event log entries contained within the event logs. Each event log may have multiple detail event log entries which contain more information surrounding the main event log. The optional exception event log will contain error information if something went critically wrong.

Table 3: Preliminary detail event log components

Detail event log components	Description
Date	Date when the detail log entry was created
Event type	Contains information about the escalation level of the event that occurred
Detail	Contains more detail around the event that occurred for the specific detail entry
<i>Exception</i>	Contains the exception message if the event type was an exception

Shown in Table 3, the date of the detail event log contains the creation timestamp corresponding to the date when the detail event log was created. The event type is an indication of the severity of the detail event log. This enables a software maintainer to specify severity levels for the detail event log entries, which will also be used for the calculation of the highest escalation of the event log shown in 2.1 of Figure 13. All the detail event logs embedded in the main event log are used to calculate the highest escalation after a process has finished. The detail component of the detail event log contains more information corresponding to the specific detail event log entry.

To explain the exception entry in the detail event log, Table 4 shows the components that are contained in the exception event log entries. These entries will only be embedded into the detail event log if an exception has occurred.

Table 4: Preliminary exception event log

Exception event log components	Description
Message	The message contained within the exception when a critical error occurs
Source	The application or object that caused the exception to occur
Stack trace	Immediate frames on the call stack when the exception is thrown

From Table 4, it can be seen that the message of the exception event log is what describes the exception and is contained within the exception event log. Along with the message is the exception source which explains the originating application or object that threw the exception that occurred. Lastly, the exception event log also contains a stack trace which provides more information around the call stack's immediate frames. This can be very useful to a software maintainer in finding the root of an event that occurred within the software application [51].

Preliminary event log implementation algorithm

For the implementation of the proposed preliminary event log structure, Figure 14 explains the algorithm that the software application will have to implement for the event logs to be structured, as shown in Figure 13.

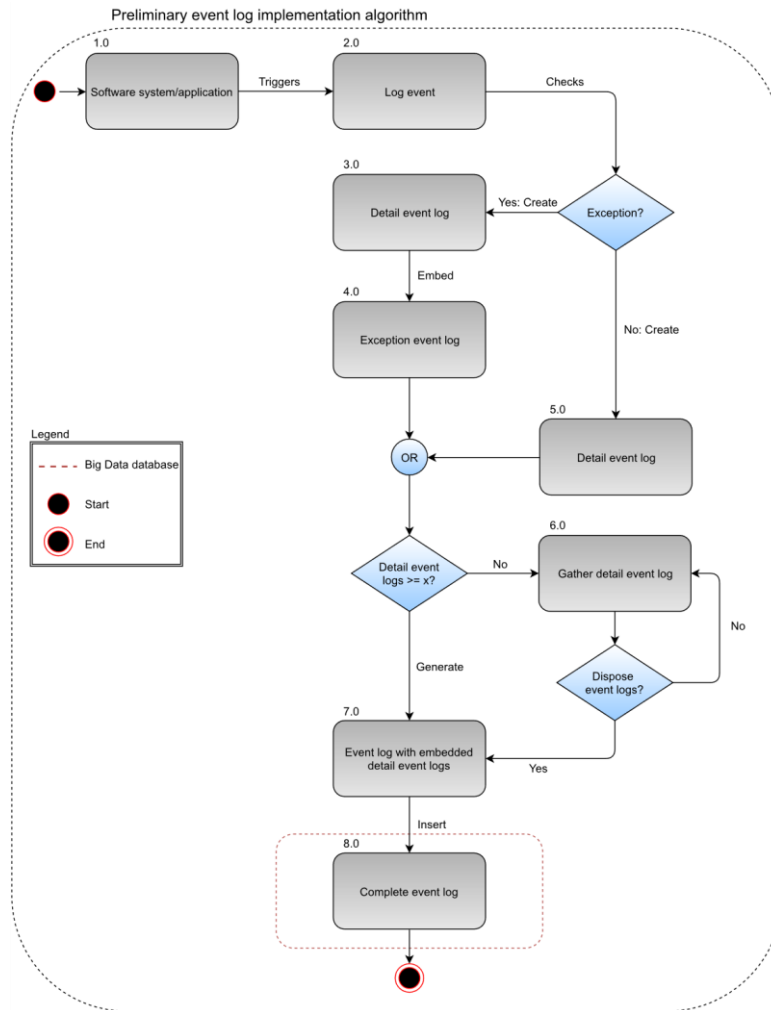


Figure 14: Preliminary event log implementation algorithm

Any generic software system will trigger event logs after the software maintainers have implemented the event log structure in the software application or system. When an event log call is triggered, the event logger will be responsible for creating the event log. These event logs should contain all the communicated information of the software system implemented by the software maintainers.

Firstly, the embedded detail event logs will be created, checking whether the event that has occurred in the software system is an exception that was thrown. If it was an exception, the exception, together with its message, source, and stack trace, will be embedded into the detail event log.

Next, the detail event logs will be collected in batches to ensure that database writes are kept to a minimum. The batch sizes of the detail event logs that are collected will differ depending on the type of software system that is of interest. The application will also have to implement a dispose method which will ensure that the event logs are written to a database at regular

intervals. This can either be at each application instance or as often as a software maintainer requires the event logs to be available.

Finally, the event log will be created, embedding the detail event logs that have been collected during the execution of the software application or system. The final event log, together with the embedded detail event logs, will be written to a NoSQL database to form the complete nested event log structure, as shown in Figure 13.

2.4.2 Final design

The final design is derived from the preliminary event log designs. The preliminary event log structure and implementation algorithm were both important design steps that were taken for designing the final event log structure. The final event log structure and implementation algorithm will be used in the implementation phase.

Final event log structure

The preliminary event log designs were derived from previously gained knowledge and is used for designing the final event log structure together with its implementation algorithm. It is hypothesised that it is the design that will solve the initial problem.

Figure 15 shows the final event log structure derived from the preliminary event log structure. As with the preliminary event log structure, this event log is also structured and can be generically implemented.

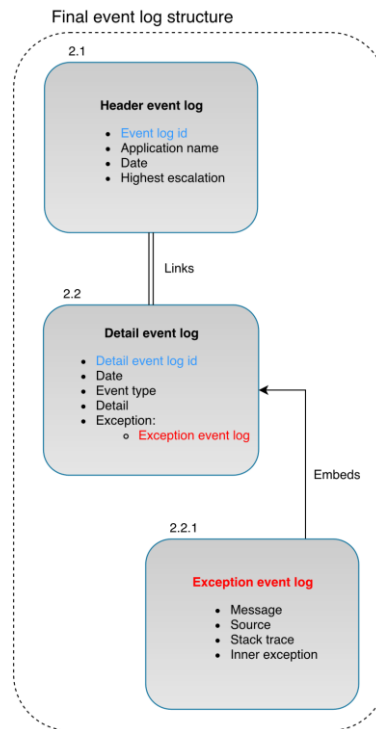


Figure 15: Final event log structure

The preliminary event log structure proposed a nested or folded structure, which meant that the event log embedded a detail event log, which also embedded an exception event log. For the final event log structure, however, it has been decided that a less nested structure will be used. Rather, two separate event logs will be used, namely the header event log and the detail event log. This aims to ease the querying process when the final event log structure is implemented. To explain the final event log structure shown in Figure 15, the following tables, Table 5 - Table 7, can be referred to.

Table 5: Final header event log components

Header event log components	Description
Event log id	The main event log id that is unique to this header event log
Application name	The location/system/application triggering a log event
Date	A timestamp when the event log was created
Highest escalation	Escalation level calculated from the detail log entries

As shown in Table 5, each component that forms part of the main header event log shown in 2.1 of Figure 15 is explained. The reason for referring to this event log as the header event log is because in the final design, the event log will be a separate entry into a database with

no embedded detail event log, as previously shown in the preliminary event log structure design. This is aimed at simplifying the event log structure and reducing the querying complexity.

The event log id in the header event log is the id that will be unique to the header event log entry in the database. The application name is the origin of where a log event trigger that created a header event log entry has occurred. This can also be used as the actor field as mentioned in the literature [35] if an application name is not needed. The date is the creation timestamp which will be used to identify the date and time that the header event log was created. Lastly, the highest escalation is the highest level of the criticality calculated from the detail event logs linked to the header event log. The detail event log components will be explained in Table 6.

Table 6: Final detail event log components

Detail event log components	Description
Detail event log id	Unique id for identifying the detail event log
Date	Timestamp when the detail event log was created
Event type	Criticality or event code of the detail event log
Detail	More information surrounding the detail event log
Exception	If an exception occurred, an exception event log is embedded to the detail event log

Table 6 explains the detail event log shown in 2.2 of Figure 15. The main header event log will link to multiple detail event logs via a corresponding id. The detail event log id which identifies each individual detail event log will be used to link to the id component of the main header event log. This means that multiple detail event logs will be linked to only one header event log.

The date of the detail event log indicates the timestamp when a log event trigger created a detail event log. The event type which will be predetermined by a software maintainer will be used to indicate the criticality of the detail event log. This is also used by the main header event log to determine the detail event log with the highest escalation. This field can also be extended to contain specific event types, stored as an event code, for aggregating by a specific

event code. The detail component will provide more information with regards to the detail event log.

The exception component will contain an exception event log in the case that an exception occurred within the software system or application. Table 7 explains the components contained in the exception event log shown in 2.2.1 in Figure 15. The exception event log will only be optionally embedded in the detail event log in the case that an exception was thrown by the software application or system.

Table 7: Final exception event log components

Exception event log components	Description
Message	The message contained in the thrown exception
Source	The place where the exception originated in the software system or application
Stack trace	Description of the call stack frames
Inner exception	Provides more information around the error that caused the exception that occurred

The message component of the exception event log describes a more readable form of the reason why the exception occurred. The source of the exception is used for retrieving the name of the location that caused the exception to occur. A stack trace is used to provide more internal details of the exception regarding the frames of the call stack which originated at the source. The last component is the inner exception. This is a further component that was added as it can provide the software maintainer with additional information surrounding the error that just occurred when a log event with an exception was triggered.

The final event log structure in Figure 15 will be used in the final event log implementation algorithm. The algorithm will provide information for implementing the final event log structure. The algorithm forms part of 2.0 in Figure 7, explaining how the method for software maintenance can implement the final event log structure.

Final event log implementation algorithm

Figure 16 shows the algorithm to be used by a software system or application to implement the final event log structure, as seen in Figure 15. It also expands 2.0 of the method for software maintenance shown in Figure 7.

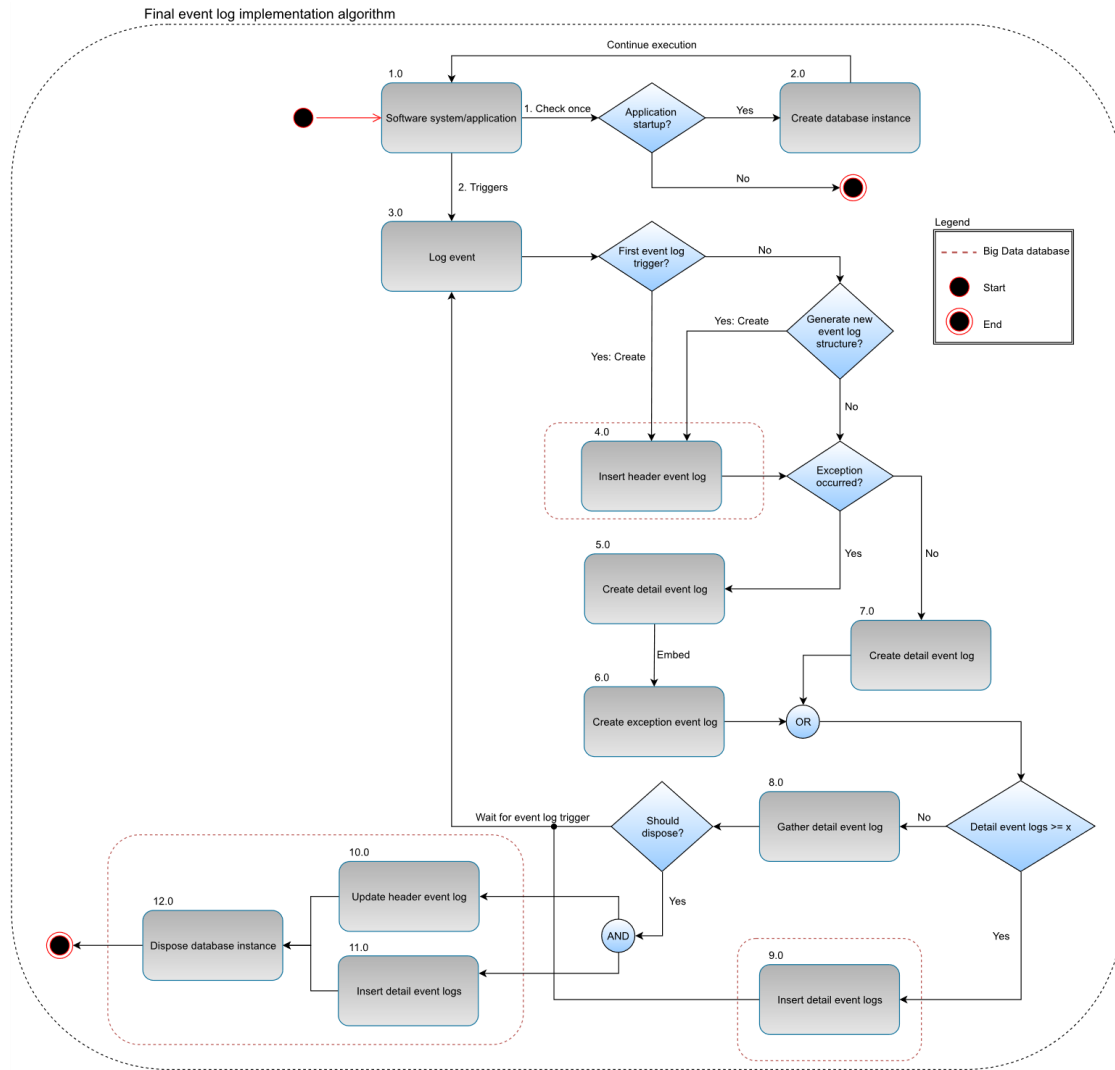


Figure 16: Final event log implementation algorithm

When the software system or application starts, as shown at 1.0 in Figure 16, it will first check if the start-up was successful. In the case that the start-up was successful, an instance of the database will be created and execution will continue, as shown at 2.0. Otherwise, the execution of the application ends.

Once an instance of the database has been successfully created, the application can trigger multiple log events, as seen at 3.0. When an event log is triggered, the algorithm tests whether it is the first event log trigger since the system was started up. If it is the first event log that has been triggered, the header event log will be created, as previously explained in Table 5 and shown at 4.0.

In the case that it is not a first event log trigger by the software system or application, the algorithm will assess whether an exception has occurred. If an exception occurred, a detail

event log will be created, and it will embed an exception event log, as shown at 5.0 – 6.0. If no exception has occurred, only a detail event log will be created, shown at 7.0.

The algorithm also tests whether a new event log structure should be created. This functionality should be decided on by a software maintainer, depending on the application or software system. When a new event log structure entry is necessary, a new header event log will be created, and the procedure is repeated.

Once all the event log components of the event log structure have been constructed, the detail event log entries will be tested for whether a predetermined amount of detail event logs have been collected, as shown in 8.0. If the amount exceeds the predetermined amount, the detail event logs that have been collected will be inserted into the database, as shown in 9.0. If the detail event log entries are less than the predetermined amount, the detail event log will be added to an internal collection in the system or application memory. The gathered event logs, as shown in 8.0, will wait until a dispose is triggered.

When a dispose is triggered, the detail event logs that have been gathered will be inserted into the database as shown in 10.0. Also, the highest escalation component of the header event log will be calculated and the header event log that has been created in 4.0 will be updated as shown in 11.0. Finally, as shown in 12.0, the database instance will be disposed when the software system or application has completed.

2.5 Summary

From the literature study, the requirements for a method for software maintenance using event logs were identified. The main purpose of the method is that it should support software maintainers to reduce the time that is spent on software maintenance. Therefore, a functional design was developed for use by software maintainers, which integrates event logs.

The high-level functional design of the software maintenance method provides an overview of all the functional phases. Each phase of the method builds from the event logs that are the main focal point of the functional design.

Since the event logs are the key focus of the software maintenance method, it was important to use the knowledge that has been previously gained. This provided basic knowledge to develop an event log structure that was used for the design of a preliminary event log structure.

After developing a preliminary event log structure, an algorithm was developed which proposed an implementation method for the structure that has been developed.

It is important to note that a preliminary and a final event log design, as well as an implementation algorithm, had been developed. The preliminary event log structure was a consequential design from previous knowledge, and this is an intermediate phase between the event log structure that was directly derived from the literature, leading up to the final phase. The final phase, which is the final event log structure design along with its implementation algorithm, was derived from the intermediate preliminary event log design and algorithm.

CHAPTER 3 IMPLEMENTATION AND RESULTS

3.1 Preamble

Three case studies are considered in this section, on which the method for software maintenance is applied. The case studies consider three independent software systems. The three software systems each implement the designed event log structure using the implementation algorithm previously discussed.

Each independent software system that is regarded in the case studies runs on a remote server that has already implemented file-based flat unstructured event logs. The file-based event logs were, therefore, also stored on the remote server, resulting in restricted access to the file-based event logs for all the software maintainers. Since access is restricted, the file-based event logs were mostly only used by the software maintainer responsible for all the software systems running on the specific remote server.

Therefore, an example of a file-based unstructured event log which already existed will be discussed and compared to an example of the implementation of the structured event logs. To implement the software maintenance method using structured event logs for each case study, an interface that has been developed will be used. This provides a platform for software maintainers to maintain independent software systems.

To verify that the time spent on software maintenance can be reduced, an online survey was conducted in each case study, tasking 17 software maintainers to use the unstructured event logs and the structured event logs to execute maintenance tasks. Finally, to validate that the problem is solved and the requirements in Section 2.2 have been met, the developed interface was used to show results from the three systems, demonstrating overall system improvements, which would not be possible with unstructured event logs.

3.2 Implementing the method for software maintenance using event logs

The structured event logs will be implemented and generated using the implementation algorithm that was previously designed. It was decided that MongoDB will be the NoSQL database used for storing the generated structured event logs. For this implementation, a MongoDB database was already available in an enterprise environment and therefore chosen. The method for software maintenance will be applied using the designed event logs from section 2.4.2. The main reason for implementing customised structured event logs compared

to existing event log systems, is mainly due to cost implications in an enterprise environment. Additionally, it will also mean that a deployment of additional cloud services or server hardware will be necessary. In addition, this will require maintenance with software maintainer resources already being limited.

Initially, it is expected that there will be resource and cost implications to implement the structured event logs. However, since it will be implemented on already available resources, the maintenance burden and cost due to this should eventually decrease.

The following systems will be considered in Sections 3.3.1 through 3.3.3, which will implement the designed event log structure and implementation algorithm, shown in Figure 15 and Figure 16 respectively:

- Case study 1: Data translation system
- Case study 2: Data extraction system
- Case study 3: Data reporting system

Figure 17 shows an example of unstructured event logs that are representative of a flat file that can be found on each of the systems running in a production environment, which will be discussed in the case studies. These systems contain confidential information and are referred to in a generic form.

```
;2019-07-11,02:26:02,,Error: The following Values failed: 5a213d7f07bd0e206c0ff0a5-2019-07-11 02:00:00-0,04
;2019-07-11,02:26:02,,Error: The following Values failed: 5a213d6707bd0e206c0ff09a-2019-07-11 02:00:00-0,09
;2019-07-11,02:26:02,,Error: The following Values failed: 5a213d5e07bd0e206c0ff096-2019-07-11 02:00:00-28,1
;2019-07-11,02:26:02,,Error: The following Values failed: 5a213d5107bd0e206c0ff090-2019-07-11 02:00:00-8472,7
;2019-07-11,02:26:05,,Error in translating file: C:\DataLogs\AISystems\Data\CompanyX\Inbox\2019-07-11 02-00-00 V01.01 SystemX DATLogger_Log Pumps,DAT
;2019-07-11,02:26:05,,A critical error occurred: CriticalError. The error message is: An error occurred in the MongoDB data push thread. Error code: 1.
;2019-07-11,02:28:00,,Mongo DB client created
;2019-07-11,02:28:01,,Translating file : C:\DataLogs\AISystems\Data\SystemY\Inbox\Untitled attachment_20190711022746875179.txt
;2019-07-11,02:28:01,,Translating file : C:\DataLogs\AISystems\Data\SystemY\Inbox\SystemY New Server_2019-07-11,DAT
;2019-07-11,02:28:01,,TranslationSystem was given a file it cannot handle:C:\DataLogs\AISystems\Data\SystemY\Inbox\Untitled attachment_20190711022746875179.txt
;2019-07-11,02:28:01,,Mongo DB client created
;2019-07-11,02:28:01,,Translating file : C:\DataLogs\AISystems\Data\CompanyX\Inbox\Untitled attachment_20190711022745547061.txt
;2019-07-11,02:28:01,,TranslationSystem was given a file it cannot handle:C:\DataLogs\AISystems\Data\CompanyX\Inbox\Untitled
;attachment_20190711022745547061.txt
;2019-07-11,02:28:01,,Translating file : C:\DataLogs\AISystems\Data\CompanyX\Inbox\2019-07-11 02-30-04 V01.01 UGLogger DATLogger_Log Main Fans,DAT
;2019-07-11,02:28:01,,Translating file : C:\DataLogs\AISystems\Data\CompanyX\Inbox\Untitled attachment_20190711022746250177.txt
;attachment_20190711022746250177.txt
;2019-07-11,02:28:01,,Translating file : C:\DataLogs\AISystems\Data\CompanyX\Inbox\2019-07-11 02-30-04 V01.01 UGLogger DATLogger_Log Pump Temperatures,DAT
;2019-07-11,02:28:01,,Translating file : C:\DataLogs\AISystems\Data\CompanyX\Inbox\2019-07-11 02-30-04 V01.01 UGLogger DATLogger_Log Booster Fans Vibr,DAT
;2019-07-11,02:28:01,,Translating file : C:\DataLogs\AISystems\Data\CompanyX\Inbox\2019-07-11 02-30-04 V01.01 UGLogger DATLogger_Log Booster Fans
;Temperatures,DAT
;2019-07-11,02:28:03,,Translating file : C:\DataLogs\AISystems\Data\CompanyX\Inbox\2019-07-11 02-30-00 V01.01 TGLLogger DATLogger_Log Compressor
;Temperatures,DAT
;2019-07-11,02:28:03,,Error: The following Values failed: 59f85bb7efbb4102f4c242ff-2019-07-11 02:30:00-0
;2019-07-11,02:28:03,,Error: The following Values failed: 5a73228d07bd0d3e107c5d2b-2019-07-11 02:30:00-0
;2019-07-11,02:28:03,,Translating file : C:\DataLogs\AISystems\Data\CompanyX\Inbox\2019-07-11 02-30-00 V01.01 TGLLogger DATLogger_Log Alarm Values,DAT
;2019-07-11,02:28:03,,Translating file : C:\DataLogs\AISystems\Data\CompanyX\Inbox\2019-07-11 02-30-04 V01.01 UGLogger DATLogger_Log Frigde Plant Trips,DAT
;2019-07-11,02:28:03,,Error: The following Values failed: 59f85c4cefb4102f4c243d0-2019-07-11 02:30:00-70
;2019-07-11,02:28:03,,Error in translating file: C:\DataLogs\AISystems\Data\CompanyX\Inbox\2019-07-11 02-30-00 V01.01 TGLLogger DATLogger_Log Compressor
;Temperatures,DAT
;2019-07-11,02:28:03,,A critical error occurred: CriticalError. The error message is: An error occurred in the MongoDB data push thread. Error code: 1.
;2019-07-11,02:28:03,,Translating file : C:\DataLogs\AISystems\Data\CompanyX\Inbox\2019-07-11 02-30-04 V01.01 UGLogger DATLogger_Log PP Vibrations(1),DAT
;2019-07-11,02:28:06,,Error in translating file: C:\DataLogs\AISystems\Data\CompanyX\Inbox\2019-07-11 02-30-04 V01.01 UGLogger DATLogger_Log Booster Fans
;Temperatures,DAT
;2019-07-11,02:28:06,,A critical error occurred: CriticalError. The error message is: An error occurred in the MongoDB data push thread. Error code: 1.
;2019-07-11,02:28:06,,Translating file : C:\DataLogs\AISystems\Data\CompanyX\Inbox\2019-07-11 02-30-00 V01.01 TGLLogger DATLogger_Log Pump Vibrations,DAT
;2019-07-11,02:28:06,,Error: The following Values failed: 59f85c37efbb4102f4c243b3-2019-07-11 02:30:00-0
;2019-07-11,02:28:06,,Translating file : C:\DataLogs\AISystems\Data\CompanyX\Inbox\2019-07-11 02-30-00 V01.01 TGLLogger DATLogger_Log Pump Performance,DAT
```

Figure 17: Example of unstructured event logs

The flat file contains all the information that has been recorded by a software system or application. From the flat files, it can be noted that there are multiple event logs that occur at the same time. Sometimes, software systems run parallel processes which cause the event logs to be logged in non-chronological order. This can cause confusion, not knowing which process logged which event. Another observation is that the different event log types are not grouped or sorted, resulting in a tedious filtering process.

On the other hand, the structured event logs were implemented in all the software systems which, therefore, replaces the unstructured event logs, as previously shown. In Figure 18, an example of structured event logs is shown. As with the previous example of unstructured event logs, the example of structured event logs is also representative of the implemented structured event logs found in the aforementioned systems running in a production environment which will be considered in the case studies to follow.

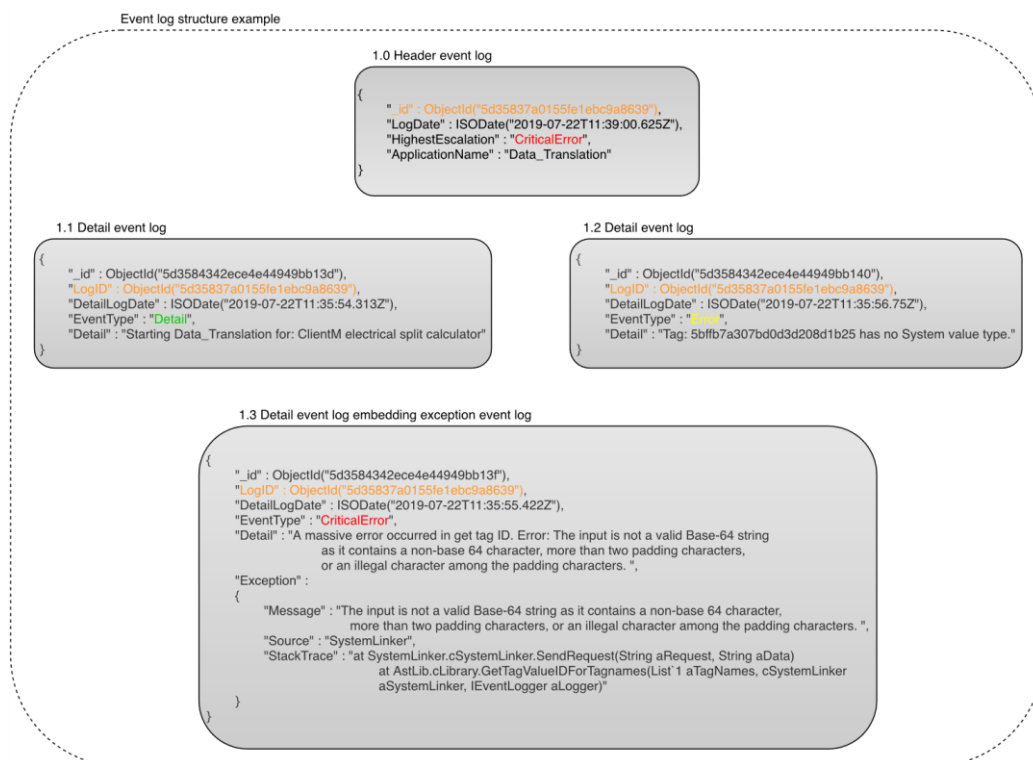


Figure 18: Example of structured event logs

As designed, the example of the structured event logs shown in Figure 18 contains a main event log, called the header event log shown at 1.0, with multiple detail event logs shown from 1.1 to 1.3. The detail log which embeds the exception event log is shown at 1.3. It is embedded when a critical error occurs, providing more information surrounding the critical error.

The implementation algorithm as seen in Figure 16 together with the designed event log structure is implemented as a library written in C# language. This is used across the three individual systems. The library is responsible for generating structured event logs, shown as an example in Figure 18, which are written into MongoDB using the drivers included from MongoDB.

A typical example of how an event used to be logged using the legacy system is shown in Figure 19. This, however, also remains the same for the new library used in the three individual software systems, which ensures that the upgrade to the new library is not time consuming.

```
catch (Exception lException)
{
    Log.LogEvent("Failed to retrieve group bucket name and group key from the database", true, lException, tEventEscalation.CriticalError);
    return cReporterErrorCodes.MainMassiveError;
}
```

Figure 19: Code snippet of an event being logged

To implement the method for software maintenance, as shown in Figure 7, an interface was developed which provides access to the structured event logs of independent software systems.

Software maintainers will use the method for software maintenance on each of the independent software systems to do both simple and more advanced software maintenance, as shown in Figure 7 by using the developed interface shown in Figure 20 as defined in section 2.3. It is important to note that the developed interface relies on the communicated information of the event logs.

The interface contains filters which can be used to search through all the available event logs for the system that has been selected. The event logs can also be filtered by the type of event that occurred. The querying is done by using MongoDB's C# drivers to build pipeline definitions and perform multistage queries. Indexing is added on the date and ID fields of the collections containing the header and detail event logs. Additionally, a text index is also added on the detail field of the detail event logs. All of this is done to increase the performance of the querying operations.

System Logs

Hi, Ryan

Notifications

Help

Home

Logout

Filters

Systems

Data Translation

Data Extraction

Data Reporting

Event Type

All Events

From

2019-07-29 00:00:00

To

2019-07-29 19:15:54

Search Log Event

Search

Search by Log ID

Log ID

Exclude Events Containing

Exclude

Search

Note: Only first 100 results will be displayed

Detailed Event Information

Event Escalation Overview

Event Log Analysis

Event log count: 3

Detail	Log Date	Event Type	Log ID	Exception	More Detail
Example system starting for: Example report	2019-07-29 03:00:16	Detail	5d3e45200155fe1ccc906b9e	No Exception	View
Example tag error: 5bfb7a307bd0d3d208d1b94 has no known value type	2019-07-29 04:48:07	Error	5d3e5e620155fe0bbcc213d0	No Exception	View
Example system completed successfully.	2019-07-29 15:00:04	Detail	5d3eedb80155fe15c0b02f0e	No Exception	View

Results per page: 20

Figure 20: System logs interface – Detailed event information

The first view which implements the method for software maintenance is the detailed event information view. This enables software maintainers to have access to the details of the event logs of the required software system and is immediately available during a system’s runtime. There are filters on the side which can be used to select the required software system, the event type, date range, event log id, and an exclude string which should be ignored when searching the event logs. The software maintainers can retrieve more than 100 results as the view allows to load more at the end of the result set.

If a software maintainer would find it necessary to get an overview of the event types that have occurred for a date range, the event escalation overview table can be used to view a count of each event type for all of the software systems, as seen in Figure 21.

System Logs

Hi, Ryan

Notifications

Help

Home

Logout

Filters

Systems

Data Translation

Data Extraction

Data Reporting

Event Type

All Events

From

2019-07-29

To

2019-07-29

Search Log Event

Search

Search by Log ID

Log ID

Exclude Events Containing

Exclude

Search

Note: Only first 100 results will be displayed

Detailed Event Information

Event Escalation Overview

Event Log Analysis

Application	Info	Detail	Warning	Error	CriticalError
Data Translation	0	0	0	0	0
Data Extraction	0	0	0	0	0
Data Reporting	0	0	0	0	0

Figure 21: System logs interface – Event escalation overview

52

A date range can be selected after which a search can be done for an overview of all the event counts for each event type in the software systems. Typically, this will be used to give an overview of a specific software system to understand whether the system is operating as expected. When errors occur within the software system, the previous detailed event information view can be used to investigate the error.

Shown in Figure 22, a software maintainer can analyse the software system that is of interest. This view provides more advanced software maintenance functionality, as seen in Figure 7. The analysis can be done according to the event type and a date range. Furthermore, the analysis graphs can be used to provide trends of a software system's performance over a predefined search period.

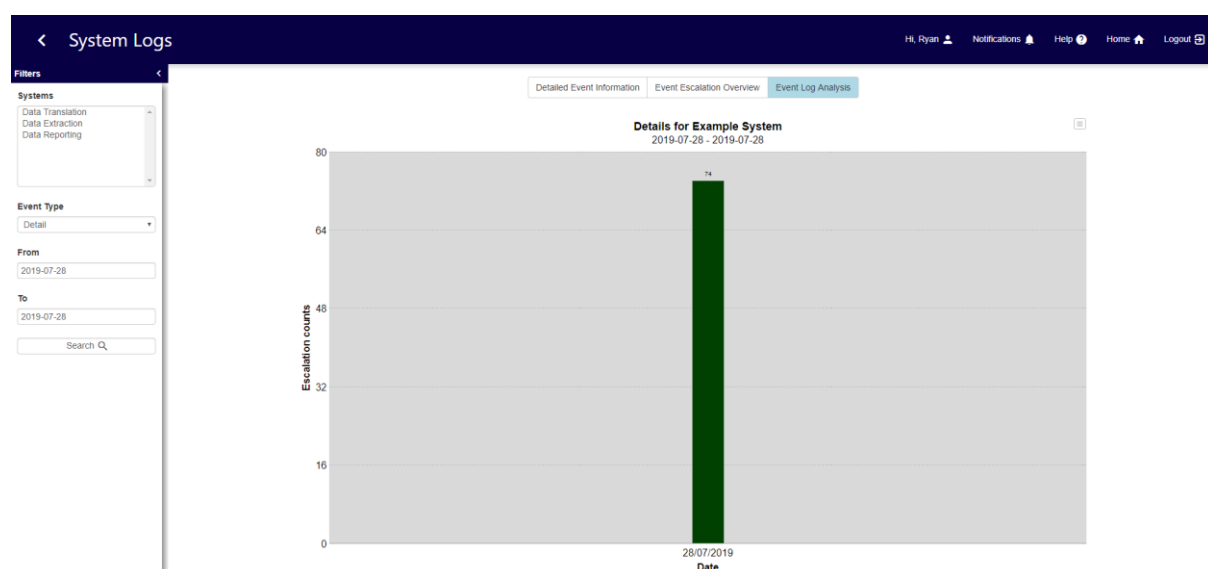


Figure 22: System logs interface – Event log analysis

Using the event log analysis functionality, it is also possible to view the analysis graphs of critical errors which caused exceptions inside the software system. This was implemented into the interface for software maintainers to determine which type of exceptions have occurred over time.

The interface which implements the designed method for software maintenance, integrating structured event logs in all the independent software systems, will be used in the case studies to follow. The aim is to reduce the time that software maintainers spend on software maintenance, therefore addressing the problem and need for the study.

3.3 Verification through case studies

To verify that the solution to the problem highlighted in this study has been correctly implemented, an online survey was conducted. This was specifically aimed at verifying whether the time spent on software maintenance is reduced by the implementation of the method for software maintenance using structured event logs. The survey tasks 17 software maintainers to execute maintenance tasks.

These tasks use the unstructured event logs shown in Figure 17 and the structured event logs shown in Figure 18 to execute the maintenance tasks. The maintenance tasks are explained for each case study by describing the task that the software maintainer had to execute as well as the result that the maintainer had to find. This yielded timed results from the software maintainers which will be interpreted in Section 3.3.4.

To ensure that the tasks in the online survey were not biased toward a specific outcome, the type of question for using the unstructured event logs and structured event logs remained the same. The only changes made to the questions were time ranges and detail to search for in the event logs to ensure that the software maintainer would not already know what to search for between using the old method implementing unstructured event logs and the new method implementing the structured event logs. The order that the unstructured and structured event logs were used by the software maintainers also changed in every case study.

The software maintainers were given background information on the method for software maintenance and how it implements structured event logs. The designed interface was also clearly explained to each software maintainer so as to be familiarised with it before using it in the software maintenance tasks. Each software maintainer was provided with all the available unstructured event logs at the point in time when the online survey was used. This was in the form of event log flat files containing actual event log information of each relevant system described in each case study.

An automatic timer started when the software maintainer initiated the maintenance task for the unstructured or structured event log task. After completing the task, a specific answer to the task was expected and before the answer was provided, the software maintainer was instructed to stop the timer. This was done to ensure consistency over all the maintenance tasks that had to be executed.

After all the maintenance tasks were completed by each individual software maintainer, feedback demonstrating the experience using unstructured and structured event logs were recorded and will be interpreted in Section 3.3.4.

3.3.1 Case study 1: Data extraction system

Background on the data extraction system

The data extraction system is used in this case study and shown in Figure 23. The purpose of this system is to retrieve raw data from a data source which is stored in a NoSQL database. All the data that is extracted from the database is collected and converted into a comma separated values (CSV) format. Each CSV file that is created is transferred to a remote server for use by other systems.

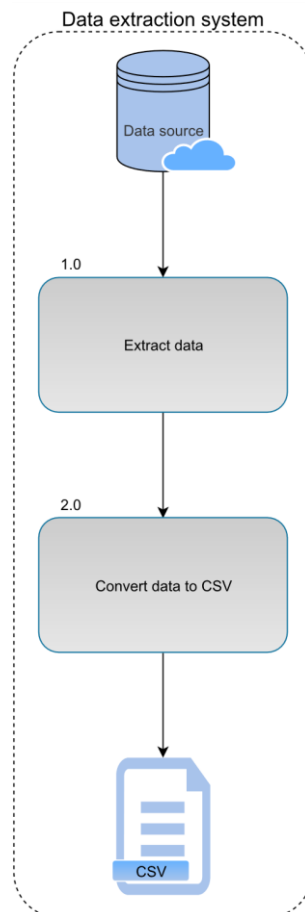


Figure 23: Data extraction system

Using the old and new method for the data extraction system

For the data extraction system, software maintainers were firstly tasked to use the event log files that were provided. This task consisted of finding the amount of event logs that contained

“missing data” between 07:00 and 08:00 on 07-07-2019. This specific question was asked since it was a typical issue that had occurred in the system prior to using the new method for doing software maintenance.

The event log files that had to be used to achieve the result to the maintenance task had to be filtered until the right event log file was found. When the right event log file was found, software maintainers had to manually filter through the event logs by opening the file in any preferred viewer.

In Figure 24, an example of the result that the software maintainers had to retrieve in the data extraction system for the task, as previously described, can be seen. The only change made to the question was the time range which was changed to between 06:00 and 07:00 on 07-07-2019.

The screenshot shows the 'System Logs' interface. On the left, there are filters for 'Applications' (Data Extraction, Data Reporting, Data Translation), 'Event Type' (All Events), 'From' (2019-07-07 06:00:00), 'To' (2019-07-07 07:00:00), 'Search Log Event' (missing data), 'Search by Log ID' (Log ID), and 'Exclude Events Containing' (Exclude). The main area displays a table of log entries with columns: Detail, Log Date, Event Type, Log ID, Exception, and More Detail. The table shows 24 results, with the first few rows indicating missing data for various reports and load shifts.

Detail	Log Date	Event Type	Log ID	Exception	More Detail
For report: Daily load shift report ID: 936 missing data was found.	2019-07-07 06:10:34	Detail	5d2170b90155fe0ad8701d9	No Exception	View Q
Report Estimated load shift missed opportunity tariff_R/kWh has missing data for the dates: 2019-07-06	2019-07-07 06:10:34	Detail	5d2170b90155fe0ad8701d9	No Exception	View Q
Report has missing data for: Report Estimated load shift missed opportunity tariff_R/kWh has missing data for the dates: 2019-07-06	2019-07-07 06:10:34	Detail	5d2170b90155fe0ad8701d9	No Exception	View Q
There was missing data for report Daily load shift report	2019-07-07 06:10:34	Warning	5d2170b90155fe0ad8701d9	No Exception	View Q
For report: Daily load shift report ID: 940 missing data was found.	2019-07-07 06:10:35	Detail	5d2170ba0155fe07001f362	No Exception	View Q
Report Estimated load shift missed opportunity tariff_R/kWh has missing data for the dates: 2019-07-06	2019-07-07 06:10:35	Detail	5d2170ba0155fe07001f362	No Exception	View Q
Report has missing data for: Report Estimated load shift missed opportunity tariff_R/kWh has missing data for the dates: 2019-07-06	2019-07-07 06:10:35	Detail	5d2170ba0155fe07001f362	No Exception	View Q
There was missing data for report Daily load shift report ID 564	2019-07-07 06:10:35	Warning	5d2170ba0155fe07001f362	No Exception	View Q
For report: Daily load shift report ID: 941 missing data was found.	2019-07-07 06:10:40	Detail	5d2170bf0155fe104426a60	No Exception	View Q
Report Estimated load shift missed opportunity tariff_R/kWh has missing data for the dates: 2019-07-06	2019-07-07 06:10:40	Detail	5d2170bf0155fe104426a60	No Exception	View Q
Report has missing data for: Report Estimated load shift missed opportunity tariff_R/kWh has missing data for the dates: 2019-07-06	2019-07-07 06:10:40	Detail	5d2170bf0155fe104426a60	No Exception	View Q
There was missing data for report ID: 235	2019-07-07 06:10:40	Warning	5d2170bf0155fe104426a60	No Exception	View Q
For report: Daily load shift report ID: 936 missing data was found.	2019-07-07 06:40:21	Detail	5d2177b30155fe146c70e8a	No Exception	View Q
Report Estimated load shift missed opportunity tariff_R/kWh has missing data for the dates: 2019-07-06	2019-07-07 06:40:21	Detail	5d2177b30155fe146c70e8a	No Exception	View Q
Report has missing data for: Report Estimated load shift missed opportunity tariff_R/kWh has missing data for the dates: 2019-07-06	2019-07-07 06:40:21	Detail	5d2177b30155fe146c70e8a	No Exception	View Q
There was missing data for report Daily load shift report ID 897	2019-07-07 06:40:21	Warning	5d2177b30155fe146c70e8a	No Exception	View Q
For report: Daily load shift report ID: 245 missing data was found.	2019-07-07 06:40:27	Detail	5d2177b90155fe1e7480ce59	No Exception	View Q
Report Estimated load shift missed opportunity tariff_R/kWh has missing data for the dates: 2019-07-06	2019-07-07 06:40:27	Detail	5d2177b90155fe1e7480ce59	No Exception	View Q
Report has missing data for: Report Estimated load shift missed opportunity tariff_R/kWh has missing data for the dates: 2019-07-06	2019-07-07 06:40:27	Detail	5d2177b90155fe1e7480ce59	No Exception	View Q
There was missing data for report Daily load shift report ID 645	2019-07-07 06:40:27	Warning	5d2177b90155fe1e7480ce59	No Exception	View Q

Figure 24: Case study 1 – Data extraction maintenance task

The software maintainers used the event log interface to retrieve the details of the “missing data” events that had occurred in the data extraction system. The from and to date could be selected to narrow down the search as well as a text part of the event logs that contained the missing data.

Verification results using the online survey

The following result data were collected from the online survey. The timed results are shown for each individual software maintainer in Figure 25. The order of this maintenance task was

to use the unstructured event log files first, following the same task but with different parameters to search for using the event log interface.

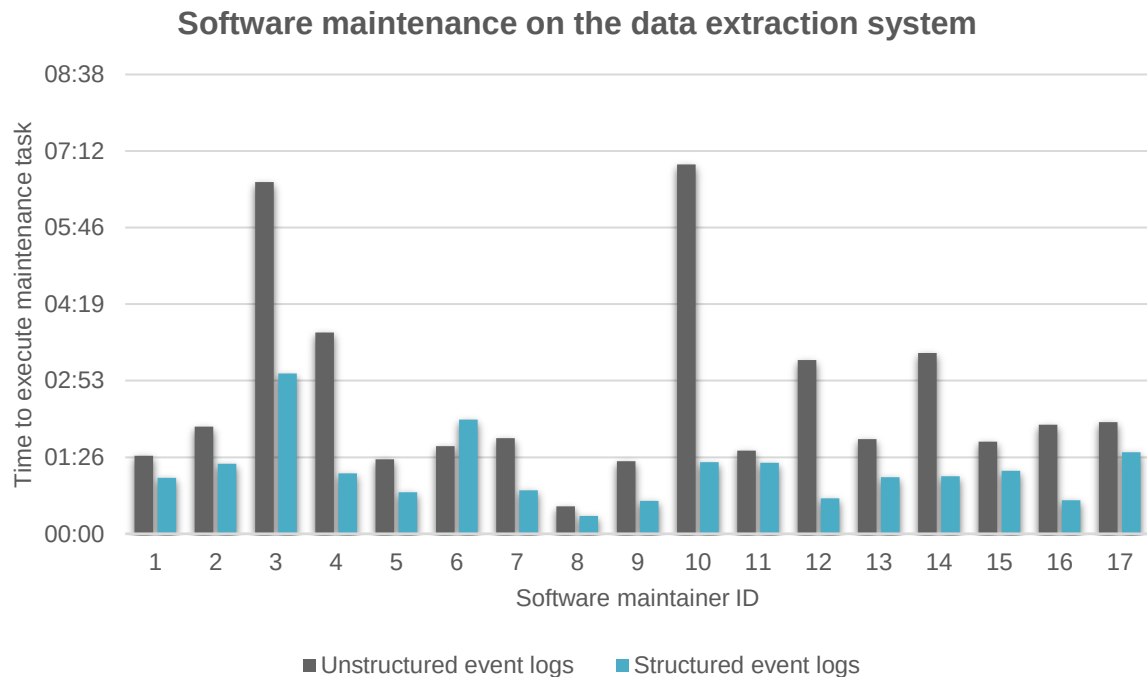


Figure 25: Software maintenance on the data extraction system

From the results seen in Figure 25, all the software maintainers, except for the software maintainer with $ID = 6$, returned time improvements when using the new method for executing a typical maintenance task. This could be due to a mistake made when using the filters provided in the event log interface while searching for the event logs.

It is also notable that software maintainers that took considerably longer to execute the maintenance task using the unstructured event logs showed a larger time improvement over the other software maintainers. Maintainers using less time to execute the maintenance task showed almost the same time spent on the interface using the structured event logs.

To summarise what had been shown in Figure 25 and to determine whether there was an overall time improvement for executing a maintenance task in the data extraction system, Figure 26 is shown.

Average time spent on maintaining the data extraction system

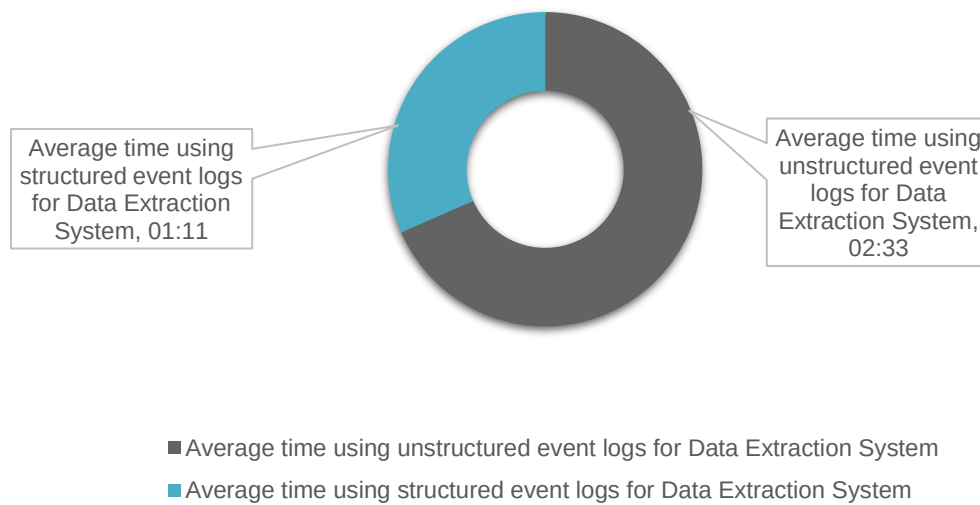


Figure 26: Average time spent on maintaining the data extraction system

The average time spent by all the software maintainers is shown in Figure 26. A time improvement can be observed by the portions of the circle. The maintenance task using the unstructured event logs saw an average time of 02:33, and for the structured event logs, an average time of 01:11 was observed.

Overall, this translates to a time improvement for the same maintenance task in the data extraction system of approximately 54%.

3.3.2 Case study 2: Data translation system

Background on the data translation system

For this case study, the data translation system that will be considered is briefly explained in Figure 27. The data translation system is mainly responsible for gathering data from different external data sources in various formats. This system is used to generate a custom format which is used to ensure consistency throughout processes that use the translation system.

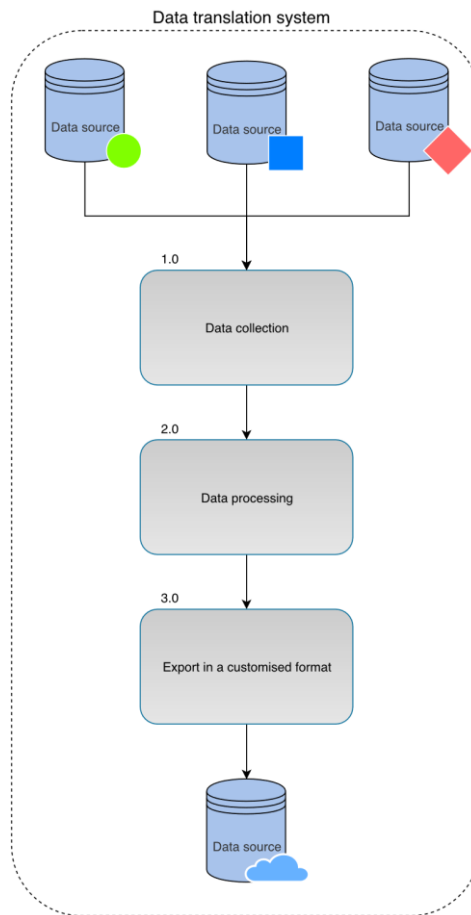


Figure 27: Data translation system

Data is collected from each of the different data sources as an input for the translation system. After all the different data in their variable data sources have been collected, the data is processed and converted to a standard format. This will be recognised by the processes that use the translation system.

Using the old and new method for the data translation system

For the data translation system, software maintainers were tasked to find all critical errors that occurred between 09-07-2019 00:00:00 and 11-07-2019 23:59:59 that contained a message in the details of the event log, namely "Connection to MySQL failed". The reason for this specific question was that this critical error had occurred in the translation system, and the specific date was not available for pinpointing the exact time.

The translation system performed as expected on 08-07-2019 and software maintainers only realised the translation system did not output any customised format data in the database on 11-07-2019. Therefore, the critical error had to be between the dates mentioned in the maintenance task.

Firstly, software maintainers were tasked to use the event log interface to execute the maintenance task. Figure 28 shows the result that the software maintainers had to find.

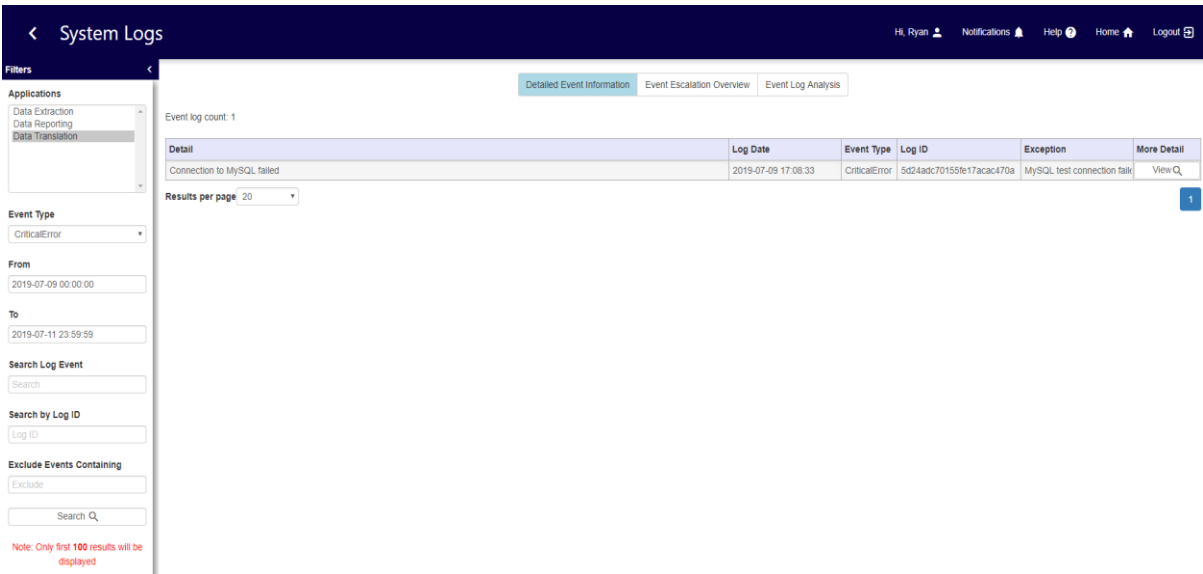


Figure 28: Case study 2 – Data translation maintenance task

Using the unstructured event logs in the files, software maintainers were tasked to provide an occurrence count of critical error events from 10-07-2019 00:00:00 to 12-07-2019 23:59:59 which contained "Connection to MySQL failed" in the details of the event log. The answer to this question also resulted in only one event log between three different event log files. This question was asked in the same format as the maintenance task that had to be executed using the event log interface. This was done to ensure that there would not be biased time results between the unstructured and structured event logs.

What would have made this maintenance task more complex using the unstructured event log files was that the software maintainers had to filter through three different event log files to find the specific event log entry.

Verification results using the online survey

Using the online survey, the results of the data translation system for comparing the unstructured event logs with the structured event logs were collected. Shown in Figure 29 are the results showing the time for each individual software maintainer executing the maintenance task previously mentioned.

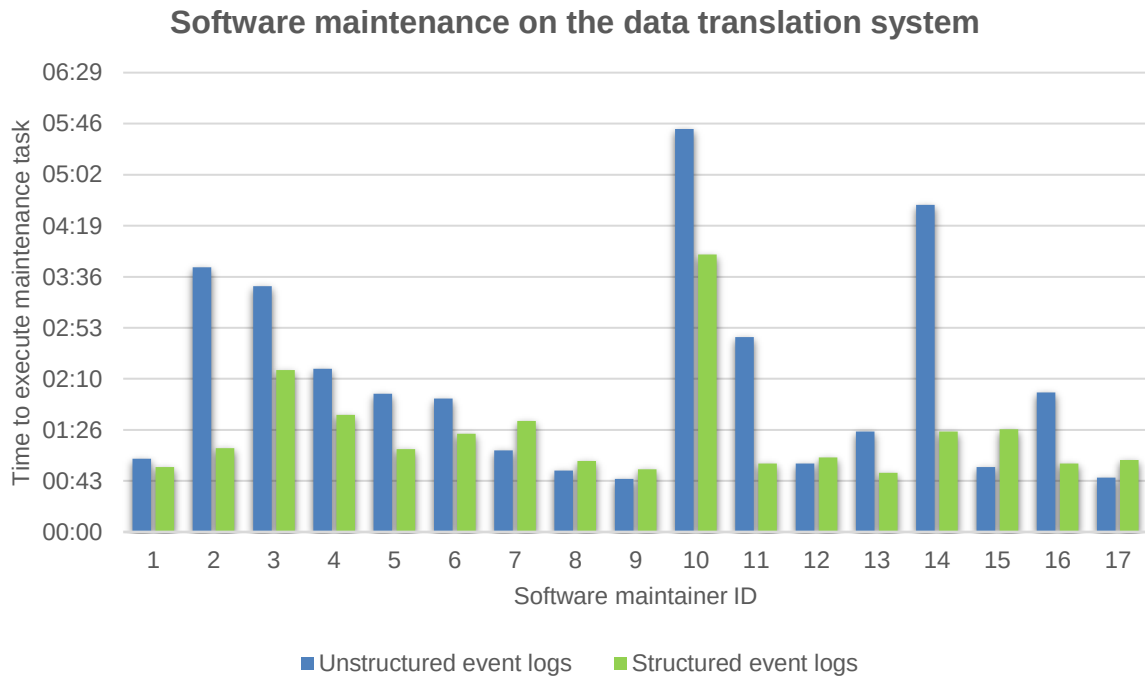


Figure 29: Software maintenance on the data translation system

Analysing the results obtained in Figure 29, it can be seen that there are overall time improvements when comparing the unstructured event logs to the structured event logs. Another observation is that for this case study, there were six software maintainers spending more time on the maintenance task while using the structured event logs compared to the unstructured event logs.

However, what is noticeable from this observation is that the time difference is negligibly small between using the unstructured event logs and structured event logs. Since the query had to execute and retrieve data over a few days on the structured event logs, a plausible reason for the extended time taken for software maintainers to execute the maintenance task could be due to this.

Nevertheless, in this case study, 11 out of the 17 software maintainers presented a time reduction when using the new method which implements structured event logs over the old method which implements unstructured event logs. Again, the software maintainers who were spending more time using the old method, presented larger time improvements using the new method.

Figure 30 summarises what had been observed in Figure 29. This shows the average time that was spent by all the software maintainers on the maintenance task for the data translation system.

Average time spent on maintaining the data translation system

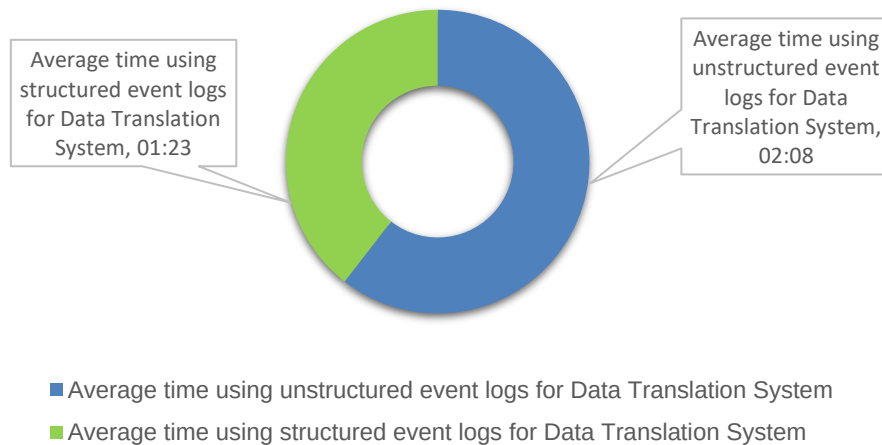


Figure 30: Average time spent on maintaining the data translation system

To determine whether there were any time improvements, Figure 30 shows a pie chart, and by visually analysing the portion of the circles, it can be seen that there was an overall time improvement when using the structured event logs in the new method compared to the unstructured event logs.

The average time using the unstructured event logs was 02:08, compared to the structured event logs which was 01:23. This yielded an average time improvement of 0:44. Therefore, this results in a time improvement for this case study of approximately 35%.

3.3.3 Case study 3: Data reporting system

Background on the data reporting system

The data reporting system that will be considered in this case study is mainly responsible for processing Microsoft Excel files. In Figure 31, the high-level processes of the data reporting system are shown.

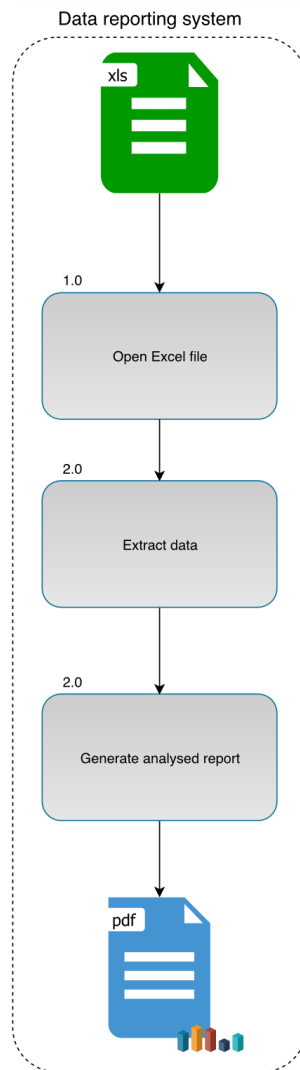


Figure 31: Data reporting system

Every Excel file in the data reporting system is opened automatically and all the different sheets inside of the file which contain data are read. The data from each sheet is extracted, which is used to generate report sheets.

Each report sheet in the file has a standard template which is populated with the extracted data, and finally a PDF report is generated containing all the information processed and populated by the data reporting system.

Using the old and new method for the data reporting system

For the data reporting system, software maintainers were firstly tasked to use the unstructured event logs in the files to execute the maintenance task. Software maintainers had to find the frequency at which a report called “Water Consumption Report” started processing between 08-07-2019 00:00:00 and 11-07-2019 23:59:59 in the data reporting system.

Since the data reporting system is responsible for generating PDF reports, it is important to understand the throughput rate of the reports that had processed in a specified date range in the data reporting system.

Shown in Figure 32 is the example of the result that software maintainers had to retrieve when executing the maintenance task explained below.

The screenshot shows a web application titled 'System Logs'. On the left is a sidebar with filters for Applications (Data Extraction, Data Reporting, Data Translation), Event Type (All Events), From (2019-07-08 00:00:00), To (2019-07-11 23:59:59), Search Log Event (Electricity Consumption Report), Search by Log ID, and Exclude Events Containing. The main area displays a table of event logs with columns: Detail, Log Date, Event Type, Log ID, Exception, and More Detail. The table shows 12 results, all of which are 'Starting Date Reporting System for: Electricity Consumption Report' with no exceptions. A note at the bottom states: 'Note: Only first 100 results will be displayed'.

Detail	Log Date	Event Type	Log ID	Exception	More Detail
Starting Date Reporting System for: Electricity Consumption Report	2019-07-08 12:22:43	Detail	5d2319730155fe142c0008eb	No Exception	View Q
Starting Date Reporting System for: Electricity Consumption Report	2019-07-09 08:15:16	Detail	5d2430f40155fe1d4c406a51	No Exception	View Q
Starting Date Reporting System for: Electricity Consumption Report	2019-07-09 08:59:09	Detail	5d243b3d0155fe1c804eb717	No Exception	View Q
Starting Date Reporting System for: Electricity Consumption Report	2019-07-10 06:46:49	Detail	5d256db90155fe1c3c60211	No Exception	View Q
Starting Date Reporting System for: Electricity Consumption Report	2019-07-10 07:42:23	Detail	5d257ab0155fe18a4868d2d	No Exception	View Q
Starting Date Reporting System for: Electricity Consumption Report	2019-07-10 08:33:47	Detail	5d2586ca0155fe1b306d89a0	No Exception	View Q
Starting Date Reporting System for: Electricity Consumption Report	2019-07-10 09:52:07	Detail	5d2599270155fe1b80ecf26	No Exception	View Q
Starting Date Reporting System for: Electricity Consumption Report	2019-07-10 09:54:37	Detail	5d2599b40155fe1b6b29f4a	No Exception	View Q
Starting Date Reporting System for: Electricity Consumption Report	2019-07-11 06:16:52	Detail	5d2644cc0155fe0b94a4f4b6	No Exception	View Q
Starting Date Reporting System for: Electricity Consumption Report	2019-07-11 06:29:22	Detail	5d2647420155fe0b645af79d	No Exception	View Q
Starting Date Reporting System for: Electricity Consumption Report	2019-07-11 06:31:36	Detail	5d2647cb0155fe146c22087	No Exception	View Q
Starting Date Reporting System for: Electricity Consumption Report	2019-07-11 11:02:29	Detail	5d26f8250155fe0b4837296c	No Exception	View Q

Figure 32: Case study 3 – Data reporting maintenance task

Using the structured event logs in the interface, software maintainers were tasked to find the amount of times that a report called “Electricity Consumption Report” had processed for the same time range as the unstructured event log files. The two maintenance tasks returned to the same results to ensure that biasing was counteracted when using the old method with unstructured event logs compared to the structured event logs in the new method.

From Figure 32 it can be seen that the report processed multiple times on each day in the date range. Using the structured event logs in the interface, it was an easier task to execute compared to using the unstructured event logs in the files. Multiple files had to be opened by software maintainers to find the amount of times that the specific reports processed.

Verification results using the online survey

Using the online survey, the results shown in Figure 33 can be analysed to determine whether there was a time increase or decrease per software maintainer executing the maintenance tasks previously mentioned.

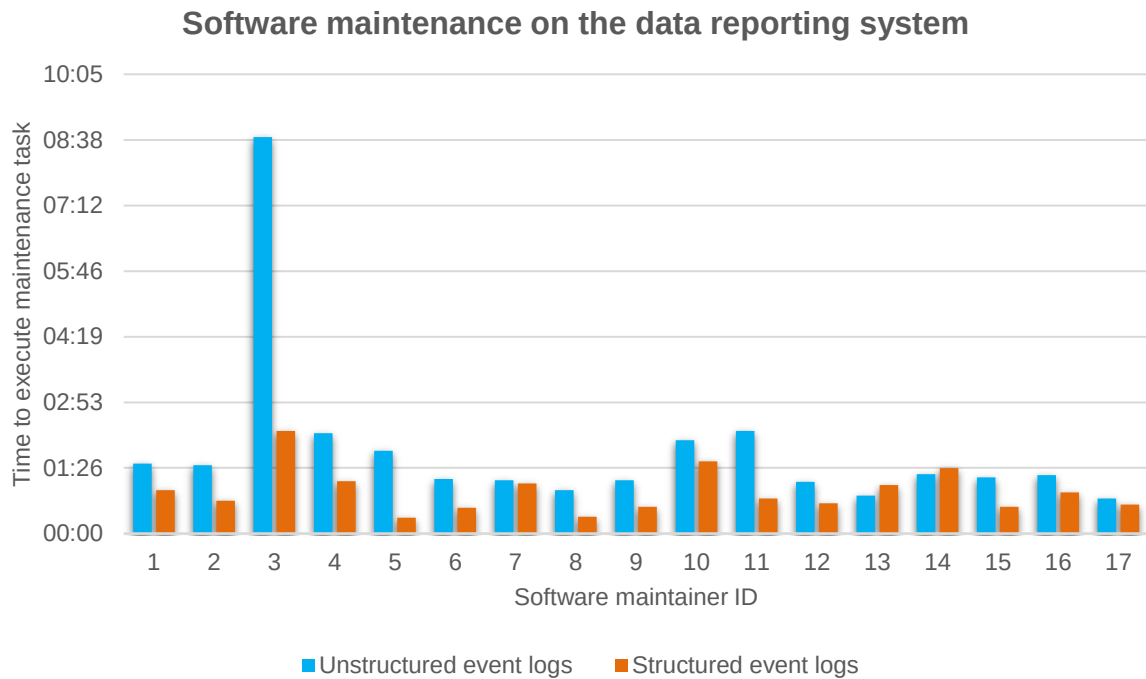


Figure 33: Software maintenance on the data reporting system

It can be seen from Figure 34 that there were two software maintainers consuming more time using the structured event logs in the new method by using interface. Again, the time difference between using the unstructured event logs compared to the structured event logs are insignificantly smaller.

The contributing factor to this could be due to a connection issue from the software maintainers' side since the time it took to return the results was longer than expected. It could also be due to the software maintainer using the interface's filters incorrectly or struggling to retrieve the correct results due to the slight change in the maintenance task that had to be executed.

Overall, 15 out of the 17 software maintainers proved time improvements when using the structured event logs in the interface. An observation that can be made is the outlier at software maintainer with $ID = 3$ took longer to execute the unstructured event log maintenance task.

This is possibly weighted toward the software maintainer finding it hard to use multiple unstructured event log files to find all the reports in the maintenance task. This also demonstrates that using the structured event logs in the interface significantly reduced the time it took to execute the maintenance task.

The average time spent on the maintenance task for the data reporting system is shown in Figure 34.

Average time spent on maintaining the data reporting system

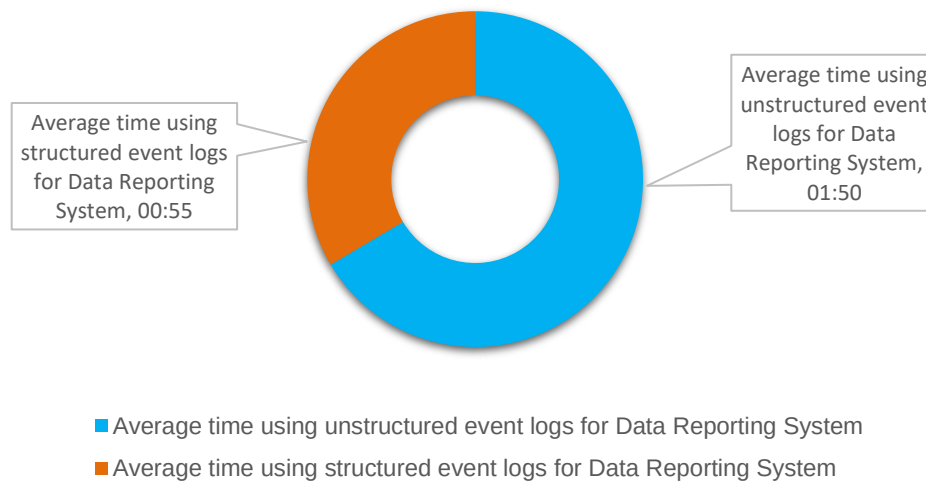


Figure 34: Average time spent on maintaining the data reporting system

Analysing the chart in Figure 34, it can be observed that the average time it took software maintainers to execute the maintenance task using the structured event logs was considerably less compared to using the unstructured event logs in the new method.

The maintenance task using the unstructured event logs took 01:50 and using the structured event logs took 0:55. This results in an improvement of approximately 50% for executing the same maintenance task, but with a different search parameter.

3.3.4 Interpretation of verification results from case studies

To verify that the method for software maintenance which implements structured event logs met the specifications and objectives that were derived from the need for the study, a summary of the results found in the three case studies is shown in Figure 35. Software maintainers are denoted as “SM” in Figure 35. The main objective of the method for software maintenance implementing the structured event log design and integrated in all three case studies, was to reduce the time spent on software maintenance.

If the method for software maintenance is correctly implemented, there should be an overall time decrease across all the case studies. This will verify that the proposed solution to the problem was correctly implemented.

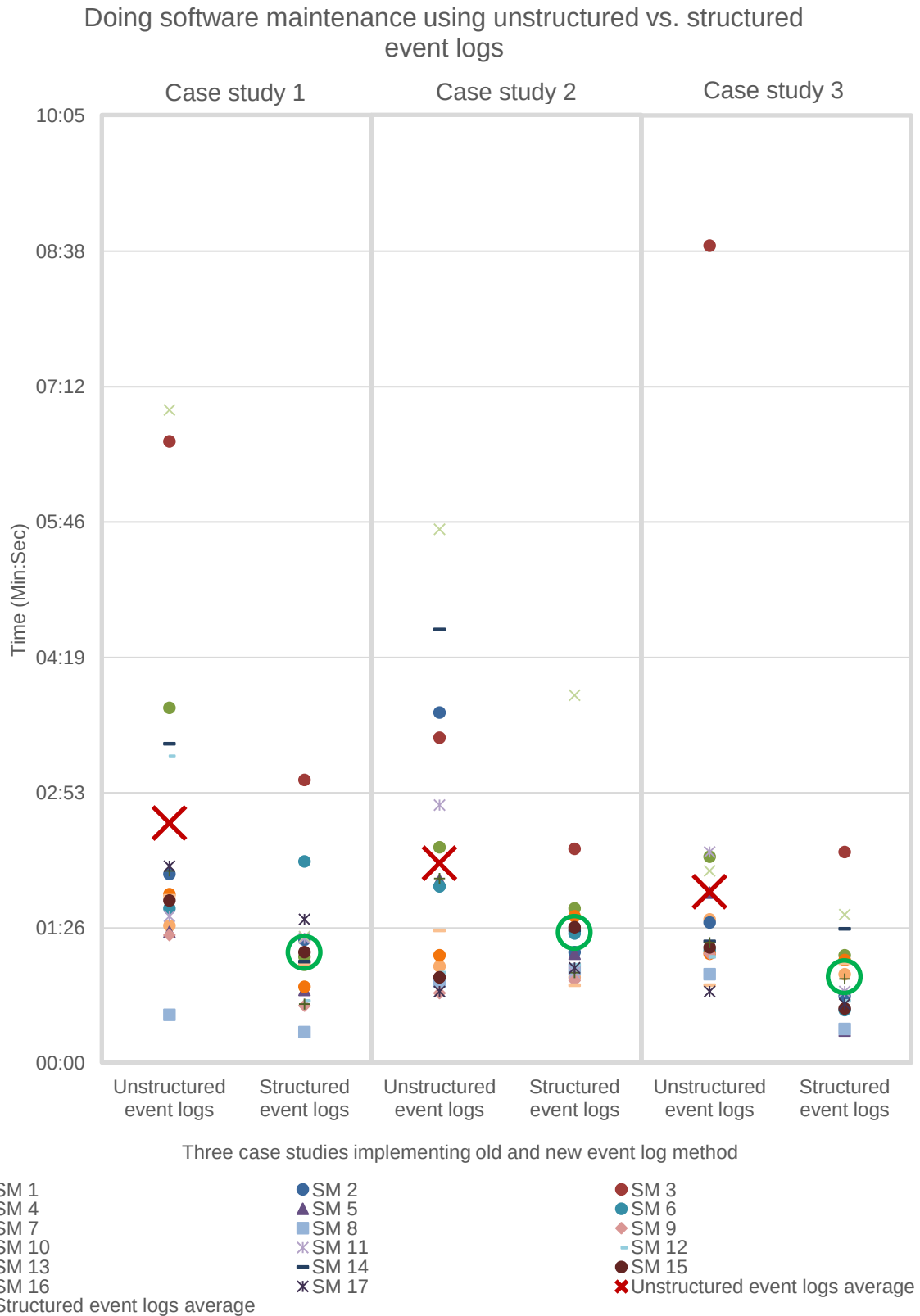


Figure 35: Software maintenance using unstructured vs. structured event logs

From the summary in Figure 35, the comparisons between the unstructured and structured event logs can be seen. It is important to keep in mind that the order in which the case studies were completed is chronological, as shown in Figure 35.

It is evident that there was one outlier, SM 3, across all the case studies when comparing the unstructured event logs to the structured event logs. This resulted in higher averages for each case study. Another observation is that the average time consumed during each case study using the unstructured event logs is less for each case study. This was expected since it is possible for software maintainers to find a better way to filter through the multiple unstructured event logs in the files.

Nevertheless, the overall time improvements can be seen for each case study. Case study 1 showed the largest time improvement, with case study 3 showing the second largest time improvement. From these results, it demonstrates that changing the order in which the software maintenance tasks were executed counteracted biasing toward a certain positive outcome.

Shown in Figure 36 are the average times between using the unstructured event logs compared to the structured event logs summarised for all the case studies.

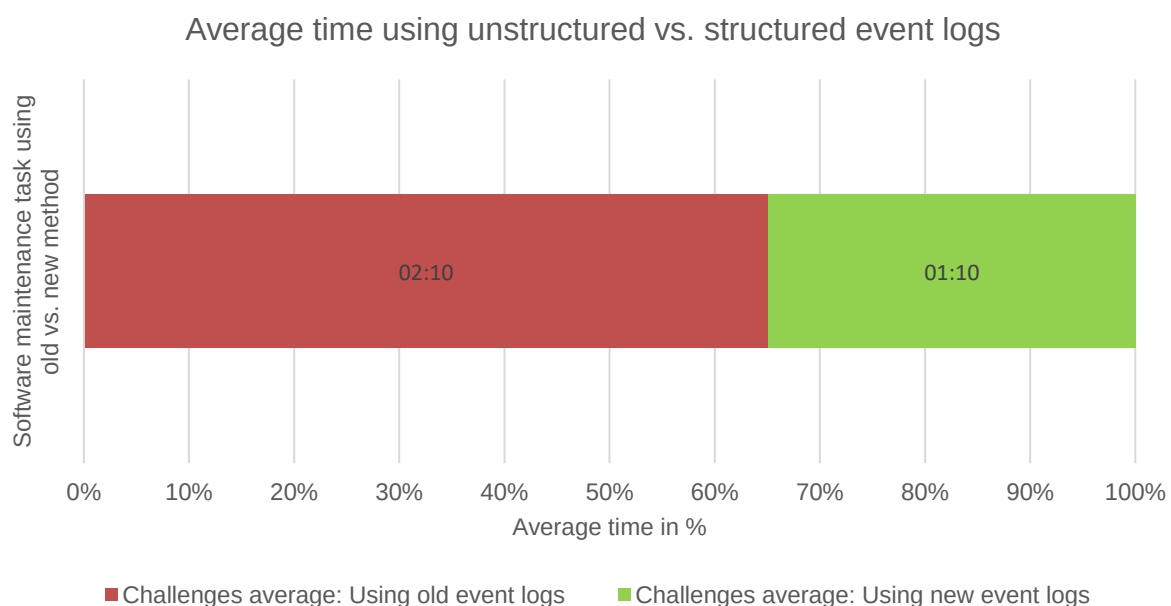


Figure 36: Average time using unstructured vs. structured event logs

Overall, the average time improvement across all three case studies was approximately 47%. This means that, on average, software maintainers consumed almost half as much time using the structured event logs compared to the unstructured event logs.

3.4 Validation

To validate that the solution that was developed addresses the problem and the need for the study, the online survey results were used. This was done by evaluating the timestamps at which the software maintainers accessed the event logs to validate the availability of the event logs. The experiences of the software maintainers using the event logs are also discussed regarding the accessibility, ease of use, and structure of the event logs. This was also used for validation from the software maintainers' perspective.

From the online survey that had been completed by the 17 software maintainers, the access timestamps were also recorded to validate that the structured event logs are available and can be accessed at any time. This is shown in Table 8.

Table 8: Online survey - Structured event logs access timestamps

Date and time	Software maintainer ID
2019/07/12 10:12:31 AM	1
2019/07/12 11:37:00 AM	2
2019/07/12 11:45:05 AM	3
2019/07/12 12:10:36 PM	4
2019/07/12 12:13:34 PM	5
2019/07/12 2:46:09 PM	6
2019/07/12 2:47:52 PM	7
2019/07/12 3:40:12 PM	8
2019/07/12 3:42:41 PM	9
2019/07/15 9:06:14 AM	10
2019/07/15 9:49:03 AM	11
2019/07/15 11:19:45 AM	12
2019/07/15 3:48:50 PM	13
2019/07/16 2:45:47 PM	14
2019/07/16 9:40:00 PM	15
2019/07/17 8:40:19 AM	16
2019/07/17 10:31:53 AM	17

From Table 8, the timestamps when the structured event logs were accessed varied throughout the days that the online survey was conducted. An important observation that can be made is that software maintainer 15 conducted the survey and accessed the structured event logs at 21:40, which is out of typical office hours. This means that the structured event logs can be accessed at any given time from a central location.

Software maintainers' experience using unstructured vs. structured event logs

Furthermore, to validate that the event logs are accessible and structured, the experience of the software maintainers using the unstructured event logs in the old method and the structured event logs in the new method will be analysed.

Figure 37 shows the effort level software maintainers experienced during the software maintenance tasks. The reason for asking the following questions were to retrieve experience results from the software maintainers using the structured event logs. This forms part of ISO/IEC 14764 [21], where the effort level should be determined as part of software measurement of the different types of maintenance, as discussed in Section 1.3.

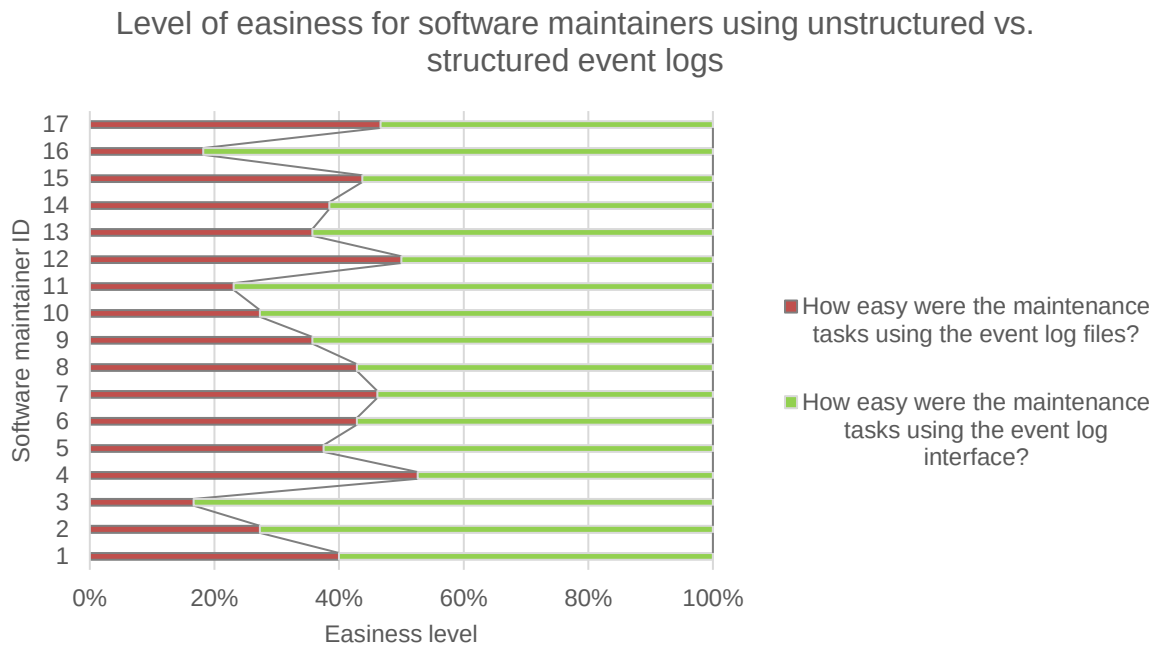


Figure 37: Ease of use of unstructured vs. structured event logs

From Figure 37, the software maintainers were asked to score how easy it was to use the unstructured event logs in the files compared to the structured event logs. It can be seen that, overall, software maintainers found it easier to use the structured event logs. There was only one software maintainer, with ID = 4, who found it easier to use the unstructured event logs. Software maintainer with ID = 12 found it equally easy using the unstructured event logs compared to the structured event logs.

Software maintainers were asked to score how structured the event logs were when using the event log files. This can be seen in Figure 38. Software maintainers were allowed to

experience both the old and the new method, using the unstructured event logs in the files and the structured event logs in the interface, as structured or unstructured.

Were the event logs structured in the files (old method)?

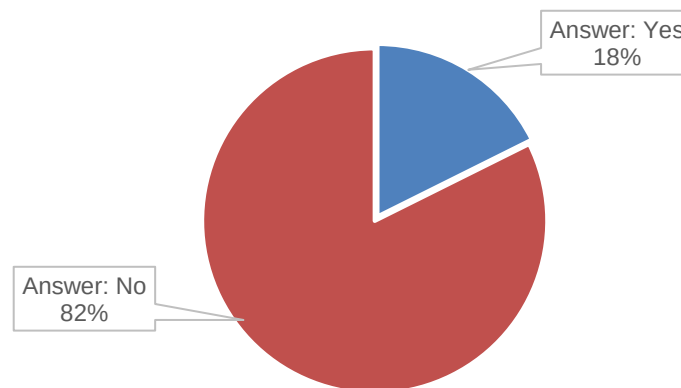


Figure 38: Event log structuring using old method

From Figure 38 it is clear that software maintainers found the event logs using the old method unstructured. Only 3 out of the 17 software maintainers experienced the event logs structured compared to 14 out of the 17 software maintainers experiencing the event logs unstructured.

Figure 39 demonstrates the software maintainers' experience by using the new method for doing software maintenance using the structured event logs.

Were the event logs structured in the interface (new method)?

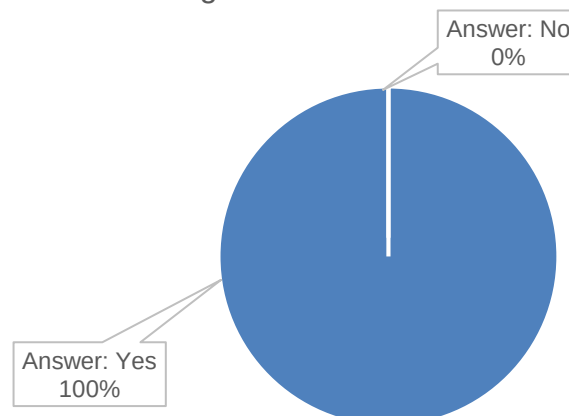


Figure 39: Event log structuring using new method

Compared to the results in Figure 38, all the software maintainers experienced the structured event logs in the interface as structured. The result from Figure 38 and Figure 39 validates that the event logs were structured in the new method compared to the old method.

Finally, software maintainers were asked to score the experience on how accessible the event logs were using the unstructured event logs in the files compared to the structured event logs in the interface which implemented the new method for software maintenance. The results are shown in Figure 40.

Accessibility of unstructured vs. structured event logs

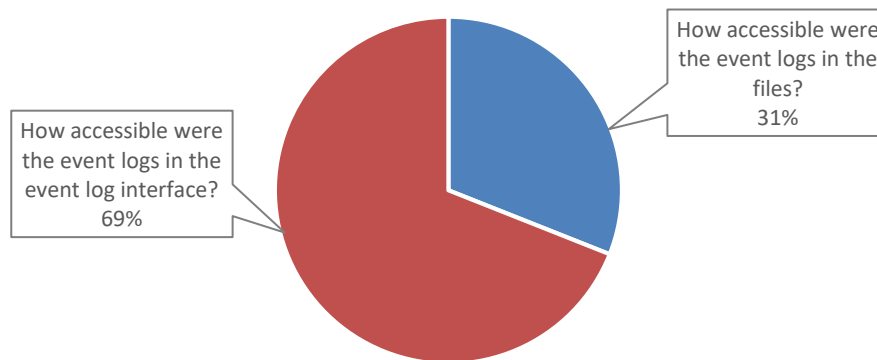


Figure 40: Accessibility of unstructured vs. structured event logs

From the chart in Figure 40, it can be seen that software maintainers found the event logs more accessible in the interface that implements the structured event logs compared to the unstructured event logs in the files. Five out of the 17 software maintainers found the unstructured event logs in the files accessible compared to 12 out of the 17 of the software maintainers that found the structured event logs accessible in the interface, which validates that the structured event logs are accessible.

Another validation that was done specifically focussed on using the structured event logs for analysis to improve the software systems' performance by executing aggregated pipeline queries on the event types of the detail event logs in the background of the developed interface. This is done automatically and stored for a month. This enhances the querying performance when retrieving the aggregated information to generate the graphs on the developed interface. For this, the scenarios that follow were used. Each of the scenarios were derived from the software systems on which the structured event logs were implemented.

For the first scenario, the data translation system is considered. It is important to see whether the system is performing as expected for the data it processes. Therefore, the structured system event logs can be used to monitor the performance of the system. In the following scenario that will be discussed, the interface is used to view and monitor the system's throughput.

Figure 41 shows the detail event type escalation counts aggregated daily for the detail event logs for the data translation system where an audit was conducted to understand whether the system achieved the desired throughput.

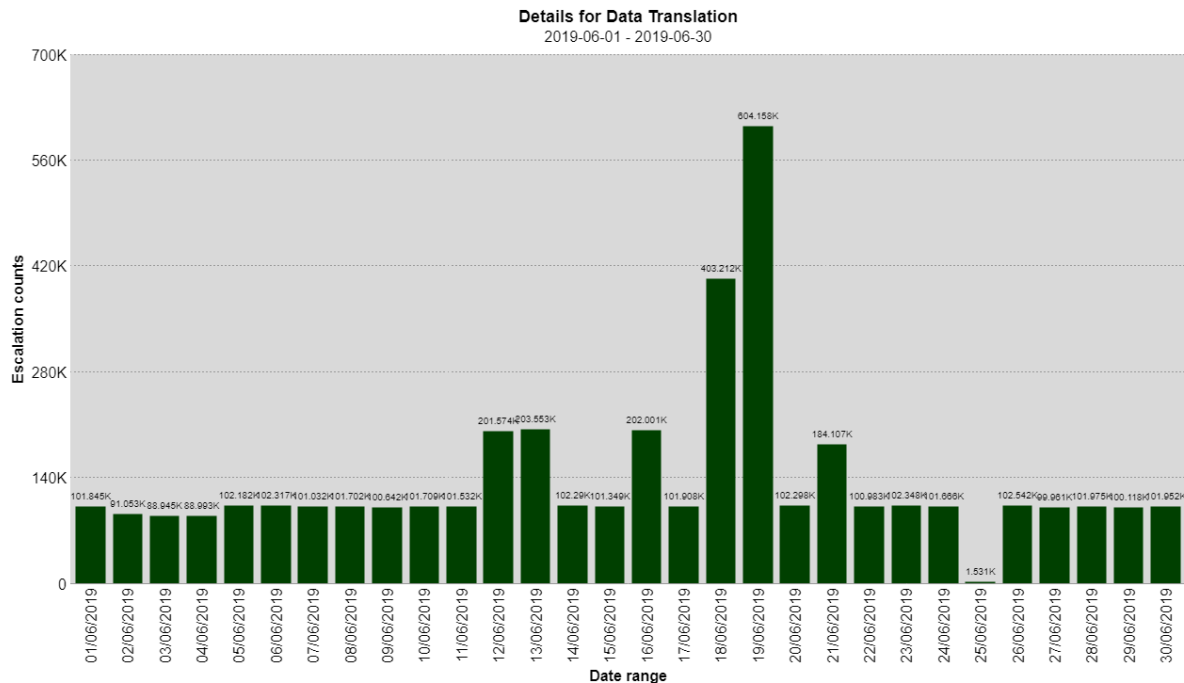


Figure 41: Data translation system throughput analysis

From the analysis graph showing the detail event type for the data translation system over a period of one month, it is observed that the system had a low throughput on 25-06-2019. Using the analysis graph and looking into the detailed event log information for that specific day, it was possible to identify why there was a decline in data processing throughput for the system. Analysing the event logs linked to this date highlighted that there were certain data which did not reach the system.

The following scenario of the data reporting system, shown in Figure 42, has the analysed event logs that were generated before and after a system update that had been deployed. The period was also set to aggregate the error escalation counts daily and displayed for a period of a month on the graph.

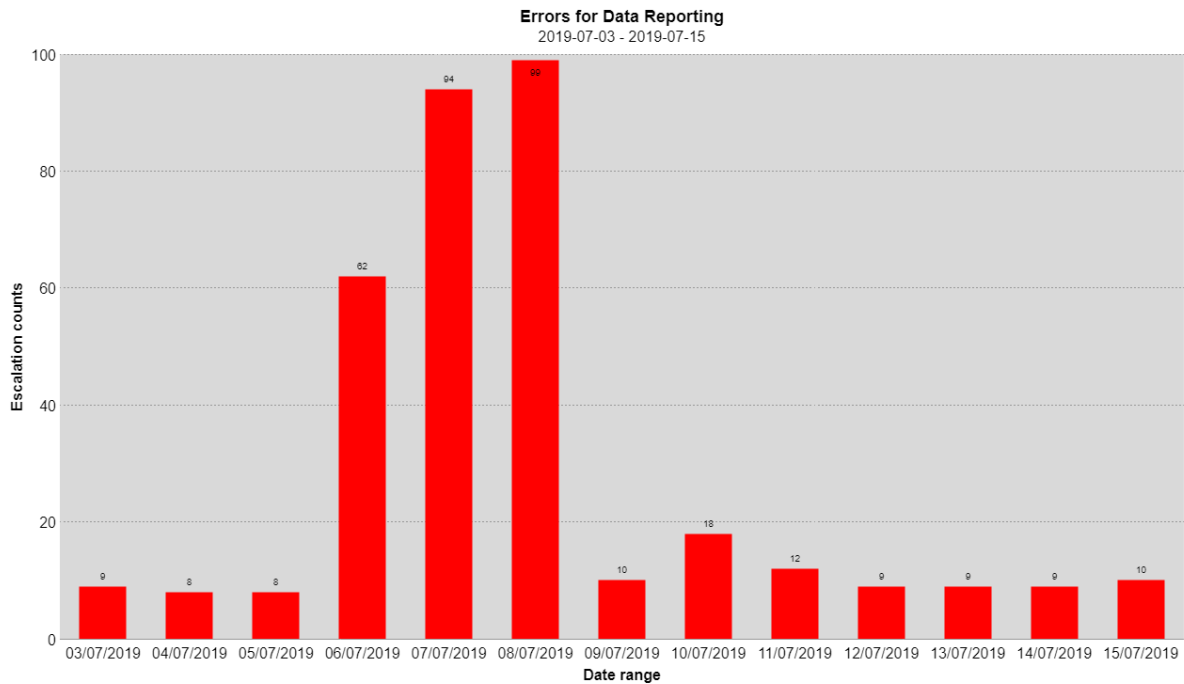


Figure 42: Reporting system errors after a deployment

The scenario shown in Figure 42 is a circumstantial use case but remains a valid example of where the analysis of the structured event logs play a role in improving overall system performance. The data reporting system processes explained in Figure 31 happens every day at 12:00 until all the pending files have been processed. On Friday afternoon of 05-07-2019 a new version of the data reporting system was deployed. When the data reporting system initiated processing the following day on 06-07-2019, more errors than usual occurred.

The software maintainers were not available during the weekend between 06-07-2019 and 07-07-2019; therefore, after the processing on 08-07-2019, which was still out of office hours, the errors also increased. Since software maintainers use the structured event logs in the interface on a daily basis, this issue was quickly identified. The software maintainers reacted on the errors on 08-07-2019 which, therefore, resulted in the errors decreasing again the following day.

Before the structured event logs were available, the software maintainers would not be able to analyse the event logs to detect errors that occur in the software systems. Since the same event log structure had been implemented for independent software systems as in the beforementioned scenarios, it validates that the event log structure is generic.

3.5 Summary

In this chapter, the method for software maintenance using structured event logs was implemented. This was done by implementing the structured event logs on three independent software systems which were considered as case studies.

An interface that was developed was used to provide software maintainers with access to the structured event logs. This enabled software maintainers to execute different maintenance tasks in each of the case studies. The case studies tasked 17 software maintainers to execute maintenance tasks using unstructured event logs in the form of flat logs and structured event logs using the interface.

The aim of the maintenance tasks was to verify that the method for software maintenance has been correctly implemented together with the integrated structured event logs to reduce the time that software maintainers spend on software maintenance. The verification was done by creating an online survey, which returned timed results from the software maintainers. The results were interpreted and an average overall time improvement across all three case studies was found to be 47%. This verified that the method for software maintenance reduces the time spent on typical maintenance tasks on the software systems.

To validate that the structured event logs in the method for software maintenance improves overall software systems' performance, scenarios of software systems where the method integrating the structured event logs was implemented were discussed. The interface using the structured event logs provided analysis of the structured event logs and actual software system issues could be identified and acted upon by software maintainers. This increased the overall software systems' performance.

Finally, to validate that the need for the study and the problem was addressed, the online survey was also used to ask software maintainers to score the experience using the unstructured event logs compared to the structured event logs. From the results it could be seen that software maintainers had access to the structured event logs at any given time. Software maintainers found the structured event logs accessible, structured, and easier to use compared to the unstructured event logs.

CHAPTER 4 CONCLUSION

4.1 Summary

In this study, literature has shown that software maintenance is the most critical phase of the SDLC and contributes to the most time, effort, and costs in software projects. The software maintenance phase is said to contribute between 60% and 80% of the total project cost due to the amount of time required in this phase.

From the literature, a problem was identified in the software maintenance phase since the time spent on software maintenance is underestimated. Therefore, a need arose to reduce the time spent on software maintenance. It has been shown that event logs can be used to obtain critical information surrounding a software system.

These event logs that currently exist on software systems were mostly unstructured and lacked quality which resulted in event logs that are meaningless. Another problem identified from the literature was that no generic method exists which implements structured event logs which can be used to reduce the time spent on software maintenance.

Therefore, a need was identified to develop a method for software maintenance which integrates structured event logs. The aim of the method for software maintenance was to reduce the time spent on software maintenance – by having event logs structured in such a way that it is available, accessible, generic and structured.

A method for software maintenance was designed which uses the event logs as a base for doing software maintenance. As part of the method design, an event log structure was derived from literature and expanded into a generic preliminary event log structure. This event log structure required an algorithm to be implemented into the method and at first, a preliminary event log implementation algorithm was designed to achieve this.

The preliminary event log structure design posed some complexities because of its nested nature which lead to a final event log structure design. This design contains all the event log structure information as the preliminary event log design but is less complex and less nested. A final implementation algorithm was designed for implementing the final event log structure.

The final event log structure was implemented on three distinct software systems which include a data extraction-, data translation-, and data reporting system. These software systems had already implemented event logs, but these event logs were unstructured in flat files. An interface was developed for software maintainers to gain access to the event logs so that the method for software maintenance can be used.

The three software systems were used as three individual case studies and 17 software maintainers were tasked to execute software maintenance tasks by means of an online survey. The software maintainers were given background on the interface, software maintenance method, and event log structure before completing the tasks in the online survey.

Verification was done on the case studies to verify that the solution was correctly implemented to reduce the time spent on software maintenance. The online survey was used for the verification, and each one of the case studies returned timed results. Case studies 1, 2 and 3 yielded time improvements of approximately 54%, 35%, and 50%, respectively.

From the case studies, an average time improvement of approximately 47% could be observed, which means that software maintainers using the method for software maintenance which implements the structured event logs reduced the time of executing typical maintenance tasks by almost half.

Finally, validation was done to validate that the developed solution addresses the problem and need for the study. The availability of the event logs was validated by using the timestamps at which the event logs were accessed. From this it could be seen that the event logs could be accessed and were available at any given time.

Software maintainers' experiences were also recorded to validate that the event logs were structured, accessible, and easy to use. Software maintainers found it easier to use the event logs in the interface than the unstructured event logs. Using the interface, all the software maintainers found the event logs in the new method structured. Using the old method, only 18% found the event logs structured.

Asking software maintainers to score their experience on whether the event logs are accessible when comparing the unstructured and structured event logs resulted in 31% and 69%, respectively.

Using the method for software maintenance to improve the overall software systems' performance was validated by using real scenarios of the systems on which the structured event logs were implemented. On the data translation system, software maintainers used the structured event logs in the interface to find a data translation throughput problem.

A circumstantial case was used in the data reporting system which pointed out system errors which occurred over a weekend which was out of typical office hours. Since this could be highlighted with the structured event logs on the system, software maintainers were able to quickly react to this issue.

Since the same event log structure was implemented on various systems and the event logs could be used to detect issues in the software systems, it also validates that the event log structure is generic.

4.2 Recommendations for further study

This study aimed to reduce the time spent on software maintenance using event logs in a method that was developed. It was verified that the time was reduced on three specific case studies on which the same event log structure was implemented. It is, therefore, expected that implementing the method on other software systems outside of this study should achieve the same results.

The generic method for software maintenance can also be improved, specifically focussing on the implementation of the event logs using the implementation algorithm. It could be seen from the implementation that more complex systems which do not start and run to completion could require another approach to implementation. Therefore, using the final event log structure, the implementation algorithm can be adapted to rather have a separate collection in MongoDB allocated to each individual system or application.

Along with this, the implementation algorithm can be adapted to create a header event log which will be allocated to each process. This will ensure that each process creates a new header event log where the detail event logs are logged against the header event log id. This means that the complete event log can be traced by its corresponding id. This can be very useful in the case where multithreaded applications are used or where different layers of the application require separate event logs.

This study also mentioned that the software maintenance phase is the largest contributor to costs and time involved in the software development life cycle. However, the primary focus was on reducing the time spent on software maintenance. Further studies could possibly use the reduction in time and correlate it with the costs that could also be reduced by using the method for software maintenance which implements structured event logs.

Furthermore, this study mostly focussed on the method for software maintenance which integrated the designed structured event logs. More studies can be done by using the large amount of data which will be generated by the software systems in the form of event logs to do analyses.

These analyses could possibly include using artificial intelligence which automatically analyses the event logs of the software systems to predict future failures by learning from previous event log trends. This can provide software maintainers with advanced notifications and possibly provide solutions when errors occur within the software systems. As the event logs grow, the more accurately the artificial intelligence engine can provide metrics surrounding the event logs.

BIBLIOGRAPHY

- [1] P. Gupta, K. Mehlawat, and D. Mahajan, "Multi-objective optimization framework for software maintenance, component evaluation and selection involving outsourcing, redundancy and customer to customer relationship," *Inf. Sci. (Ny).*, vol. 483, pp. 21–52, 2019.
- [2] T. Mens, "Evolving Software Systems," T. Mens, A. Serebrenik, and A. Cleve, Eds. Heidelberg New York Dordrecht London: Springer, 2014, pp. 65–162.
- [3] V. Lenarduzzi, A. Sillitti, and D. Taibi, "Analyzing Forty years of software maintenance models," *Proc. - 2017 IEEE/ACM 39th Int. Conf. Softw. Eng. Companion, ICSE-C 2017*, pp. 146–148, 2017.
- [4] N. Martin, M. Swennen, B. Depaire, M. Jans, A. Caris, and K. Vanhoof, "Retrieving batch organisation of work insights from event logs," *Decis. Support Syst.*, vol. 100, pp. 119–128, 2017.
- [5] Y. Djenouri, A. Belhadi, and P. Fournier-Viger, "Extracting useful knowledge from event logs: A frequent itemset mining approach," *Knowledge-Based Syst.*, vol. 139, pp. 132–148, 2018.
- [6] W. Song, X. Xia, H. A. Jacobsen, P. Zhang, and H. Hu, "Heuristic Recovery of Missing Events in Process Logs," *Proc. - 2015 IEEE Int. Conf. Web Serv. ICWS 2015*, pp. 105–112, 2015.
- [7] J. Akoka, I. Comyn-Wattiau, and N. Laoufi, "Research on Big Data – A systematic mapping study," *Comput. Stand. Interfaces*, vol. 54, no. January 2017, pp. 105–115, 2017.
- [8] L. Bonnet, A. Laurent, M. Sala, B. Laurent, and N. Sicard, "Reduce, you say: What NoSQL can do for data aggregation and BI in large repositories," *Proc. - Int. Work. Database Expert Syst. Appl. DEXA*, pp. 483–488, 2011.
- [9] M. Bala, O. Boussaid, and Z. Alimazighi, "A Fine-Grained Distribution Approach for ETL Processes in Big Data Environments," *Data and Knowledge Engineering*, vol. 111, pp. 114–136, 2017.
- [10] A. Gupta and S. Sharma, "Software Maintenance: Challenges and Issues," *Issues*, vol. 1, no. 1, pp. 23–25, 2015.
- [11] A. M. E. Hamid, "Software Change Request: A Road Map for Implementation," *2018 Int. Conf. Comput. Control. Electr. Electron. Eng. ICCCEE 2018*, pp. 1–5, 2018.
- [12] A. R. Chen, "An Empirical Study On Leveraging Logs For Debugging Production Failures," *2019 IEEE/ACM 41st Int. Conf. Softw. Eng. Companion Proc.*, pp. 126–128, 2019.
- [13] P. Marounek, "Simplified approach to effort estimation in software maintenance," *J. Syst. Integr.*, vol. 2012, no. 3, pp. 51–63, 2012.
- [14] A. . April, A. . Abran, and R. Dumke, "Software Maintenance Productivity measurement: how to Assess the readiness of your organization," *Proc. 14th Int. Work. Softw. Meas.*, pp. 231–244,

- 2004.
- [15] C. C. Venters *et al.*, "Software sustainability: Research and practice from a software architecture viewpoint," *J. Syst. Softw.*, vol. 138, pp. 174–188, 2018.
 - [16] A. Abran and J. W. Moore, "Guide to the Software Engineering Body of Knowledge," United States of America: Angela Burgess, 2004, pp. 90–98.
 - [17] S. Tarwani and A. Chug, "Agile Methodologies in Software Maintenance: A Systematic Review," *Informatica*, vol. 40, pp. 415–426, 2016.
 - [18] K. Shray, A. Shah, J. Shubham, and L. Ramanathan, "Software Maintenance : Challenges and Issues and Models for Reducing the Maintenance Cost," *Int. J. Adv. Res. Comput. Sci.*, vol. 8, no. 3, pp. 1026–1032, 2017.
 - [19] M. S. Krishnan, T. Mukhopadhyay, and C. H. Kriebel, "A decision model for software maintenance," *Inf. Syst. Res.*, vol. 15, no. 4, pp. 396–412, 2004.
 - [20] S. Dalal, "Stability evaluation model for object oriented software," *Int. J. Adv. Res. Comput. Sci.*, vol. 9, no. 2, pp. 871–876, 2018.
 - [21] ISO/IEC, "Standard for Software Engineering — Software Life Cycle Processes — Maintenance Norme," 2006.
 - [22] A. R. Gawali, "Impact of Agile Software Development Model on Software Maintainability," Doctor of Philosophy, Walden University, Minnesota, 2012.
 - [23] M.-A. Zoller, M. Baum, and M. F. Huber, "Framework for mining event correlations and time lags in large event sequences," in *2017 IEEE 15th International Conference on Industrial Informatics (INDIN)*, 2017, no. 3, pp. 805–810.
 - [24] Y. Azizi, M. Azizi, and M. Elboukhari, "Log files Analysis Using MapReduce to Improve Security," *Procedia Comput. Sci.*, vol. 148, pp. 37–44, 2019.
 - [25] W. Xu, L. Huang, A. Fox, D. Patterson, and M. I. Jordan, "Detecting large-scale system problems by mining console logs," *ICML 2010 - Proceedings, 27th Int. Conf. Mach. Learn.*, pp. 37–44, 2010.
 - [26] M. Leemans, W. M. P. Van Der Aalst, and M. G. J. Van Den Brand, "Recursion aware modeling and discovery for hierarchical software event log analysis," *25th IEEE Int. Conf. Softw. Anal. Evol. Reengineering, SANER 2018 - Proc.*, vol. 2018-March, pp. 185–196, 2018.
 - [27] A. M. Valle, E. A.P. Santos, and E. R. Loures, "Applying process mining techniques in software process appraisals," *Inf. Softw. Technol.*, vol. 87, pp. 19–31, 2017.
 - [28] A. Pecchia, M. Cinque, G. Carrozza, and D. Cotroneo, "Industry Practices and Event Logging: Assessment of a Critical Software Development Process," *Proc. - Int. Conf. Softw. Eng.*, vol. 2, pp. 169–178, 2015.

- [29] D. E. G. Vinitha and J. S. Duela, "A secured event log storage management system in cloud computing," *2014 Int. Conf. Inf. Commun. Embed. Syst. ICICES 2014*, no. 978, pp. 1–5, 2015.
- [30] T. Prakash, M. Kakkar, and K. Patel, "Geo-identification of web users through logs using ELK stack," *Proc. 2016 6th Int. Conf. - Cloud Syst. Big Data Eng. Conflu. 2016*, pp. 606–610, 2016.
- [31] S. J. Son and Y. Kwon, "Performance of ELK stack and commercial system in security log analysis," in *2017 IEEE 13th Malaysia International Conference on Communications, MICC 2017*, 2017, vol. 2017-Novem, no. 13, pp. 187–190.
- [32] M. Zanoni, F. Perin, F. A. Fontana, and G. Viscusi, "Pattern detection for conceptual schema recovery in data-intensive systems," *J. Softw. Evol. Process*, vol. 26, no. 12, pp. 1172–1192, 2014.
- [33] A. Shalan and M. Zulkernine, "Runtime prediction of failure modes from system error logs," *Proc. IEEE Int. Conf. Eng. Complex Comput. Syst. ICECCS*, pp. 232–241, 2013.
- [34] M. De Leoni, W. M. P. Van Der Aalst, and M. Dees, "A general process mining framework for correlating, predicting and clustering dynamic behavior based on event logs," *Inf. Syst.*, vol. 56, pp. 235–257, 2016.
- [35] P. Juneja, D. Kundra, and A. Sureka, "Anvaya: An Algorithm and Case-Study on Improving the Goodness of Software Process Models Generated by Mining Event-Log Data in Issue Tracking Systems," *Proc. - Int. Comput. Softw. Appl. Conf.*, vol. 1, pp. 53–62, 2016.
- [36] B. Russo, G. Succi, and W. Pedrycz, "Mining system logs to learn error predictors: a case study of a telemetry system," *Empir. Softw. Eng.*, vol. 20, no. 4, pp. 879–927, 2015.
- [37] X. Rao, G. Yin, H. Wang, D. Shi, and Y. Zhu, "Filtering error log as time series in complex service-based storage systems," *7th Int. Conf. Networked Comput. Adv. Inf. Manag.*, pp. 226–231, 2011.
- [38] D. Tovarnak and T. Pitner, "Normalization of unstructured log data into streams of structured event objects," *2019 IFIP/IEEE Symp. Integr. Netw. Serv. Manag. IM 2019*, pp. 671–676, 2019.
- [39] S. Sharma and R. Jain, "Modeling ETL process for data warehouse: An exploratory study," *Int. Conf. Adv. Comput. Commun. Technol. ACCT*, pp. 271–276, 2014.
- [40] A. Karagiannis, P. Vassiliadis, and A. Simitsis, "Scheduling strategies for efficient ETL execution," *Inf. Syst.*, vol. 38, pp. 927–945, 2013.
- [41] A. P. Kurniati, G. Kusuma, and G. Wisudaiawan, "Implementing heuristic miner for different types of event logs," *Int. J. Appl. Eng. Res.*, vol. 11, no. 8, pp. 5523–5529, 2016.
- [42] P. Ardimento, M. L. Bernardi, M. Cimitile, and F. M. Maggi, "Evaluating Coding Behavior in Software Development Processes: A Process Mining Approach," *2019 IEEE/ACM Int. Conf. Softw. Syst. Process.*, pp. 84–93, 2019.

- [43] T. Savickas and O. Vasilecas, "Business process event log use for activity sequence analysis," *2015 Open Conf. Electr. Electron. Inf. Sci. eStream 2015 - Proc.*, pp. 1–4, 2015.
- [44] S. Suriadi, R. Andrews, A. H. M. ter Hofstede, and M. T. Wynn, "Event log imperfection patterns for process mining: Towards a systematic approach to cleaning event logs," *Inf. Syst.*, vol. 64, pp. 132–150, 2017.
- [45] M. Hinkka, T. Lehto, and K. Heljanko, "Assessing Big Data SQL Frameworks for Analyzing Event Logs," *Proc. - 24th Euromicro Int. Conf. Parallel, Distrib. Network-Based Process. PDP 2016*, pp. 101–108, 2016.
- [46] J. King, J. Stallings, M. Riaz, and L. Williams, "To log, or not to log: using heuristics to identify mandatory log events – a controlled experiment," *Empir. Softw. Eng.*, vol. 22, no. 5, pp. 2684–2717, 2017.
- [47] A. Balliu, D. Olivetti, O. Babaoglu, M. Marzolla, and A. Sîrbu, "A Big Data analyzer for large trace logs," *Computing*, vol. 98, no. 12, pp. 1225–1249, 2016.
- [48] S. Chickerur, A. Goudar, and A. Kinnerkar, "Comparison of Relational Database with Document-Oriented Database (MongoDB) for Big Data Applications," *Proc. - 8th Int. Conf. Adv. Softw. Eng. Its Appl. ASEA 2015*, pp. 41–47, 2016.
- [49] R. Lawrence, "Integration and virtualization of relational SQL and NoSQL systems including MySQL and MongoDB," *Proc. - 2014 Int. Conf. Comput. Sci. Comput. Intell. CSCI 2014*, vol. 1, pp. 285–290, 2014.
- [50] M. Cinque, D. Cotroneo, and A. Pecchia, "Event logs for the analysis of software failures: A rule-based approach," *IEEE Trans. Softw. Eng.*, vol. 39, no. 6, pp. 806–821, 2013.
- [51] Y. Otsuki, Y. Kawakoya, M. Iwamura, J. Miyoshi, and K. Ohkubo, "Building stack traces from memory dump of Windows x64," *Digit. Investig.*, vol. 24, pp. S101–S110, 2018.