User-based activity logging and analysis to improve system maintenance

C Scheepers

orcid.org/ 0000-0001-6089-7110

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

Supervisor: Dr. P. Goosen

Examination: Nov 2022

Student number: 25899880

Abstract

Title:	User-based activity logging and analysis to improve system maintenance
Author:	Mr Cornelius Scheepers
Supervisor:	Dr Jaco Prinsloo
Degree:	Master of Engineering in Computer and Electronic Engineering
Keywords:	Software maintenance, logging mechanism, user activities, system utilisation, Web-based

Maintenance of software is continuous and is a reduced form of software development. Research suggests that allocating 15% of the total development cost is to implement a maintenance model on a specific software system. Unused software components or software not meeting the user's requirements will increase over the project's life cycle. Deprecating, some of these systems may reduce the number of resources needed to maintain the entire project. Deciding how many resources for maintenance needs to be allocated to each system can be difficult without a suitable method.

Software logging is an essential mechanism to improve software maintenance in the form of troubleshooting. In large software systems, logging enables the development team to monitor specific events. System utilisation of each software system can be identified if a suitable logging mechanism is implemented. In Web-based applications, each software system's utilisation is based on how much the users will interact with it. The interactions between the user the software system can be logged.

Analysing these logs can be challenging when the logging mechanism does not track the desired user-based events. Developing a method to track these events for a specific purpose is more efficient. Relevant data creates a more effective analysis of a specific topic and increases the optimisation of a model. The need to integrate the method used to create a logging mechanism and analyse the data will improve software systems' maintenance.

During this study, a method is developed for a Web-based application to track the user's activities. The logging mechanism is designed to get any user activities on any software systems they are interacting with. Ajax-requests contains significant and relevant data than tracking all the actions of the user of a specific event. Capturing each Ajax-requests generated from the user activities is the main communication interface between the user's client device and the server that runs the software systems.

The analysis is performed on the logs that were obtained from the user activity logging mechanism. Software systems utilisation is compared to each other. The resources allocated to maintain each software system is compared to each other. Using the resource allocated and the system utilisation, new decisions can be made for the software maintenance. This formed part of the data-driven decision making about the utilisation of the different software components.

Acknowledgements

Thanks...

Table of contents

List of figures

List of tables

Nomenclature

Abbreviations:

IEEE	Institute of Electrical and Electronics Engineers
SDLC	Software Development Life Cycle
LCP	Life-Cycle Phases
OSS	Open-source software
№	Ordinal numeration
AJAX	Asynchronous JavaScript and XML
I/O	Input/Output
CPU	Central Processing Unit
PHP	Hypertext Preprocessor
ERD	Entity Relationship Diagram
MVC	Model-View-Controller
F/R	Functional Requirements
VARCHAR	Variable Character
INT	Integer
ENUM	Enumeration
JSON	JavaScript Object Notation
HTML	HyperText Markup Language
URI	Uniform Resource Identifier
URL	Uniform Resource Locator

Units:

kVAr	Kilovolt-ampere	Reactive power

Chapter 1

Introduction

1.1 Background

In the modern era, most types of businesses will make use of digital products and services to maximise their profits [1]. This increases the need to create new innovative software systems to cater to the specific needs of the users. Software projects may vary in size, type and degree of difficulty of implementation therefore, the software quality is crucial [2]. There are certain defined attributes and characteristics the software systems need to adhere to, that is called software quality [2].

Using a System Development Life Cycle (SDLC) methodology, such as the agile software development methodology, is crucial in making the software development process efficient and predictable. This enforces various degrees of disciplines to the software development process to ensure that the software quality is acceptable for the user requirements [2, 3]. Table 1.1 lists the Life-Cycle Phases (LCP) of the SDLCs.

Table 1.1: *System Development Life Cycle phases [2, 4]*

Life-Cycle phase	**Description**
Initiation	The sponsor identifies a need or an opportunity and a concept proposal is created for the new opportunity. This new opportunity will need a software solution to make the project successful for all the stakeholders involved.
System Concept Development Phase	In this phase the feasibility and appropriateness of the concept proposal are reviewed when it needs to be approved. The Systems Boundary Document identifies the scope and needs additional approval before the planning phases are implemented.
Planning	In the planning phase the project management plan and other planning documents are created. These documents define the available budget, project resources, activities, schedules, tools and reviews.
Requirements Analysis	In this phase the user requirements are defined and analysed. The functional requirements are created with the functional requirement document. Non-functional requirements are also defined to ensure that the software system operations will not deviate to work within the non-functional specifications. These non-functional requirements can be negative and costly to the software system's operations and potentially reduce its usability or life cycle.

Continued on next page

Table 1.1 – continued from previous page

Life-Cycle phase	Description
Design	In this stage the detailed requirements are completed in the System Design Document that describes the detailed logic specifications of the software system.
Development	In this phase the design specifications are converted into an executable software system. This also includes acquiring other third-party software or internal software to install them on certain software environments, creating and testing databases, performing test readiness reviews and other software development activities.
Integration and Test	In this phase the software systems are integrated and systematically tested to examine if it meets all the functional requirements and other accreditation activities for test approval and user approval using user tests.
Implementation	This phase is initiated after the software systems passed the testing phase and are accepted by the users. This phase contains the implementation of the software system in a production environment and the deployment of the production version of the software. This phase continues until the software systems are operational in a final production environment.
Operations and Maintenance	In this phase continuous monitoring of the performance of the software system is done in line with the defined user requirements. Any additional modifications after the initial software development are done here and any other tasks are needed to maintain the software system. The Post-Implementation and In-Process Reviews for the software system form part of the documentation for this LCP.

Continued on next page

Table 1.1 – continued from previous page

Life-Cycle phase	Description
Dispostition	This phase is any disposition activities to properly terminate the software system. Any important data that is needed for the reactivation of the software system in the future is also preserved. Any other termination policies and activities should be defined and documented. This ensures that the termination of the software system is done correctly and completed and that the correct data is permanently removed or preserved.

There exist various adoptions of the LCP listed in Table 1.1 for various SDLC methodologies that are used in the software development industry [3]. The planning phases of the SDLC should be well documented and the software architecture should be well structured and defined. By doing this, it makes it easier for the development and maintenance-related LCPs to be implemented [5].

Each LCP ensures that the software development is correctly implemented if it is part of the adopted SDLC methodology. The SDLC methodology is implemented for the entire life cycle of the software system. There will be alterations to the SDLC methodology to keep up with any new design and development requirements.

In the actual implementation of the SDLC methodology, some design patterns could stray away from the initial software designs [6]. This can be due to unforeseen issues occurring during the Development phase due to some time or other resource constraints. The software system's initial design might also not be flexible enough for any newer modifications that can also occur in the Operational and Maintenance phase.

In both situations, software development could deliver workarounds or shortcuts that resolve the identified problems. These short-term benefits will not always translate to good long-term software quality due to prioritising functionality above good software design patterns. This decreased quality of the software code is caused by technical debt if the software system was not implemented with suitable SDLC practices [6, 7].

Technical debt can be defined as the technical compromises that software engineers and developers will introduce to a software system for short-term goals that may increase the complexity and sustainability of the system in the long term [1, 8]. With the need to always deliver software systems for more innovative, complex and larger software systems the risk of introducing technical debt increases as well [2, 6].

The Operations and Maintenance phase of the software system is where it is usually resolved or reduced to some extent. More technical debt will increase the maintenance efforts that need to be implemented to offset the negative impact that the short-term goals can potentially create. Figure 1.1 is a representation of the technical debt repayment for the software system during its entire life cycle.

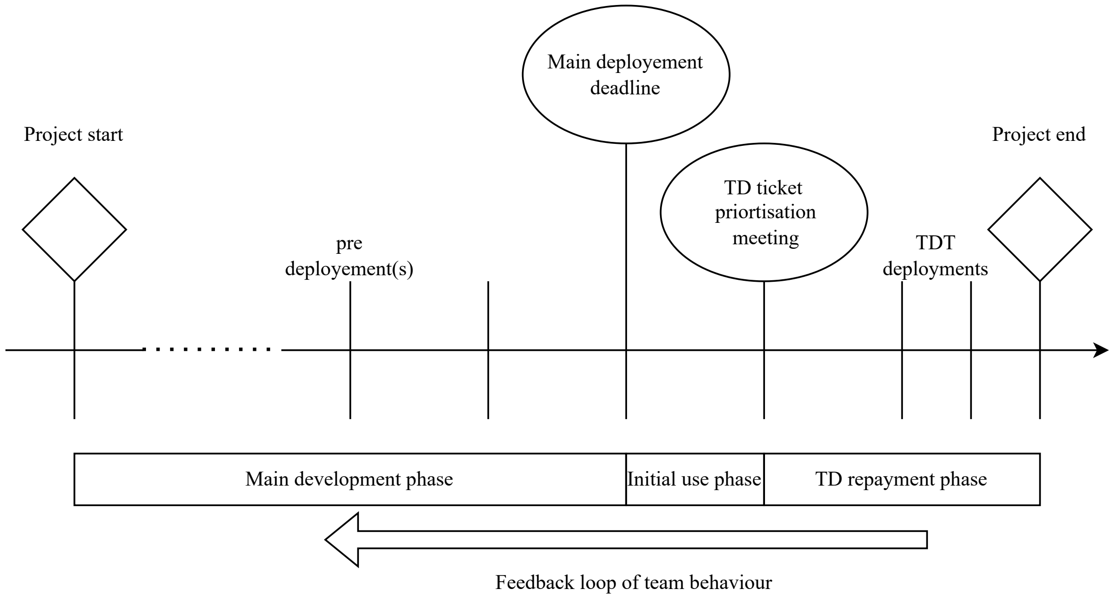

Figure 1.1: *Project plan with included technical debt repayment phase [9]*

Technical debt will always be present to some degree in software systems throughout their entire life cycle [9]. In Figure 1.1 technical debt repayment is the software development activities that aim to resolve the technical debt issues that are identified after the initial deployment of the software system.

Tickets are assigned to the identified issues that are caused by technical debt when the functional requirements are reviewed in the initial use phase. The technical debt tickets are then prioritised based on their importance and utilisation. This ensures that the Operations and Maintenance phase development efforts are efficiently implemented for user satisfaction and the software systems' sustainability.

According to the United States Department of Commerce, the software maintenance efforts of the Operations and Maintenance LCP of the SDLC in Table 1.1 will contribute to about 60%-80% of the total development cost for the software system's entire life cycle [5, 10, 11]. Therefore following adequate software maintenance practices is needed to avoid [7]:

- additional software and hardware resources needed that can be costly,
- software quality issues,
- make any new modifications impossible without negatively impacting existing software features or systems,
- shortening the useability of the software system, which can lead to earlier termination of the software system.

Software maintenance of the Operations and Maintenance LCP phase is an essential task in software development. It can directly reduce the cost and effort to create new software systems or modify them in the future and reduce technical debt [7, 12].

1.2 State of the art

1.2.1 Software maintenance

Maintenance of software systems is continuous and is a reduced form of software development and is aimed at modifying software systems while preserving their integrity for current and future operations [5, 13, 14]. Software maintenance aims to improve software's:

- **correctness** of the software systems will always have some defects or faults that need to be corrected to improve the software system's traceability, consistency and completeness,
- **enhancements** of the software system to improve existing software components to adapt to changes to the user's requirements and to improve system performance and sustainability.

Improving the correctness of the software system and enhancing it will need to follow a defined maintenance process to implement it. According to the *IEEE Standard 1219*[1] software maintenance includes the following phases [15, 16]:

- Identifying the problem or modification and classification of it
- Analysis of the identification of the maintenance issue
- Design of the solution to implement maintenance
- Implementation of the solution
- System testing of the modified software system
- Acceptance test on the fully integrated system
- Delivery requirements met of the modified software system

These software maintenance phases cannot be omitted when the software system is still active. To ensure that the software system can keep up with defined user requirements and new user requirements in the future software maintenance needs to be implemented. This will increase the maintenance that needs to be done on both new and old systems [16–18]. In Figure 1.2 is the representation of the total resource cost of implementing software maintenance.

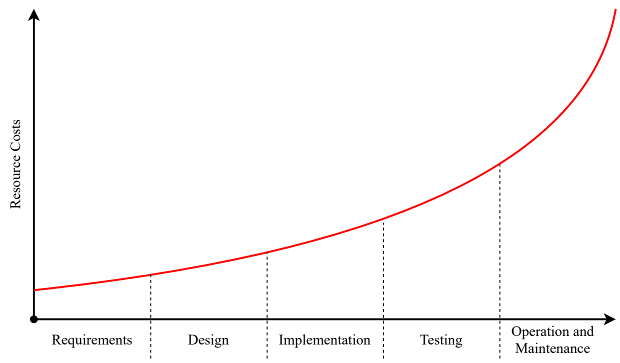

Figure 1.2: *Resource cost of software maintenance [10]*

[1] **IEEE Standards** documents are developed within the Technical Committees of the IEEE Societies, and the Standards Coordinating Committees of the IEEE Standards Board [15].

In Figure 1.2 the total resource cost will significantly increase as the need for software maintenance increases. As previously stated that software maintenance can use as much as 60% − 80% of total resources, and it can be expected that most of the resource costs will be for the Operation and Maintenance LCP. These software resources that cost both money and time can be:

- software developers and other support staff involved in the software maintenance process,
- software development tools and services such as testing software environments, analysis tools, online surveys and software fault reporting systems etc.

Software maintenance types

Maintenance problems or modifications are regularly identified and are usually addressed based on an initial priority ranking. This priority ranking is determined by using classification models to determine which type of maintenance needs to be done as described in Table 1.2 for the Operations and Maintenance LCP of Table 1.1 [11, 19].

Table 1.2: *Software maintenance types [16, 19]*

Maintenance type	**Description**	**% of maintenance activities**
Adaptive	Adaptive maintenance in software systems is any modification or enhancements to keep it usable with a changing or changed software environment.	≈ 37.5%
Perfective	Perfective maintenance based are modifications made based on the change of the end-users' new requirements. It can also improve the performance or maintainability of the software system in its life cycle.	≈ 37.5%
Corrective	Corrective maintenance are improvements made to fix certain defects or errors in a software system.	≈ 20%
Preventive	Preventive maintenance are improvements made to software systems that prevent problems in the future.	≈ 5%

The maintenance types of Table 1.2 In Table 1.2 the adaptive and perfective maintenance types are about 75% of the total maintenance software development for the Operations and Maintenance LCP. These two types of maintenance are aimed to resolve technical debt issues that may appear after the initial software deployment and will mostly be defined by the technical debt tickets described in Figure 1.1. Adaptive and perfective maintenance ensures the software system will continue to evolve and improve to meet the system requirements to ensure that it is usable and feasible [20].

There will always exist software faults or defects that will need to be fixed and deployed. These maintenance software changes are usually smaller and are aimed to increase the correctness of the software system. This can also be any preventative measures to avoid technical debt issues in the future with preventive maintenance efforts.

For software defects and faults t can be a demanding task to prioritise the available resources for certain parts of the software system to do maintenance to prevent or fix any software defects [15, 16]. The defect density of a software system is the number of possible defects divided by the size of the software system as in Equation (1.1):

$$Defect\ Density = \frac{CNDD}{KLoC}, \tag{1.1}$$

where:

- $CNDD$ is the cumulative number of defects in the post-release version of the software system
- $KLoC$ (thousands of lines of code) is the size of the observed executable code in a software system

A lower defect density indicates that the quality of the software is good [21, 22]. This does not indicate that the software did indeed fulfil the user's requirements but that there may be fewer possible faults present.

In open-source software systems (OSS), the defect density increases due to the number of developers working on the same software system and the size of the system itself [23]. Adding more developers to improve a software system may not always have an improvement in each of the maintenance types in Table 1.2.

This increases the need for corrective maintenance efforts as more developers are already trying to resolve other existing maintenance issues. Larger and more complex software systems will have a higher defect density due to the possibility of more software defects increasing [24].

Problems with implementing software maintenance

Under most circumstances, maintenance is implemented if a software system does not meet the required functions specified by the user or performance requirements [10, 13]. Maintenance can be difficult to implement due to:

- **Problem domain being complex:** The software may not be well defined or structured during the planning LCP phases of Table 1.1. This is due to how large the software systems grow over their entire life cycle or duplicate software components that are made.

  A poor understanding of the system architecture or insufficient documentation about the software system exists when analysing the maintenance issues [18]. Software engineers and developers tend to not create or update documentation as it is a time-consuming task when software needs to be delivered on schedule.

- **Difficulties of managing development process:** Most companies will strive to increase their digital products and services over the life cycle of the software project to maximise possible profits with the resources invested [17]. Increasing production of the development process will only strain the maintenance efforts of the software systems [13].

Software engineers and developers already have a busy schedule to deliver software features on time [18, 25]. They will quickly feel overwhelmed and suffer from development burnout if the development process is not correctly managed to include additional software development by implementing maintenance.

- **Flexibility of the software:** Trying to predict what the possible future architecture may look like and modifying it while preserving the software's integrity may be difficult in software maintenance [26]. Software is flexible if it is adaptable to the problem domain when adding modifications to it [10].

  Most development teams will follow a software development methodology to create a future architecture that is modular and structured to preserve the development integrity of new software [27]. This will also have an impact on the type of maintenance activity (as in Table 1.2) the development team will need to implement [8, 12].

- **Change in user's requirements:** In software development, the users will often request new additional requirements to the software systems delivered to them [10]. Modifying software systems may include new and additional features that change the initial system architecture. Maintenance of these systems is crucial to ensure that existing components of the system will work as intended with the new components that are added.

- **Environmental changes:** Rapid changes in software development are always present with the need for more innovative solutions added to solve new complex problems. These changes are not always compatible with existing software systems even if the initial software architecture is well-defined [10].

  There will always be a need to make improvements to the software system in the form of third-party software updates and services used. The software system needs to be modified to accommodate these new changes as it is beneficial for its sustainability and operation.

- **Bad design:** Maintenance can be difficult if the system is badly designed when the SDLC was implemented. The maintainability of the system is impossible or too difficult without making major changes. The complexity of the software system may be too high for some developers which will cause the mentioned problem domain issue with complexity in software systems [25].

Software maintenance prioritisation

Due to time constraints and available resources, it is difficult to plan any maintenance operations for most software engineers and developers [7]. The increased resource cost of software maintenance is due to a lack of planning or preventive measures to keep the software system from degrading [22].

Continuous analysis of the identified maintenance problems or modifications enables the software engineers and developers to create a preliminary plan to address these issues in an efficient process [14]. Implementing a suitable maintenance framework to resolve the identified issues reduces technical debt.

Various maintenance models can be used to solve these issues when implementing any one of the maintenance types. A software maintenance model is an abstract representation of software systems' evolution to keep track of all the maintenance activities when implementing software maintenance [28]. The *IEEE Standard 1219* for software maintenance is the standard that should be followed when planning software maintenance as in Figure 1.3.

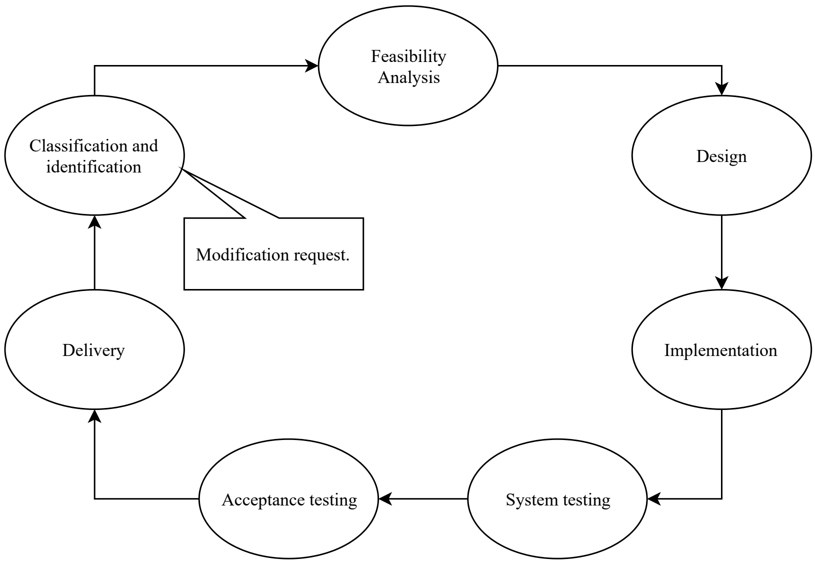

Figure 1.3: *IEEE Standard 1219 model for software maintenance [28]*

The software maintenance model in Figure 1.3 emphasises that the software defects or faults need to be identified and classified for the maintenance process. A feasibility analysis of any changes that are required should be made if the maintenance effort is worth implementing. A certain amount of resources will be allocated to implement maintenance, and this will impact the design and implementation phase.

For the feasibility analysis, making use of a system characterisation report may help identify possible maintenance focus points in a software system [29]. This will increase the effectiveness of designing solutions to implement maintenance as a system assessment focus can be made. The metrics can be defined as:

- positively or negatively impacts the performance of the system,
- fulfil the defined user's requirements,
- increase user engagement with the defect and fault fixes or expand existing software components with additional features,

- is worth for a large portion of the userbase to implement.

User engagement with the software system is what will determine if the software system is sustainable to generate revenue for the organisation. It may be the most important focus metric for the feasibility analysis to determine if a maintenance effort is worth implementing [29]. A maintenance flow model can resolve this issue such as the one in Figure 1.4.

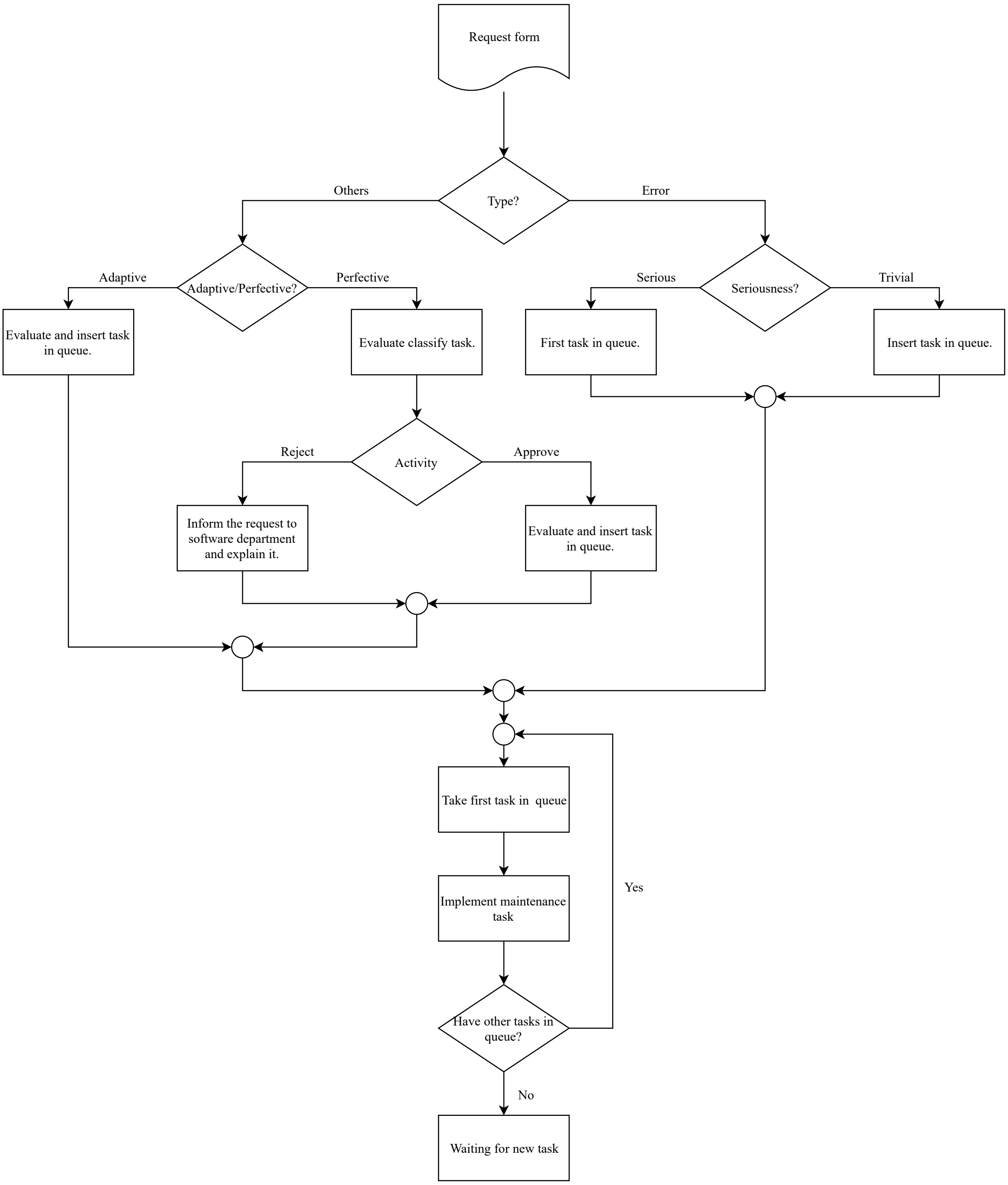

Figure 1.4: *Maintenance flow model [11]*

Figure 1.4 is an example of what a practical maintenance flow model is an organisation would likely use to implement software maintenance. Initially, a developer will complete a request form or issue indicating a new problem or feature request that needs to be implemented [11].

After all the maintenance tasks are defined, the development team will prioritise the higher-rated issues that need to be solved. This process will repeat itself until all the task for that specific software system is completed.

To fully follow the *IEEE Standard 1219* of implementing maintenance on a software system, the defects or areas of improvement should be identified. Utilisation analysis of event logs can be used to detect any hidden defects or performance issues in a software system to implement software maintenance [30–32].

System and acceptance testing are essential to ensure that the system is still fully functional and fulfils the user's requirements. After the system is thoroughly tested and approved, it will be available to the user, and the maintenance process will start again when there are new improvements to be made to the software system.

In Section 1.2.1 it was identified that software maintenance is essential to be able to fulfil the user's requirements. For any maintenance model, as described in Section 1.2.1, to be effective the software maintenance model will need to be able to prioritise the maintenance issues efficiently.

Most organisations' maintenance models will be based on Figure 1.3 to manage their maintenance efforts for the Operation and Maintenance LCP. Up to 50% of a software engineer or developer's total time invested in implementing maintenance is to understand what the software system is supposed to do [11]. This is due to the problems with implementing maintenance as discussed in Section 1.2.1.

If the issue is a problem in the software system, the severity of the problem needs to assess to decide on the priority level to resolve it. This type of maintenance is mostly corrective and can also be preventive if it is a possible solution to prevent any software failures in the future [11]. Other types of maintenance requests are either adaptive or perfective and usually placed in the development team's task queue.

Prioritising maintenance for the user's requirement to extend the useability and increase user satisfaction is preferable to maximise profits. In most cases when there is no suitable software maintenance model used, the software maintenance is reactive [29]. However, this is not a sustainable maintenance policy for larger and more complex software systems.

In order to prioritise maintenance more efficiently a system characterisation of the software system needs to be made. In Section 1.2.1 user engagement with the software system has been identified to be an important metric when implementing maintenance.

Knowing what the user uses or how they interact with the software system provides valuable data to the development team. But to access that data some form of tracking is needed to obtain the data automatically.

1.2.2 Event logging

As described in Section 1.2.1 a tracking method is needed to capture data about user engagement with the software system for maintenance purposes. It is a common practice in the software industry to record any detailed system run-time information into event logs. These event logs can be analysed later by developers or software engineers to solve software-related problems [33].

Event logging is a proven implementation to get information about the behaviour of software systems [34]. Event logs are software system-generated textual files that collect data on reported events of interest that occur during various operations of the software system. [30, 34].

The technique to collect numeric or textual data that describes the behaviour of a computer system is called event logging [34, 35]. Event logs collect textual data containing the records of events that happened in a software system and are used for system management tasks as in Table 1.3 [31, 34, 36]. Event logging has three major purposes [34, 35]:

- **state dump** which is reporting of values of certain variables or data structures inside the software system.
- **execution tracing** which is the reporting of certain states of the software system or what is currently happening in the software system,
- **event reporting** which focuses on any desired events in the software system that has textual information of that event.

Event logs are mainly used for event reporting to support debug and system integration activities to reduce the amount of code that needs to be inspected [34]. Table 1.3 is the most common use of event logging in the industry is described.

Table 1.3: *Event logs usage*

Usage	**Description**
Debugging of software systems and services	Event logging is mostly used to record events or behaviours of software systems or services during its run-time [31].
Anomaly detection	Event logs can be used to detect any abnormal system behaviour using an anomalous detection algorithm using logging data [37]. This can also be used to find any potential vulnerabilities or defect prediction in the software environment [38].

Continued on next page

Table 1.3 – continued from previous page

Usage	Description
Performance diagnosis	Software performance is important to producing quality software for the end user [34, 39]. This is also important to make informed decisions on how to improve the software system or service for improved performance and other resource and financial implications. This type of performance event logs of the software systems or services are used to monitor the software system, which is useful for resource tuning, load balancing and checking system scalability in the entire life cycle of the software system or service [40].
Auditing	In a software environment there can be significant changes to the database data that might need to be log for auditing purposes [31]. All establishments and enterprises need to ensure that compliance with the industry regulations is met with their software systems by adding audit logs. They are also legally bound to have audit logs to provide legal evidence for any legal investigation or administrative tasks to ensure that accountability is maintained.
Error and failure analysis	Event logs are used to analyse the failure behaviours of software systems which enable software engineers or developers to understand the failure modes of the system, find the root cause of these failures, prevent them and improve the reliability of the future releases of the system [30].
Analysis of security alerts	In any software environment or information technology infrastructure, security is a major concern for any organisation [38, 41]. It is important to know the overall security status of the software system.

Logging practice in software development Logging practice in software development is not always well documented and there can be multiple implementations of different logging mechanisms in the same software system [35, 42]. In modern software systems, the logging practice is a crucial part of the development of software and the maintenance of it in its entire life cycle [36].

In Appendix A there has been a rise of new studies that focus on providing suitable logging practice guidelines to software engineers and developers. Logging in the industry makes use of many third-party logging libraries and frameworks such as Apache's log4net and Microsoft's

ULS frameworks [36, 43].

Software engineers and developers can use these tools to implement logging in a compatible software system. They will still need to know how to strategically place the logging points so that the desired logs can be obtained. Using guides provided by the tools and other online guides, the logging practice can be implemented. When using a third-party logging mechanism the software engineers and developers will, in most cases, need to:

- add the logging points in the software system at locations where it can capture the desired logs,
- enable the log parsing stage to write a log entry into a database.

Logging guides can give examples and suggestions on where to place the logging points but that can still be difficult on occasion for software engineers and developers to identify these desired locations. This can be difficult due to logging guides being mostly application-specific or the logging mechanism can only capture certain event types to create event logs.

For more custom logging, the software engineers and developers will need to create new logging mechanisms. This adds new requirements for the logging mechanism to be functional. For a logging practice to be successful, two important problems need to be resolved [33, 36, 43]:

- **What needs to be logged?**

  In Table 1.3 the use of logging is diverse and will have an impact on how the logging mechanism will be designed.

- **Where to log?**

  Different types of logs will only be present in the software system at certain locations during run-time. Knowing what to log narrows down which locations can be used to obtain certain events while they are present.

To answer these two questions the event log has to fulfil certain log quality requirements. This ensures that the created logs are correct and consistent when it is extracted and viewed.

Logging quality

Software engineers and developers will need to make informed logging decisions for event logging. These decisions may affect the software system's run-time operations negatively and can impact the event logging efficiency [33, 43, 44]. An event log quality model in Figure 1.5 can be used define to ensure that the event logs will have consistent integrity when capturing the event log data.

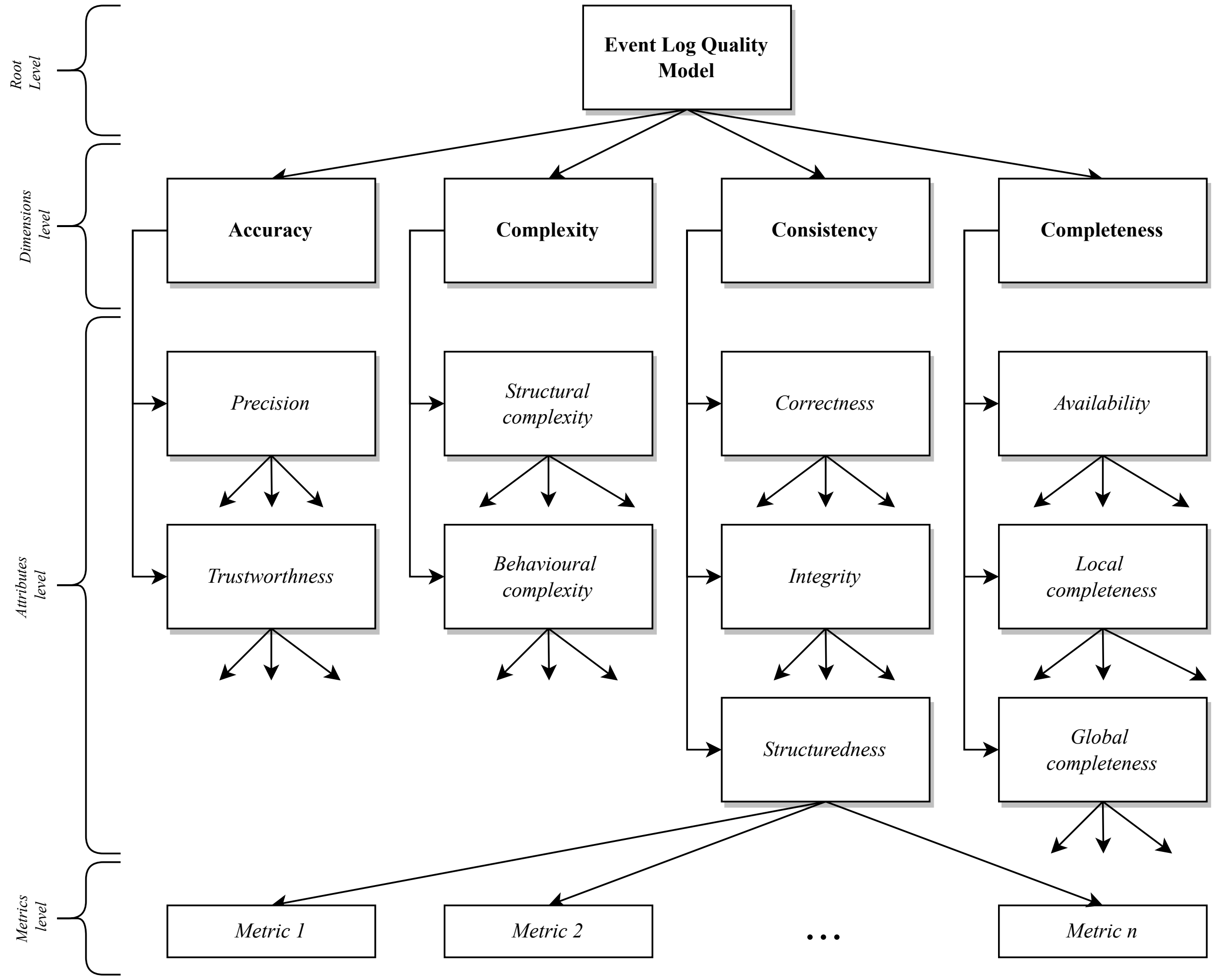

Figure 1.5: *The event log quality model [44]*

The event log quality model in Figure 1.5 consists of four different levels to define each property of the event log quality model:

- Root level for the event log quality model. These are the main requirements for the event log quality model,
- Dimension level which consists of four main dimensions of the event log quality model according to [44] (complexity, accuracy, consistency and completeness of the event log),
- Attributes level where a set of quality attributes for the dimension level,
- Measurement level which should be defined in the design process of the logging mechanism to achieve the attributes and dimension levels.

Accuracy of event logging

It can become difficult to log every event in a software system as these systems will get larger and more complex during their development life cycle [45]. Precisely capturing certain events that need to be logged needs to be consistent to ensure that the data is trustworthy and reliable to use. The log attributes of the event log also need to be precisely captured for each event log and should be correct.

Capturing more event logs doesn't guarantee that the accuracy of the event log is on an acceptable level as the log's attributes can be incorrect or cause duplicated events logs if the data is the same in a sequence of event logs. In Table 1.4 are the common problems associated with too much logging.

Table 1.4: *Problems with too much logging [43]*

Problem	Description
Excess code	Adding multiple logging points may increase the amount of code added to the software system to capture the event logs. The code may take some time to write and maintain. This can increase the structural and behavioural complexity of the code in its entire life cycle.
System resource impact	With the additional code needed to capture the logs the system resource usage such as the CPU and I/O channels will increase. This may negatively impact the performance of existing system operations or increase the cost to keep the system at the same operation speed by increasing the system resources.
Unusable logs	Adding numerous logging points or logging too much at points can produce numerous trivial or useless logs that will not improve the system utilisation analysis. When implementing the system utilisation analysis stage the logs might need to get filtered more or modified to be more meaningful as much as 70% of the logs may be irrelevant [46]. The logs are written by the software engineers and developers and can sometimes be irrelevant to other managers or system administrators when implementing a log analysis report of the event logs. More event logs can have missing or incomplete logging attributes due to the excessive logging points that have been added. The increase in the behavioural complexity of the log can impact the decisions that are made to improve software maintenance.

The accuracy and trustworthiness of the event log are more important than capturing a large number of available event logs in a software system [43, 47]. The extra unnecessary logs will also take up more storage space store which will increase costs and possibly the performance of the software system.

Event log complexities

Software always has some complexity involved and it will always increase when the software system becomes larger. For the event log quality model the complexity of the event can be split into two different complexities which are [44]:

- Structural complexity is the application of different algorithms in the software which allows the event log to be evaluated when it occurs which can alter the behaviour of the event log.
- Behavioural complexity in event logging is the complexity of the behaviour of the event logs that refers to the number of smaller events in each captured trace and the different variations of these traces within an event log.

These two complexity attributes can be costly when the event logging mechanism needs to be constantly maintained in large software systems where it can impact the rest of the system's performance or integrity of the captured event logs [10]. The constant modification of the event log software can be due to technical debt as the event log system's complexities lead to technical issues when attempting to log an event or is not compatible with other systems [7].

Consistency of event logging
The event logs' accuracy and consistency are critical when making reliable decisions based on the identified behaviour of the software system with the historical data that exists in the event logs that is discussed in the previous event log quality dimension [44, 45]. With the accuracy and trustworthiness of the event logs correctly applied, the consistency of the event logs should be on an acceptable level to be correct and verifiable when comparing it to the software system.

An event log quality model is essential to ensure that the logs will be of high quality for the data mining process of log analysis and therefore the event log data should be consistent. To ensure the consistency of the event logs the structure of the event logging points and log parsers should be consistent to capture all the important log attributes. The event log data should also be consistently analysed with different methods used on it as part of the consistency of the event logging process.

Completeness of event logging
The event logs will be analysed at a later stage, the logs should be fully complete when it is being used as some of the other logging attributes might not be available at that stage. The event logs' available attributes should be accurately captured before attempting to store them in a database to ensure that there is no missing event data or missing events if the event is discarded due to it being incomplete. There are two types of completeness attributes excluding the availability attribute in the Figure 1.5:

- Local completeness refers to all event data that can be captured for an instance of the event taking place that can be added as a log attribute to the event log [44, 48].
- Global completeness refers to the occurrence of all possible outcomes or behaviours of the event logs that can be captured which is required for the system utilisation analysis [44, 48].

Ensuring that both completeness levels are achieved and that the event log data is complete can impact performance if certain data is not directly available during the instance when the event has occurred. The logging mechanism needs to capture this as efficiently as possible without causing performance issues to the rest of the software system's operations [33, 43].

Logging parsing and log points

Knowing what to log can significantly reduce any overhead the logging mechanism may produce in the software system [35, 49]. Preserving quality (as described in Table 1.3) is a necessity to ensure that the logs that are obtained will fulfil their purpose when analysing it.

Logging attributes

Before the logs can be parsed to a structured data set the key attributes that need to be defined that will describe the event log [50]. The attributes in describing in Table 1.5 are the most basic attributes a log event should have.

Table 1.5: *Basic log event attributes [50]*

Attribute	Description
Case number	Unique identifiers for each log event. This is usually the primary identifier for the log event.
Timestamp	The time and date that the log event occurred. This is part of identifying the order of events or traces along with the case number of the event log [44].
Event type	Each log event can be grouped with other log events that have similar actions that happened. These event-type attribute needs to be classified based on a state change, failure to execute an instruction, or due to an occurrence of activity like the availability of service [46]. The event type is usually also the log level. The log level in event logging reflects the severity of the event log [51]. An event of interest may have different log levels and this makes it easier to capture certain events in which software engineers and developers are interested.
Originator	The origin from which the event took place in the software system. This can be parts of the software that performs the event action or was the cause for the event to be initiated by another part of the software.
Other metadata	This is any other relevant information that can be used as the event log's attribute that further expands the information of the log event. This can be one extra field or many other individual attribute fields.

These attributes make it possible to mine and analyse the logs based on their attributes and increase the precision and reliability of the event log [44]. The case number and timestamp attributes in Table 1.5 can be defined anytime during the logging process. This is not the case for the rest of the attributes.

Every log should have a defined action that will put it in a group of logs that can be defined as the event type. These event types can be either predefined of what is expected from the event action or will need to be observed later analysis of the logs in case there is no clear grouping of the logs [46, 50].

Log points

The sources of the log event assist on determine the location where the event took place. For event logs this essential to try to recreate scenarios or actions based on the relevant parts of the software system that participated in the event action.

Other metadata can increase the log quality by providing additional information about software instructions that were executed. These attributes add more information that can be used to recreate the scenario or action that may be unique parameters or other events that participated in the event log.

To get the attributes in Table 1.5 an instruction generates the log and parses it onto a data set. These log instructions are called logging points in the software environment [35, 43]. They can be any instruction such as a print function that displays the information for the user to more complex functions or libraries that can be created by third-party developers. In Figure 1.6 is an example of a logging point parsing a log message in a structured log.

```
/* A logging code snippet extracted from:
    hadoop/hfs/server/datanode/BlockReceiver.java */

LOG.info("Received block " + block + " of size "
     + block.getNumBytes() + " from " + inAddr);
```

Log Message

```
2015-10-18 18:05:29,570 INFO dfs.DataNode$PacketResponder: Received
block blk_-562725280853087685 of size 67108864 from /10.251.91.84
```

Structured Log

TIMESTAMP	2015-10-18 18:05:29,570
LEVEL	INFO
COMPONENT	dfs.DataNode$PacketResponder
EVENT TEMPLATE	Received block <*> of size <*> from /<*>
PARAMETERS	[“blk_-562725280853087685”, “67108864”, “10.251.91.84”]

Figure 1.6: *An illustrative example of log parsing [33]*

The defined attributes are captured by the logging point when the event takes place or occurred. The created log message is then parsed into a structured log to be safe in a database or displayed. The logging point should be strategically placed to capture the required attributes to complete the log event [46].

Determine where to place the logging point in a giving software is directly impacted by what the attributes are and if the captured log will be of high quality as described in Table 1.3. The availability of consistent high-quality logs will directly impact the process mining in the analysis of the logs [44].

To strategically place a logging point developers needs to consider what the activation of the logging point will be during the runtime of the software system [30, 35]. The activations can be simple if a statement meets certain criteria or instructions that execute after when an event or action took place (e.g. runtime errors).

The log level or event type of Table 1.5 can be used to determine where the log points should be placed. The aim of the placement of the logging point should try to capture all the log attributes when the event of interest has happened.

Log analysis in Web-based applications

With the defined log parsing and log points in Table 1.3 a log analysis can be made from the stored event logs. The log analysis is the data mining process focussed on the software system's generated event logs [52].

The log analysis will make use of the defined log attributes in Table 1.5 to complete it. Each software system will have some variation of the log attributes in Table 1.5. For a Web-based application, the log attributes will also contain any data about the requests between a web server and web client and its responses [52,53]. This will also reflect on the log level at which weblogs are obtained for log analysis.

1.2.3 System utilisation analysis

In Web-based applications, the process to get usage statistics and user behaviour data is called Web analytics [54]. Web analytics can be used for user modelling efforts and is a form of log analysis. User modelling in software engineering is the customisation and adaptation of the software systems to the users' required needs [55, 56].

The user modelling can also include the implementation of software maintenance as the maintenance is an adaptation of the software system to the user's needs which is the utilisation analysis using event logs. Web analytics focus on different analytics in Table 1.6 for the analysis.

Table 1.6: *Web analytic for user-based data*

Analytic	**Description**
Identity of the user	This is any information about the user's identity in the software system. Users can have different roles when using the software system such as a system admin or general user. These roles mostly dictated what the user can access and do on a website.
Site interaction	The different Web sites the user is accessing during their active Web session. This would also contain all the information about: • how often the users visit a website, • how much time they spent on a specific website, • navigation between different web pages of the website.

These same analytics can be used for none Web-based applications as the:

- identity of the user can be captured if the software system uses a software license,
- different parts of the system can be tracked or services that are used by the user.

Analytic tools for event logs

There exist numerous third-party analytic tools for event logging data monitoring and management to graphically visualise the log data. Choosing the correct tool to use can be dependent on what the software engineers and developers want to analyse and the availability of the tools due to external factors like cost and usability.

Event logging monitoring and management tools are sometimes underutilised and are not often used to their full potential when analysing the event logs [46]. This can be due that the log quality of event logs doesn't meet the standards of Figure 1.5 not met in Table 1.3 for the correct logs to be available for the system utilisation analysis.

1.2.4 Gap identification

In Section 1.1 the importance of software maintenance is discussed in the background and that system utilisation is needed to assist with the software maintenance efforts.

State of the art topics

From the literature, there were a few important focus points identified to create a logging mechanism for user-based activities. These focus points exist for the research done in Sections 1.1, 1.2.2 and 1.2.3.

Table 1.7: *State of the art topics*

Topic	Description	Evaluation criteria
Software maintenance	Software maintenance implementation in industry and best practices when implementing software maintenance.	1. Did the study focus on software maintenance? 2. How to implement software maintenance and resource management of software maintenance? 3. How to improve software maintenance?
Event logging	Event logging in a software environment. The use of event logging and industry practices of event logging.	1. How event logging is used in software engineering? 2. What event logging design and implementation are used?

Continued on next page

Table 1.7 – continued from previous page

Topic	Description	Evaluation criteria
User-activity	Defining user-based events in software systems and tracking the events.	1. How to classify user-based events? 2. Methods to track user-based events?
Web-based applications	Logging for Web-based applications.	1. User-based event logging defined for Web-based applications? 2. Log analysis of Web-logs defined?
System utilisation analysis	System utilisation analysis of software systems using event logging.	1. How is system utilisation analysis done for software systems? 2. How to use event logging for system utilisation analysis?

State of the art summary

Using Table 1.7 the evaluation criteria have been applied to literature studies. In Table 1.8 the studies have been sorted to which topic they were relevant to this study.

Table 1.8: *State of the Art*

Ref.	**Software maintenance**	**Event logging**	**User-activity**	**Web-based applications**	**System utilisation analysis**
[5, 6, 8, 10, 15, 18]	✓	✘	✘	✘	✘
[11]	✓	✓	✘	✘	✘
[14]	✓	✘	✘	✘	✓
[43]	✓	✓	✓	✘	✘
[30]	✘	✓	✘	✘	✓
[33, 34, 44, 49, 50]	✘	✓	✘	✘	✘
[35]	✘	✓	✘	✘	✓
[52, 53, 55]	✘	✘	✓	✓	✓
[57]	✘	✘	✘	✓	✓
[54]	✘	✘	✘	✓	✓
[51]	✘	✓	✘	✘	✓
[56, 58]	✘	✘	✘	✓	✓

Table 1.8 most studies focus on either how to design and implement a logging mechanism, log analysis or software maintenance. Even if industry logging is used to improve software maintenance, most studies only focus on how to create a suitable logging mechanism for different applications as described in Appendix A about logging practice in software engineering.

To improve software maintenance by using logging, a log analysis is needed. Some studies only focus on the analysis of existing logs and how to use them. There were a few studies that made use of user-based event logs for the system utilisation analysis.

Table 1.9: *State of the art*

Ref.	**Software maintenance**			**Event logging**			**Log analysis**
	Types	**Problems**	**Prioritisation**	**Quality**	**Parsing**	**Points**	**System utilisation analysis**
[16]	✓	✘	✘	✘	✘	✘	✘
[19]	✓	✘	✘	✘	✘	✘	✘

In 1.2.1 is mostly focus on software maintenance and how it could be implemented. [16, 19] discusses the diffrent maintenance types that are implemeneted in industry

1.3 Problem statement

Software maintenance is a vital part of the entire life cycle of any software system. Implementing maintenance on software systems is most effective if it's done on systems that are utilised more by users. Some of these systems may also not be in use anymore and can be deprecated to improve system quality and remove unused code to not waste any resources to maintain and run it.

A possible solution to this problem is using event logging to determine the utilisation of the system as it is a proven method industry to track system utilisation usage. Developers have access to third-party software logging tools to get the event logs. Most of the tools focus more on system runtime utilisation than user activities. There exist logging tools that can track user-based activities depending on the framework the software system is developed on.

Developers still need to design the overall logging mechanism and decide where to place the logging points to capture the event logs in a software system. There are proven methods to create a suitable logging mechanism but not all of them include the analysis of the logs for user-based utilisation.

1.4 Objectives of the study

This study aims to design and implement a logging mechanism to track user-based activities to perform an analysis of these logs to improve system maintenance in a software environment. The study is divided into two components to achieve the primary goal, which is the design and implementation of the logging mechanism and the analysis of the system utilisation to improve system maintenance to improve software maintenance.

1.4.1 Logging mechanism:

1. Define logging points for the base event log that needs to be captured. The requirements for a user-based event log should be also defined.
2. Design and implement logging points to capture the event logs using the user-based event log requirements and log attributes.

1.4.2 Analysis of the system utilisation to improve software maintenance

1. Implement a log analysis for the system utilisation of the different software components.
2. Investigate the results and make recommendations to improve software maintenance.

1.5 Overview of the dissertation

Chapter 1: Introduction

This chapter contains the background of software maintenance and system utilisation analysis. It defines the complexities and general issues with software maintenance when implementing it and not implementing it.

Chapter 2: Methodology

This chapter contains the design of the generic method used to create a logging mechanism from a set of defined logging points and log attributes. The software system for which the logging mechanism is made is a Web-based application. The second part of this chapter is the system utilisation analysis by using the captured logs to create an analysis report.

Chapter 3: Results

This chapter contains the results for the defined methodology to create a logging mechanism for system utilisation analysis of Chapter 2. The obtained results will be discussed and validated if they resolved the problem statement of Section 1.3 and fulfilled the objectives of the study in Section 1.4.

Chapter 4: Conclusion

This chapter will provide the conclusion of creating a logging mechanism for system utilisation analysis to improve software maintenance for a Web-based application. Limitations and recommendations will also be made based on the methodology and results.

Chapter 2

Methodology

2.1 Preamble

The literature in Chapter 1 is used for the method to create a logging mechanism that can capture user-based activity logs to improve software maintenance by analysing the obtained logs. The Web-based application system on this logging mechanism will be implemented on is an energy management system for the mining industry.

In Figure 2.1 is the basic design of the Web application the users interact with where each mine group has different toolboxes linked to it and each toolbox has different dashboards linked to them which the users can access and interact with. The activities of each of these dashboards and how the user navigates through them are necessary for the system utilisation analysis.

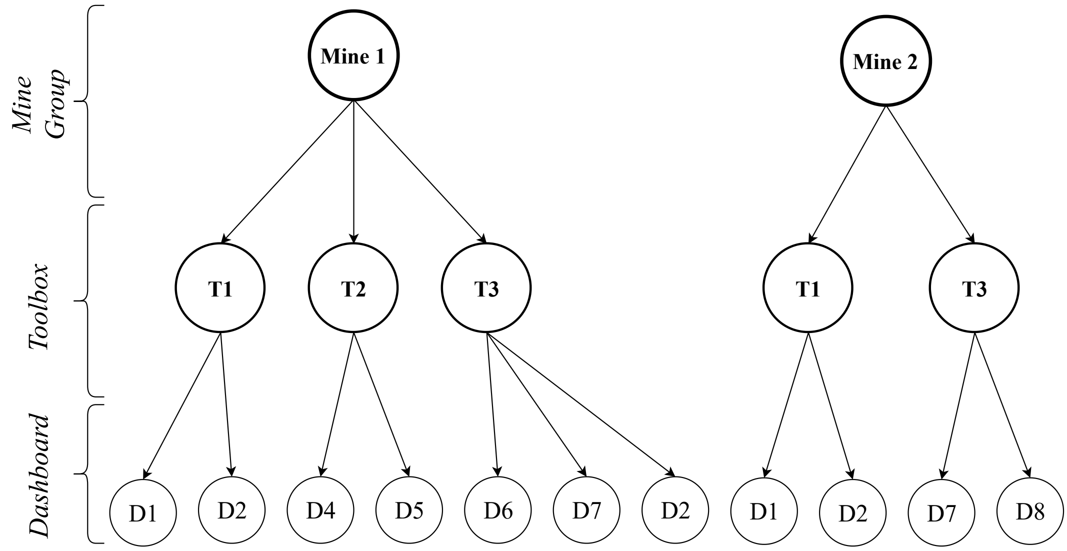

Figure 2.1: *Basic design of an energy management system*

In Section 2.2 the methodology to create a logging mechanism to capture user-generated events is discussed for web-based applications. The different functional requirements and interfaces are discussed in this section [59].

In Section 2.3 the methodology is discussed to analyse these obtained logs to improve software maintenance by using various tools visualisation tools or creating them based on the log attributes that are available.

2.2 Logging mechanism

The logging mechanism will need to meet the requirements discussed in Section 1.2.2 to capture the required logs to apply system utilisation analysis on it. Figure 2.2 is the design for the logging mechanism to capture the user's activities. In this figure, the logging mechanism is split up into two functional requirements parts (F/R) which consist of the client and server functional requirements.

Each functional requirement has an interface requirement that transfers the data from one interface to another interface. These interfaces are labelled as I/F in Figure 2.2. The Figure 2.2 is the client interface (**F/R 1**) and the server interface (**F/R 2**) that forms the entire logging mechanism to capture the user-based activity logs.

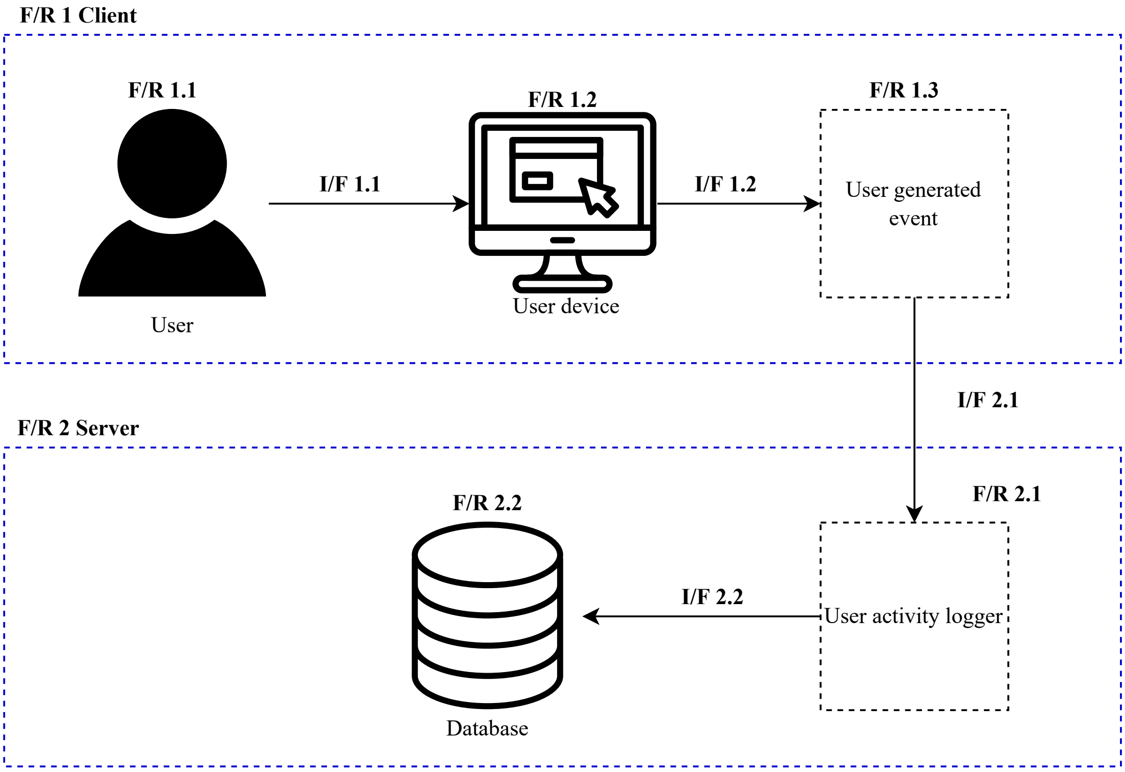

Figure 2.2: *Logging mechanism architecture design*

Clients functional requirements

The client's functional requirements (F/R 1) are where the user-based activity is triggered. In Figure 2.2 the client interface consists of three main functional requirements. These interfaces in Table 2.1 parse the input from the user to create a basic user-based action that can be parsed and captured and parsed by the logging point to create a user-based log that is parsed onto the server for further processing.

Table 2.1: *Client functional requirements (F/R 1)*

Requirement ID	**Name**	**Description**
F/R 1.1	User	The user serves as the primary initiator of the user-based activity events.
F/R 1.2	User's device	The device that the user uses to access the website from where the user-based activity events are generated.
F/R 1.3	User-generated events	These are the captured user-based activity events that have been identified by the logging point and will be sent to the server.

Server's functional requirements

The server functional requirements in Table 2.2 for Figure 2.2 is the rest of the logging mechanism. At this stage, the obtained user-generated event of Figure 2.6 will be attempted to be completed into a user-based activity log and stored in a database by the logging point.

Table 2.2: *Server functional requirements (F/R 2)*

Requirement ID	Name	Description
F/R 2.1	User activity logger	The logging points are used to capture and create the user-based event log that will be stored in a database.
F/R 2.2	Database	The event log is stored in a database until it is needed for further analysis.

2.2.1 Requirements for a user-based activity log

The user is the initiator of the logging mechanism. Each action or event they trigger by interacting with the user interface on their device (F/R 1.2) can be a potential user-generated event. In Table 2.3 is the sub-requirements for the user (F/R 1.1) which the event log should fulfil to be classified as an user-based activity log.

Table 2.3: *Requirements for an event to be a user-based activity*

Requirement ID	Description
F/R 1.1.1	The event has to be triggered by the user interacting with the user interface using their device and not any other events that the system will self-initiate. The user needs to have interacted with the UI directly. This can also be validated by tracking if the user did interact with the UI from the HTML element ids.
F/R 1.1.2	The event must consist of different cases ($ca \epsilon CA$ the cases consists of events) which are noteworthy to make the event log identifiable [52].
F/R 1.1.3	For certain types of event logs for F/R 1.1.2, the user-generated event should have an origin from which the event took place.
F/R 1.1.4	The event log should consist of attributes that expand the identity of the user-based activity.
F/R 1.1.5	The event must have the user as the initiator or input for the user-based activity. This will exclude all events triggered by the system as the user did not directly start the event.
F/R 1.1.6	Only use the first *HTTP requests*[1] that is sent to the server.

Every interaction the user has with the user interface of the device to the software system can be seen as an event triggered by the user. Most of these events won't have a meaningful impact as they won't fulfil F/R 1.1.2 and F/R 1.1.4 in Table 2.3.

For the user activity event to meet the requirement of F/R 1.1.2 it has to have defined cases that describe the activity type of each event. These activity types form the basic criteria for which events can be parsed which significantly reduces the number of logs that will be obtained. This will ensure that the event logging process will produce quality user-based logs as discussed in Table 1.3:

- A basic structural complexity to simplify log parsing and development of the logging points in the system,
- Keep the logging consistent by not deviating from the defined cases, and
- Ensure that the event log's other attributes are complete and available to increase the accuracy and trustworthiness of the event logging when further system utilisation

analysis needs to be done.

2.2.2 User activity types

The user-activity logs will be split into three main event types as in Table 2.4. The general user activity event type (F/R 1.2.3) will the be most common user activity event and be split up into different user activity events. This is determined by the need of what utilisation stage requires to analyse specific user activity events.

Table 2.4: *User activity types*

Requirement ID	Activity Type	Description
F/R 1.2.1	Web page accessed	The user may navigate through different web pages in a session.
F/R 1.2.2	Session changes	This is any user activities excluding F/R 1.2.1 that modifies the user's session: • Logging into a Web application. Both Successful and failed attempted logins. This user-based activity may cause the log attributes that identify the user will be a `NULL` value as the user's session has not started yet to verify their identity, • Ending their session through by logging out or declining to extend their session when it is about to expire, • Modifying any session or other relevant variables that can be used in the utilisation analysis
F/R 1.2.3	General activity	Any events excluding the first two user-based activity types that the user initiates when they interact with the web page. Most of the user activity logs will have this event type.

These user activity types can be further expanded in general activity (F/R 1.2.3) for analysis purposes. The general activity types will be different for each system based on what the system enables the user to do or what is needed for further system utilisation analysis such as determining if the action the user triggered was to generate a report that they downloaded.

2.2.3 Logging points

In Table 1.3 the logging points should be strategically placed in the software system to capture the log attributes for the user-based activity log. To meet the requirements of Table 2.3 for a user-based activity the logging points should adhere to the logging points functional requirements of Table 2.5.

Table 2.5: *Logging points requirements*

Requirement ID	Description
F/R 1.3.1	The logging point should be placed where the user's interaction with the software system will send a *request* back to the server.
F/R 1.3.2	Each logging point should consistently capture the user-based activity as the activity is happening.
F/R 1.3.3	Logging points should be globally complete to capture the user-based activities in the giving software system without too much modification between each point in the same software system.
F/R 1.3.4	The logging points should not interfere with the rest of the system's operations, this would be slowing down the system by causing too much overhead in each *request* that is being sent.

The logging points can either be a single code segment or consist of multiple code segments in a software system that aims to capture user-based actions as they happen. Creating multiple logging points in a software environment will:

- Increase complexity of the logging mechanism. Each point can be different from the other as it will need certain operations to capture the log,
- The consistency of the logging might differ and increase as the logging points increases in a software system.
- The correctness of the logging will be impacted if the different changes in the logging point if the logging points are unable to consistently capture the user-based activity or extract all the needed attributes to complete the user-based log.

Creating a single logging point reduces the complexity and in most cases will improve the consistency and correctness of the user-based logs. In Web applications a globally defined logging point can be used in a modified *AJAX request*[2] that will form the base template for all or most *AJAX request* used in the software system as in Section 2.2.5.

The use of a single centralised logging point doesn't guarantee that the logging mechanism will perform more efficiently and accurately than using multiple logging mechanisms. Using a single logging point may have complexity issues when it needs to capture each user-based activity consistently with different cases.

2.2.4 Log attributes

The defined logging attributes in Table 2.6 are the base attributes that form part of the main structure of the user-based event log. For web-based applications on the client side,

[2] **AJAX** stands for Asynchronous JavaScript And XML. It uses an XMLHttpRequest object to communicate with servers that can send and receive information in various formats, including JSON, XML, HTML, and text files. [61].

only some of these attributes can be obtained as the rest of the attributes can be resolved on the server side. The metadata (F/R 1.4.6) can consist of the request parameters that are obtainable on the server side but any additional captured data can be added and sent to the server.

Table 2.6: *Logging attributes*

Requirement ID	**Logging point**	**Description**
F/R 1.4.1	Identification number	The activity identification is an incremental number of the user-based event that is logged.
F/R 1.4.2	Timestamp	This is the time the user initiated the user-based activity event. This will be the timestamp the log was written into the database as the log will be made before the rest of the intended *HTTP request* is completed.
F/R 1.4.3	Activity type	Each event can be classified into user-based types. This is the user-based activity types in Table 2.4.
F/R 1.4.4	User identification	Each user has a unique identification number that links the event to them if their session has been verified and can be obtained. Will not be available when the user tries to log in to the system as their session has not been set yet.
F/R 1.4.5	Request origin	In web applications, there are always requests sent back to the server which will call the primary function to handle the request. This can be logged as either the file that the request is being sent to or the Web page from which the request came.
F/R 1.4.6	Metadata	The metadata of the event contains request parameters or other relevant request data of the event. This metadata adds more information about the user's activity. In Listing 2.1 is an example representation of the metadata that can be created for most user-based logs. Some of the event types may not have metadata added.
F/R 1.4.7	Miscellaneous	These are any non-metadata attributes that can be consistently captured to be used in the utilisation analysis. They expand the characteristics of the obtained user-based log beyond the base attributes.

Each of these log attributes combined creates the base log from which key logging points can be created in the software system to capture the user-based activity logs in Table 2.6. The activity type (F/R 1.4.3) is can be assigned during the user-based activity identification phase with a default value and resolved to a new activity type based on metadata or other parameters by:

- If it alters any of the session variables that are relevant to the system utilisation analysis,
- Access a certain part of the software system that needs all the user-based activities set to a certain type based on the nature of the procedures that need to be executed such as triggering a generation of a report that can be its user-based activity type.
- The activity type is also sorted by HTML element tags such as a button or text box.

The metadata in Listing 2.1 is the possible extra parameter that can be obtained for the user-bade activity log. These parameters can either be captured at the client side by the logging point or can either be captured on the server side when the rest of the log's attributes are being obtained.

```
{ "RequestTarget" : "/Area4/Controller4/TestFunction",
  "RequestElementID" : "Button4",
  "RequestParameters": {
    "Parameter1": 4,
    "Parameter2": "Hello World!",
    "Parameter3": true
    "Parameter4": 40.404
    "Parameter5": {
      "Parameter6": "Car",
      "Parameter7" 160000.00
    }
  }
}
```

Listing 2.1: *Metadata JSON*

The metadata will need to store as a JSON string as it can be a complex object that doesn't have a set number of parameters. This complex object can have:

- The `RequestTarget` parameter can be a file path for the *Controller* or Webpage's absolute request path from where the user initiated the event. It also contains the function that is being called by the *HTTP request*.
- The `RequestElementID` is the HTML element id which the user interacted with that cause the user-based activity. This can be used as another validation that the event was caused by the user. Some of the user-based activities can be set to some of these HTML element types by getting the HTML element tag.
- The `RequestParameters` is all the parameters in the *HTTP request* that can be serialize into a JSON string. This can be used to determine what the user tried to do by using this input for the specific function which is used for F/R 1.1.6 in Table 2.3.

2.2.5 Web application architecture

To determine the user activity types for a Web application, the Web application's architecture will be a factor in the logging mechanism. Web applications consist mostly of HTML, JavaScript and CSS programming languages. The Model-View-Controller (MVC) architecture is mostly used for web-based applications using that programming language [62]. The MVC architecture in Figure 2.3 consists of 3 basic parts which are the [62]:

- *Model:* Is the representation of the records in the database which also interacts with the database through a database access layer or service manipulating the data by using the CRUD operations:
  - *create* operation that adds new data,
  - *read* operation that gets the data from the database,
  - *update* operation that modifies the existing data,
  - *delete* operation that removes data.
- *Controller:* Is operates both the *View* and *Model* and serves as the connection between the user and the system by controlling the data flow of the *Model* and *View.*
- *View:* This shows the results of the data contained in the *Model* and enables the user to manipulate the data. The user will only interact with this part of the Web application.

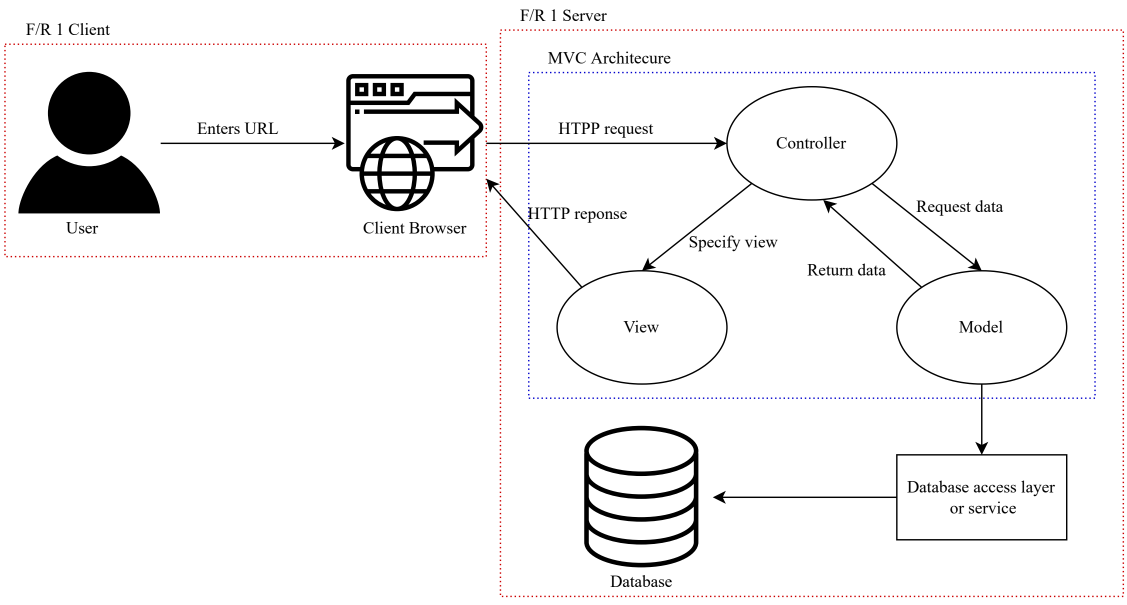

Figure 2.3: *MVC architecture for most web-based applications [63]*

In Figure 2.3 is the MVC architecture equivalent representation of Figure 2.2 where the data flow is shown of the MVC architecture. The user interacts with the Web application through their browser which will send a *HTTP requests* to the *Controller* and receive a *HTTP response*[3] from the *View.*The *Controller* will request and return the data to the *Model* which interacts with the database access layer or service to do the *create*, *update* and *delete* operations.

[3] An **HTTP response** is made by a server to a client. The response aims to provide the client with the resource it requested, inform the client that the action it requested has been carried out; or else inform the client that an error occurred in processing its request. [64].

To classify any interaction between the user (F/R 1) and server (F/R 2) to fulfil the functional requirements of Table 2.3 only the *HTTP request* are used for the logging points in Section 2.2.3 as it:

- Meet the F/R 1.1.1 and F/R 1.1.1 as the user will interact with the *View* to modify the data which needs to send back an *HTTP request* to process the data on the *Controller*.
- User activity types can be assigned for different scenarios that the user triggers when the request is being sent.
- Any additional metadata can be sent with the *request header*[4] of the *HTTP request*. This will reduce the overhead added by the logging mechanism by not sending additional *HTTP request* each time back to the server when a user-based activity has been identified.

Most Web applications will make of use JavaScript to control the content of the page that is being displayed to the user. The primary method would be making use of an *AJAX request* to communicate with the server to fulfil the user's action.

The *AJAX request* has some key features that will enable the logging points discussed in Section 2.2.3 to capture some logging attributes and classify the event as a user-based activity.

The `beforeSend` setting of the *AJAX request* enables tracking in real-time for the HTML element id and its HTML tag or other possible log attributes that need to be added to the request for the log point to capture it. The captured HTML element id, HTML tag and some other parameters can be added as a custom request header that will be sent to the server.

```
$.ajax({
  url: "https://fiddle.jshell.net/favicon.png",
  beforeSend: function (xhr) {
    xhr.overrideMimeType("text/plain; charset=x-user-defined");
  }
}).done(function (data) {
  if (console && console.log) {
    console.log("Sample of data:", data.slice(0, 100));
  }
});
```

Listing 2.2: *AJAX request example [66]*

2.2.6 Obtaining the element of user-based event

In Section 2.2.5 the user-based activity event will be use a *HTTP request* to send to the server when the user interacted with an *HTML element*. For the functional requirements activity type (F/R 1.5.3) and metadata (F/R 1.5.6) in Table 2.6 the *HTML element* needs to be obtained to get the element's tag and identification text. This can be difficult to obtain due to *bubbling*[5] that may occur when searching for the element that the user specifically

[4] A **request header** is an HTTP header that can be used in an HTTP request to provide information about the request context so that the server can tailor the response. For example, the Accept-∗ headers indicate the allowed and preferred formats of the response. [65].

[5] **Bubbling** is when an event happens on an element, it first runs the handlers on it, then on its parent, then up on other ancestors. [67].

interacted with.

In Figure 2.4 is the event propagation example of a child element that has been clicked on which executes a DOM event. The event propagation consists of three phases [67]:

- *Capturing phase:* The event propagates downwards to the targeted element that the user interacted with.
- *Target phase:* The event reaches the targeted element to execute the DOM event.
- *Bubbling phase:* The event bubbles up from the targeted element

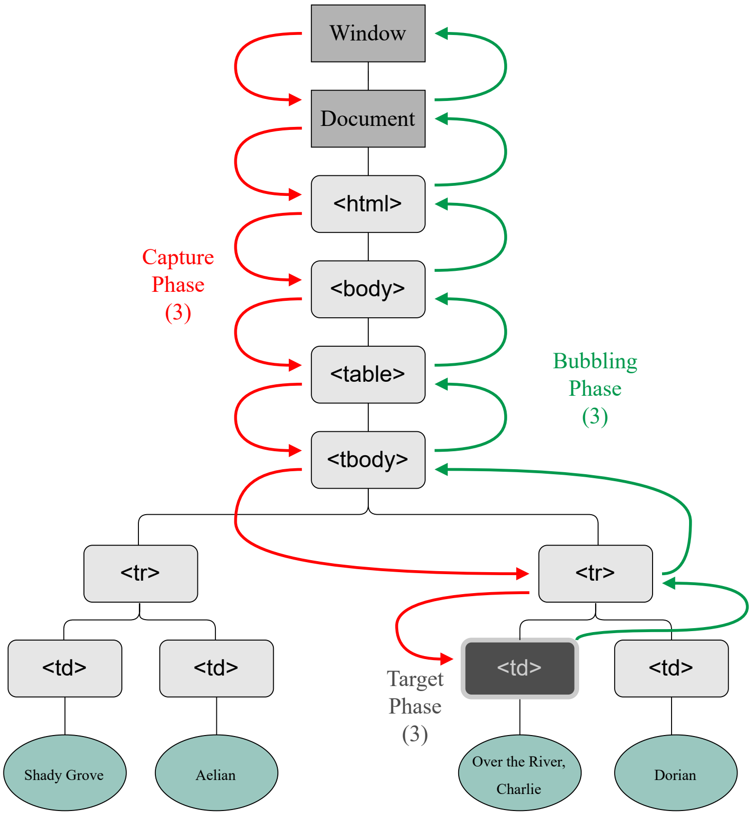

Figure 2.4: *JavaScript event propagation [67]*

Capturing the targeted element may be difficult as some Web pages may have more complex HTML where the event propagation may sometimes not obtain the correct element information which the user interacted with. Another DOM event may have started during the initial element's event, therefore it is more accurate to obtain the targeted element by obtaining the last known element the user hovered over on the user interface.

In Figure 2.5 is the flow diagram to capture the element user interacted with for the user-based activity log. This code segment will be initiated during the `beforeSend` operation of the *AJAX request* to filter HTML elements by predefined allowed elements to use. Filtering the element tag names ensures that unwanted more complex elements or more basic elements that are not expected to be the initiator of the event will be used.

If the Web location already changed or no element exists, the contents of the page might have already changed during the event propagation. The last known element that the user hovered on must be used as most likely might have been the element that the user interacted with. This will ensure there is always an element that has been detected and parsed with the request header in most UI changes.

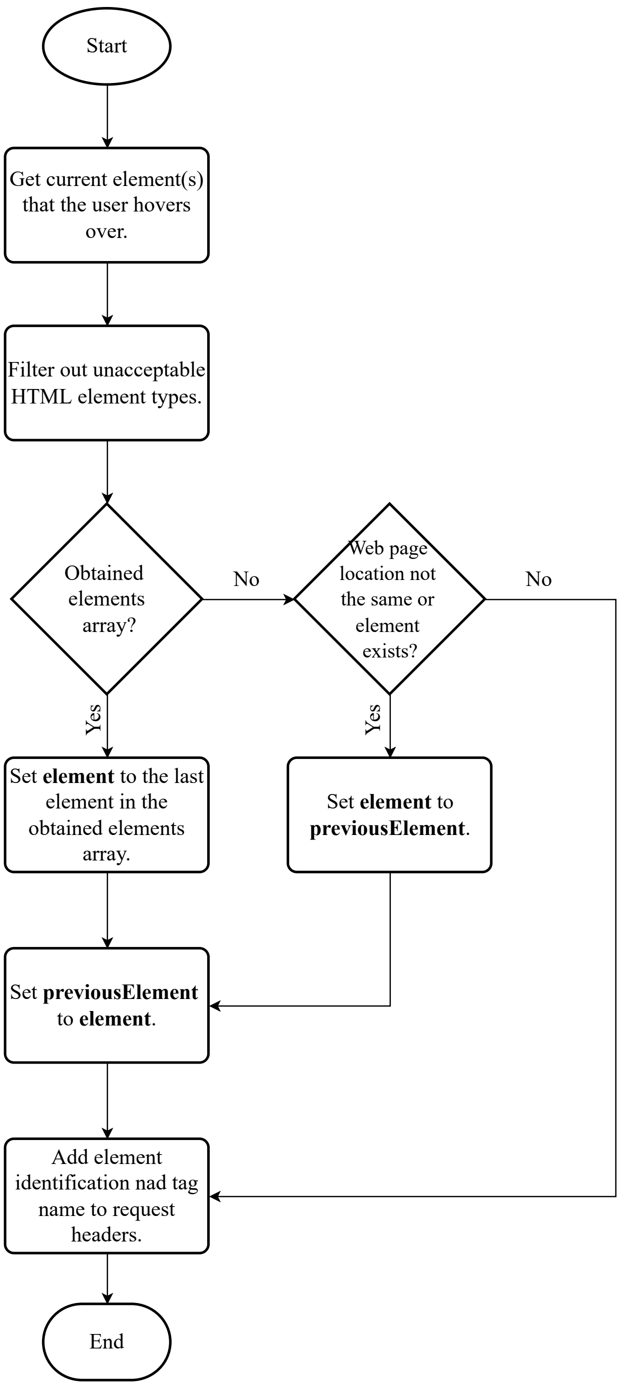

Figure 2.5: *HTML element capturing flow diagram*

2.2.7 Client functional requirements interaction

In Figure 2.6 is the complete process of the user interacting with the user interface to trigger a user-based activity event to be logged later for the client's functional requirements. It starts with the user interacting with the user interface. In the `beforeSend` operation of the *AJAX request* the HTML element's id and tag should be attempted. The default activity type is set to general activity (F/R 1.2.3) until it is further processed later in the logging mechanism.

If the activity has any additional metadata such as other request parameters, it will also be logged by adding searching for it in the `beforeSend` operation. The other metadata can also be captured in this stage from the client side like the element that the user clicked to initiate the event. The captured metadata is placed in a custom request header afterwards, and the *AJAX request* continues its normal operations and sends the data back to the server.

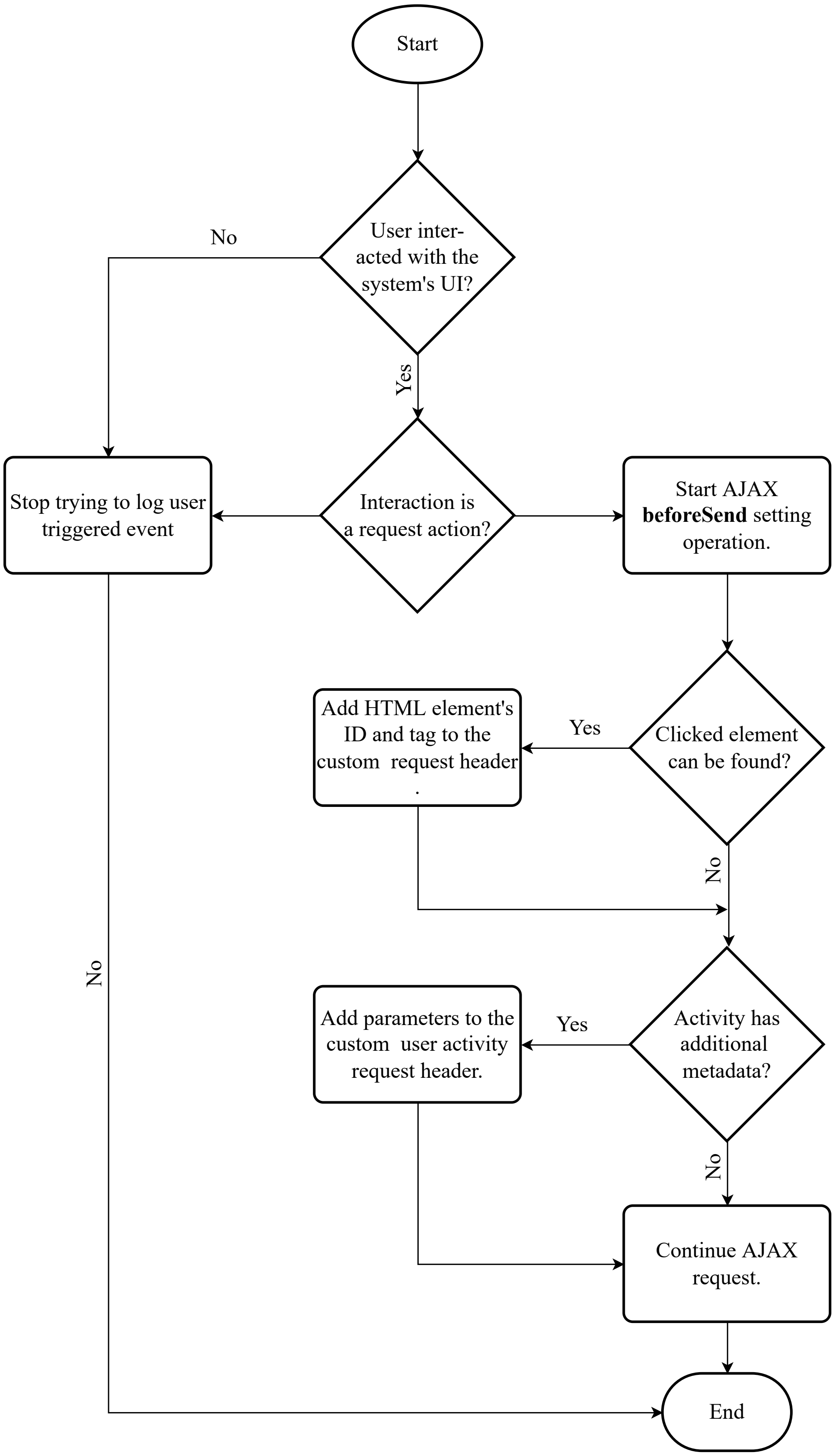

Figure 2.6: *User-based activity log classification flow diagram*

2.2.8 Server side logging point

In Section 2.2.3 the functional requirements for the logging point need to be fulfilled to create a suitable logging point for a software system to track user-based activities. The *HTTP request* will call a function in the *Controller* to execute the user's actions, these *request* information can be obtained and parsed on to the logging point.

The logging attribute can be created as a centralised code segment that all the software system's components can execute before executing the targeted software system in Web applications such as:

- In frameworks like `C#`'s *.NET Framework*[6] the *HTTP request* data can be extracted from the in build *HTTP request* models to obtain the custom request headers set on the client side. Using the `ActionFilterAttribute` enables the creation of a global single logging point that can be called for every *HTTP* function to execute the logging point first before continuing with the rest of the main targeted function. The rest of the log attributes can also be obtained during the execution of the filter by using the `FilterContext` to obtain:
  - **Absolute URI path:** The string containing the absolute URI path of the currently active controller is part of the `FilterContext`. This does not reference the controller that handles the request at all times but the controller that the user is active on before initiating the request.
  - **Absolute request URL:** The requested URL contains the targeted controller's name and function that the request needs to execute.
  - **Action parameters:** The `FilterContext` contains action parameters which are the request parameters sent with the AJAX request from the client device.
- In other older Web applications that is created with programming languages such as `PHP` a more direct approach needs to be taken when accessing the request data. In this case, using multiple logging points that call the logging points' main code segment to capture the attributes and store the log in a database. The parameters may need to be extracted before parsing them to the main logging point code segment.

As long as the logging attributes and the *HTTP request* headers are obtainable, the logging mechanism can be created on the server side to extract the data and process it. The activity type can be resolved by the defined cases e.g. if the request calls the `Index` function of the *Controller* of the Web page, it can be identified as the Web page accessed user activity type (F/R 1.2.1) of Table 2.4.

If the user-based event is using the *Controller* or functions that modify the session, it can be classified as session changed event (F/R 1.2.3) and the rest of the user-based activity events need to be tested afterwards if they meet certain criteria defined for the general activity types. If it fails all three types of classification the event is likely not user-generated or comes from *AJAX request* that was executed after the initial first request. In such cases, the last HTML element id that triggered the event should not be listed as a clicked element in JavaScript.

[6] **.NET Framework** is a run-time execution environment that consists of common language run-time (*CLR*) and a `.NET Framework Class Library` [68].

2.2.9 Storing the user-based actvity logs

In Section 2.2.8 the logging point at the server side needs to extract the log attributes to be placed in the database. The database that is going to be used for the software system is a `MySQL` database.

The captured parameters of the log attributes may have some sensitive user data that should not be logged. Functions can be excluded or assigned a new user activity type that will need to filter out certain parameters or not log any parameters a all. This will be any functions that include:

- Session handling functions that contain passwords or other user information that should not be available for anyone but the user. This could lead to unintentional information disclosure of any personal information in the system utilisation analysis if it is available for anyone who can see and use the user-based activity logs,
- Complex parameters such as file upload streams of files that the user tries to upload. This information cannot be broken down to a simple `JSON` structure as in Listing 2.1, other metadata such as the file size, name and type can rather be logged. This can also be defined as a separate user-based activity event type by detecting these complex parameters.

In Table 2.7 is the SQL data type of the parameters and the functional requirements that it needs will need to fulfill of Table 2.6. Additional for more modern systems such as the `C#`'s *.NET Framework* the request origin (F/R 1.4.5) can be split into two different fields:

- The **Area** is the subsystems where different MVC systems are grouped.
- The **Controller** this field will only contain the name of the controller that executes the request.

Table 2.7: *Log attributes for SQL table*

Column Name	SQL Data Type	Requirement
ActivityID	INT(11)	F/R 1.4.1
Timestamp	DATETIME	F/R 1.4.2
ActivityType	ENUM	F/R 1.4.3
UserID	INT(4)	F/R 1.4.4
Subsystem	VARCHAR(45)	F/R 1.4.5
Controller	TEXT	F/R 1.4.5
GroupID	INT(4)	F/R 1.4.7
MetaData	JSON	F/R 1.4.6

The log attributes in Table 2.7 will have foreign key references to other tables in the database. In Figure 2.7 is an ERD diagram that describes the relationship of the table created to store the log attributes with other relevant tables. In the system utilisation analysis, this enables different fields of the other tables to be used to categorise the logs.

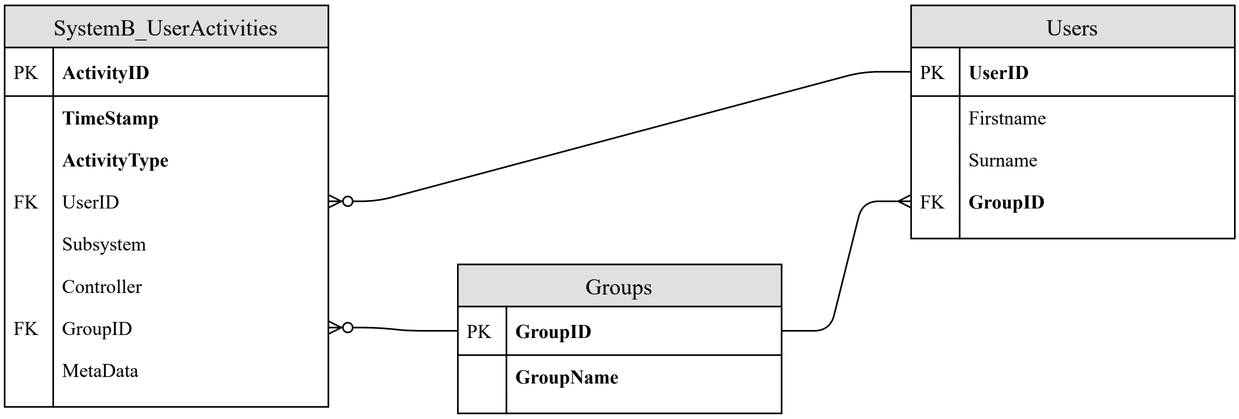

Figure 2.7: *ERD of the user activities*

2.2.10 Server side log parsing

In Figure 2.8 is the server side log parsing of the obtained possible user-based activity events using for a *.NET Framework* software system. The defined `ActionFilterAttribute` will start the user-based activity process before the targeted process is executed. At this stage, if anything goes wrong with the logging at during the execution of this filter, it should be abandoned and let the software system continue to ensure that it doesn't interfere with the software system's operations (F/R 1.4.4 of Table 2.5).

In the case of the request method `NULL` or empty due to errors such as incorrect parameter types for the targeted procedure in the controller, the logging point should stop attempting to log the user-based log. The issue would most likely appear as a runtime error and any user-based activity logging procedures will also fail due to incomplete data or cause the logs to be not complete and consistent (F/R 1.4.3 of Table 2.5).

If the captured user-activity log contains any parameters it should be checked for any session-related parameters or any other potential user data that should be removed from the meta-data to prevent any personal information from being accessed by not the owner of the user account. If it doesn't contain any request parameters the `ElementInfo` should be set to `NULL`.

The `ElementInfo` contains all the metadata send from the client side logging point that captured the HTML element data in Figure 2.5. In Listing 2.3 is the `JSON` data of the `ElementInfo` which consist of:

- `ElementTagName`, is the HTML element's tag name which is one of the defined accepted tag names such as `button`, `label` and `td` etc.,
- `ElementID`, identification of the element if it has been assigned to the element and can be obtained on the client side,
- `ElementDataKey`, additional captured data attributes that expand on the identity of the element if it is a custom made HTML element control. Some software systems may have other custom-created HTML elements which also can trigger a user-based activity. It can be other miscellaneous elements such as a `label` which are not normal input controls.

```
{ "ElementTagName" : "button",
  "ElementID" : "submitButton",
  "ElementDataKey": "submit-control"
}
```

Listing 2.3: *Element properties JSON*

If the `FilterContext`'s requested procedure is called and it is the `Index` which is the first procedure that needs to be executed for a Web page being accessed. This activity type is the first user activity type at this point of the log parsing before it is processed again to another user activity type.

If it is not the `Index` the process will continue to the next operation which checks if the `ElementInfo`'s `ElementDataKey` is either a null or empty value. If there is any data available the activity type can be set to the custom control defined activity type or just custom control to represent all these custom-made elements. The `ElementID` is set to custom control element or the defined custom control's identification.

If the `ElementInfo`'s `ElementDataKey` is null or an empty value the user activity type is set to the element's defined activity type. After the activity type is resolved the request origin of the user-based activity is obtained by getting the request's absolute path.

After the request origin has been obtained, other relevant session information such as the group that represents a certain entity data can be obtained as well as the user's identification and other relevant metadata that is available at this stage to complete the log attributes that needs to captured from Table 2.6 to complete the user-based activity log.

The data is parsed to the activity logger that will write the log into a database if the log was successfully obtained. This will end the logging process until a new user-based event log is ready to be processed and stored in the database.

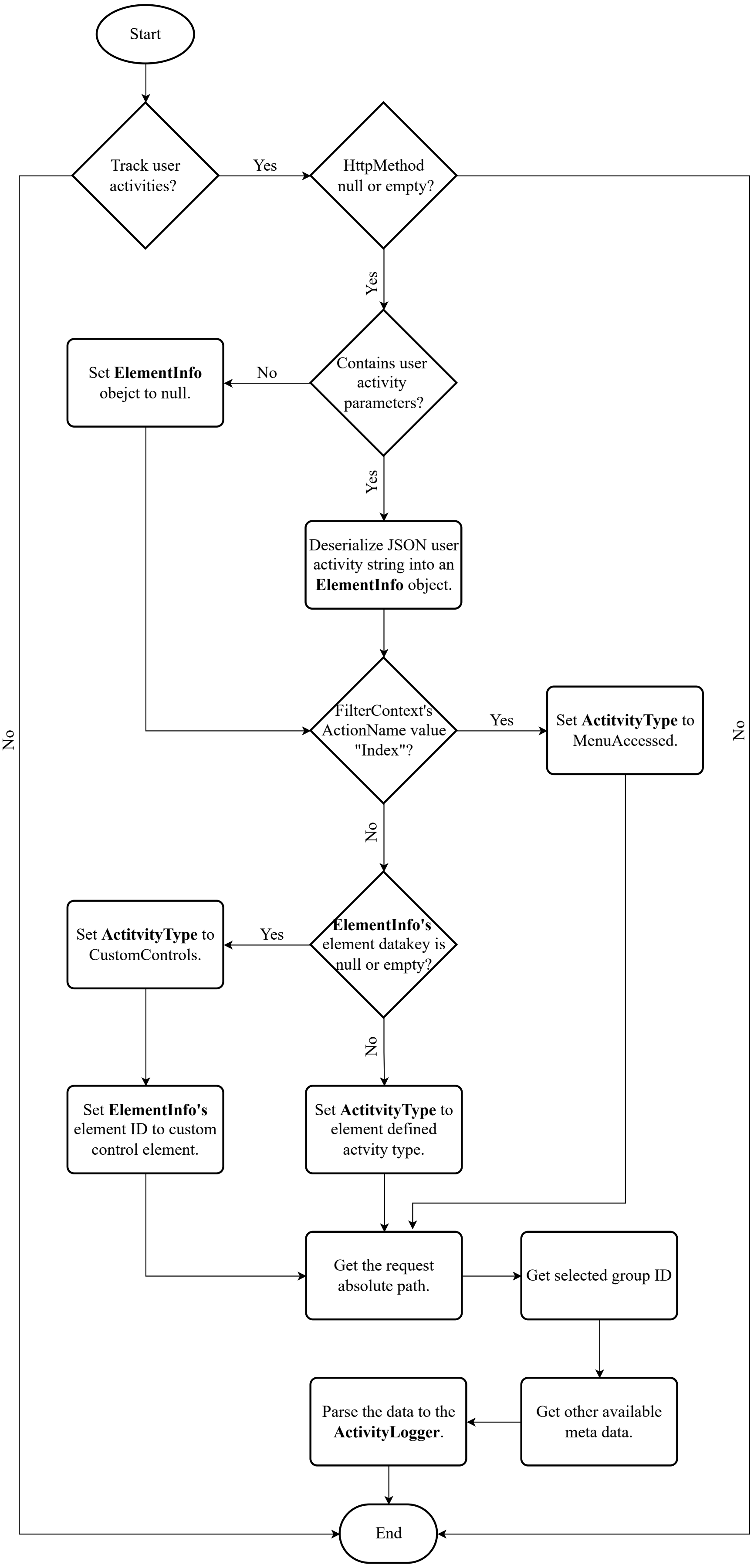

Figure 2.8: *Server side log parsing flow diagram*

2.3 System utilisation analysis

The system utilisation analysis will make use of a Web-based graphical user interface to view the raw logs stored in the database and use any other visualisation tools for data insights into the user-based activities. In Table 2.8 are the functional requirements for the utilisation analysis.

Table 2.8: *System utilisation analysis functional requirements*

Requirement ID	Requirement name	Description
F/R 3.1	Log availability	The user-based logs should be available for any defined period the logging mechanism was actively capturing the user-based events.
F/R 3.2	Log completeness	The user-based logs should be complete and there should be minimal corrections made post-logging during the log extraction process (F/R 3.3) and visualisation presentation (F/R 3.4).
F/R 3.3	Log extraction	The user-based logs are extracted from the database and imported into a visualisation presentation (F/R 3.4) for the user-based activity logs.
F/R 3.4	Log visual presentation	The visual presentation of the extracted logs should be shown to the user that will make use of the activity logs in a custom visual system or make use of other third-party tools. This will impact how the logs will be extracted (F/R 3.4) from the database as third-party systems may make use of an API to get the logs from the database.
F/R 3.5	Log comparison	By Using the F/R 3.2 the utilisation between different log attributes that are used as the defined criteria. This will be to group and compare different types of users, subsystems and activity types against each other etc.
F/R 3.6	Maintenance suggestion	Maintenance suggestions can be made from the system utilisation reports by prioritising maintenance or decommissioning software systems. This can be data or visual representations of the log comparison (F/R 3.5) using the log visual presentation systems (F/R 3.6) or creating a summary report from the visual presentation that contains the maintenance suggestions.

Each of these functional requirements ensures that the system utilisation analysis will be achieved for the created logging mechanism in Section 2.2. The main user interface of the system utilisation analysis will consist of the presentation of the user-based activities (F/R 3.4). This system will either be a custom-created system to display these logs or third-party software such as Microsoft's business intelligence platform, PowerBI.

Using the third-party tools has advantages over creating custom software for the visual presentation (F/R 3.4):

- Third-party business intelligence platforms have all the necessary analytical functionality. The tables and charts needed for the visual presentation can be created with minimal programming difficulty.
- The advanced tools in these third-party business intelligence platforms provide more ways for the user-based activity logs can be visualised for the log comparison (F/R 3.5).
- Maintenance and editing of these third-party representations are mostly trouble-free to do. The amount of support and guides that should be available to the developer instead of relying on the creator or other developers that are available to make updates to the custom visual presentation.

These third-party tools do indeed have some other drawbacks such as:

- Third-party business intelligence platforms most likely will require some sort of subscription that can be very costly for a company licence.
- Extra courses might be needed to fully use the capabilities of these platforms.
- Additional functionality such as APIs might be needed for the log extraction (F/R 3.3) to import the data to the platform.

With the drawbacks listed above the third-party business intelligence platforms is the better visual presentation tools for the system utilisation analysis if it is available for use than creating and managing a custom visualisation platform.

2.3.1 Log availability and completeness

The log availability (F/R 3.1) and completeness (F/R 3.2) functional requirements can be achieved all the functional requirements of the user-based activity log in Section 2.2.1 are accomplished with minimal processing of the raw logs afterwards. There will be always changes made to the system that can impact which possible user-based events are considered to be logged.

2.3.2 Log extraction and visual presentation

Log extraction refers to the methods used to obtain the logs from the database with any other relevant data that can be used in the visualisation presentation (F/R 3.4). The raw logs will need to make use of the foreign references to other tables in the database to provide more detail about the user-based event log as in Figure 2.7.

In Figure 2.9 is an example of a visual presentation of Microsoft's PowerBI report that displays imported data. These reports will contain the extracted data of Figure 2.7 using either an API or will extract the data like the rest of the system if it is a custom visualisation.

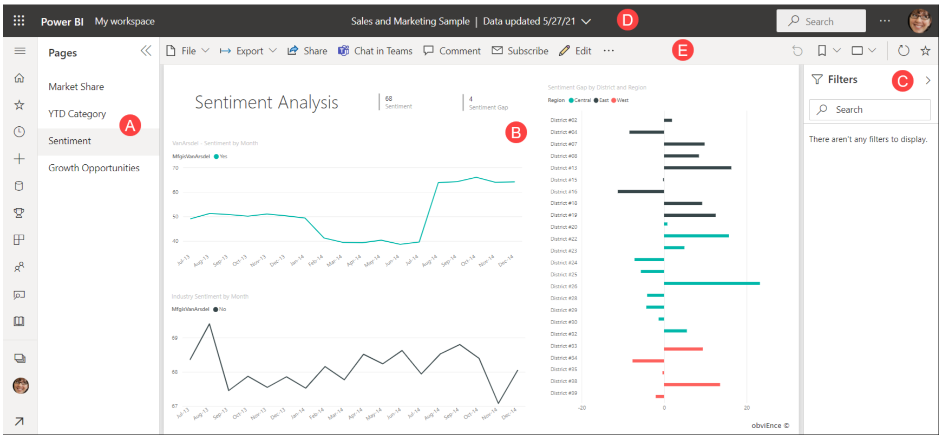

Figure 2.9: *Example of a visual presentation*

2.3.3 Maintenance improvements

The system utilisation analysis aims to provide maintenance recommendations to the developers to improve their maintenance efforts by:

- Prioritising the maintenance efforts on more frequently used systems.
- Decommission unused systems. The user-based activities provide a quantitative reason why certain systems can be decommissioned due to inactivity from the users.

Table 2.9: *System utilisation analysis categories*

Requirement ID	Requirement name	Description
F/R 3.6.1	Users	The users of the software systems can be put in different categories based on who uses the software. This can be both the customer users or the employees using the software. Using the activities of the customer users will provide the data on which systems the development team needs to put their resources into.
F/R 3.6.2	User activity types	The user activity types in Table 2.4 can be used as a category to compare different user-based activity types with each other and use a sub-category for categories such as the different users that can use the system (F/R 3.6.1).
F/R 3.6.3	Subsystem or controllers	The request origin (F/R 1.4.5) of the user-based activities can be categorised to compare different subsystems and controllers to each other.
F/R 3.6.4	Miscellaneous categories	This user-based activity category type will make use of the metadata attribute (F/R 1.4.7) of Table 2.6. The other fields which are not set as main categories can be also placed in this category as they can take multiple forms.

2.4 Verification

Using the functional requirements defined in Sections 2.2 and 2.3 the system can be verified that it satisfies the system requirements in Table 2.10.

Table 2.10: *System requirements for verification*

Requirement	Methodology reference	Satisfied
User activity types	Sections 2.2.1 and 2.2.2	✓
Log attributes	Section 2.2.4	✓
Logging points	Sections 2.2.3, 2.2.5, 2.2.6 and 2.2.8	✓
Log extraction and visualisation	Sections 2.2.9 and 2.3.2	✓
System utilisation analysis	Section 2.3.3	✓

2.5 Conclusion

In Section 2.2 are the defined functional requirements for a logging mechanism for the system utilisation analysis in Section 2.3. The user activity types defined in Section 2.2.2 are the base of what log attributes need to be logged to create a user-based log.

The logging points captured logs and send them to the server's logging point where it is processed and stored in writing in a database.

The logs are extracted into a visual presentation which enables the maintenance improvement suggestions based on the utilisation analysis in Section 2.3.

Chapter 3

Results

3

3.1 Preamble

Introduction of the chapter.

3.2 Logging mechanism

3.3 Utilisation analysis

3.4 Conclusion

Chapter 4

Conclusion

4.1 Preamble

4.2 Overview of study

4.3 Recommendation for further research

References

[1] C. Gralha, D. Damian, A. I. Wasserman, M. Goulão, and J. Araújo, "The evolution of requirements practices in software startups," *Proceedings - International Conference on Software Engineering*, pp. 823–833, 2018.

[2] P. M. Khan and M. M. Beg, "Extended decision support matrix for selection of sdlc-models on traditional and agile software development projects," *International Conference on Advanced Computing and Communication Technologies, ACCT*, pp. 8–15, 2013.

[3] N. Al-Saiyd and E. Zriqat, "Analyzing the Impact of Requirement Changing on Software Design," *European Journal of Scientific Research*, vol. 136, no. February, 2015.

[4] DOJ, "DOJ Systems Development Life Cycle Guidance Chapter 1," 2003. [Online]. Available: https://www.justice.gov/archive/jmd/irm/lifecycle/ch1.htm

[5] C. Ackermann, M. Lindvall, and G. Dennis, "Redesign for flexibility and maintainability: a case study," *2009 13th European Conference on Software Maintenance and Reengineering*, pp. 259–262, mar 2009. [Online]. Available: https://ieeexplore.ieee.org/document/4812763/

[6] D. Reimanis and C. Izurieta, "Towards Assessing the Technical Debt of Undesired Software Behaviors in Design Patterns," *Proceedings - 2016 IEEE 8th International Workshop on Managing Technical Debt, MTD 2016*, pp. 24–27, 2016.

[7] M. De Leon-Sigg, S. Vazquez-Reyes, and D. Rodriguez-Avila, "Towards the use of a framework to make technical debt visible," *Proceedings - 2020 8th Edition of the International Conference in Software Engineering Research and Innovation, CONISOFT 2020*, pp. 86–92, 2020.

[8] W. Snipes, S. L. Karlekar, and R. Mo, "A case study of the effects of architecture debt on software evolution effort," *Proceedings - 44th Euromicro Conference on Software Engineering and Advanced Applications, SEAA 2018*, pp. 400–403, 2018.

[9] M. Wiese, M. Riebisch, and J. Schwarze, "Preventing Technical Debt by Technical Debt Aware Project Management," *Proceedings - 2021 IEEE/ACM International Conference on Technical Debt, TechDebt 2021*, pp. 84–93, 2021.

[10] E. E. Ogheneovo, "On the Relationship between Software Complexity and Maintenance Costs," *Journal of Computer and Communications*, vol. 02, no. 14, pp. 1–16, 2014.

[11] L. Tang, Y. G. Mei, and J. J. Ding, "Metric-based tracking management in software maintenance," *2nd International Workshop on Education Technology and Computer Science, ETCS 2010*, vol. 1, pp. 675–678, 2010.

[12] T. F. Thamburaj and A. Aloysius, "Models for Maintenance Effort Prediction with Object-Oriented Cognitive Complexity Metrics," in *2017 World Congress on*

Computing and Communication Technologies (WCCCT). IEEE, feb 2017, pp. 191–194. [Online]. Available: http://ieeexplore.ieee.org/document/8074523/

[13] H. M. Sneed, "A cost model for software maintenance & evolution," *IEEE International Conference on Software Maintenance, ICSM*, pp. 264–273, 2004.

[14] D. Port and B. Taber, "Actionable Analytics for Strategic Maintenance of Critical Software: An Industry Experience Report," *IEEE Software*, vol. 35, no. 1, pp. 58–63, 2017.

[15] S. Mamone, "The IEEE standard for software maintenance," *ACM SIGSOFT Software Engineering Notes*, vol. 19, no. 1, pp. 75–76, jan 1994. [Online]. Available: https://dl.acm.org/doi/10.1145/181610.181623

[16] R. Hasan, S. Chakraborty, and J. Dehlinger, "Examining software maintenance processes in small organizations: Findings from a case study," *Studies in Computational Intelligence*, vol. 377, pp. 129–143, 2012.

[17] N. Niu, S. Brinkkemper, X. Franch, J. Partanen, and J. Savolainen, "Requirements engineering and continuous deployment," *IEEE Software*, vol. 35, no. 2, pp. 86–90, 2018.

[18] M. Galster, C. Treude, and K. Blincoe, "Supporting Software Architecture Maintenance by Providing Task-Specific Recommendations," *Proceedings - 2019 IEEE International Conference on Software Maintenance and Evolution, ICSME 2019*, pp. 370–372, 2019.

[19] L. Ping, "A quantitative approach to software maintainability prediction," *Proceedings - 2010 International Forum on Information Technology and Applications, IFITA 2010*, vol. 1, pp. 105–108, 2010.

[20] U. Kumar, D. Galar, A. Parida, C. Stenström, and L. Berges, "Maintenance performance metrics: a state-of-the-art review," *Journal of Quality in Maintenance Engineering*, vol. 19, no. 3, pp. 233–277, aug 2013. [Online]. Available: https://www.emerald.com/insight/content/doi/10.1108/JQME-05-2013-0029/full/html

[21] S. M. A. Shah, M. Morisio, and M. Torchiano, "An overview of software defect density: A scoping study," *Proceedings - Asia-Pacific Software Engineering Conference, APSEC*, vol. 1, pp. 406–415, 2012.

[22] M. Alenezi and M. Zarour, "Does Software Structures Quality Improve over Software Evolution ? Evidences from Open - Source Projects," *Special issue on "Computing Applications and Data Mining" International Journal of Computer Science and Information Security (IJCSIS*, vol. 14, no. 1, pp. 61–75, 2016.

[23] C. Rahmani and D. Khazanchi, "A study on defect density of open source software," *Proceedings - 9th IEEE/ACIS International Conference on Computer and Information Science, ICIS 2010*, pp. 679–683, 2010.

[24] "[ARCHIVED] Finding your way around SourceForge." [Online]. Available: http://oss-watch.ac.uk/resources/archived/sfintro

[25] V. Lenarduzzi, A. Sillitti, and D. Taibi, "Analyzing Forty years of software maintenance models," *Proceedings - 2017 IEEE/ACM 39th International Conference on Software Engineering Companion, ICSE-C 2017*, pp. 146–148, 2017.

[26] D. Garlan, "Software Architecture: a Roadmap David Garlan," *Design*, 1999.

[27] L. R. Vijayasarathy and C. W. Butler, "Choice of Software Development Methodologies: Do Organizational, Project, and Team Characteristics Matter?" *IEEE Software*, vol. 33, no. 5, pp. 86–94, 2016.

[28] Y. Ren, X. Tao, Z. Liu, and X. Chen, "Software maintenance process model and contrastive analysis," *Proceedings - 2011 4th International Conference on Information Management, Innovation Management and Industrial Engineering, ICIII 2011*, vol. 3, pp. 169–172, 2011.

[29] J. Araujo, C. Melo, F. Oliveira, P. Pereira, and R. Matos, "A Software Maintenance Methodology: An Approach Applied to Software Aging," *15th Annual IEEE International Systems Conference, SysCon 2021 - Proceedings*, 2021.

[30] M. Cinque, D. Cotroneo, and A. Pecchia, "Event logs for the analysis of software failures: A rule-based approach," *IEEE Transactions on Software Engineering*, vol. 39, no. 6, pp. 806–821, 2013.

[31] G. Rong, S. Gu, H. Zhang, D. Shao, and W. Liu, "How is logging practice implemented in open source software projects? A preliminary exploration," *Proceedings - 25th Australasian Software Engineering Conference, ASWEC 2018*, pp. 171–180, 2018.

[32] S. Levin and A. Yehudai, "Visually exploring software maintenance activities," *Proceedings - 7th IEEE Working Conference on Software Visualization, VISSOFT 2019*, pp. 110–114, 2019.

[33] J. Zhu, S. He, J. Liu, P. He, Q. Xie, Z. Zheng, and M. R. Lyu, "Tools and Benchmarks for Automated Log Parsing," *Proceedings - 2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice, ICSE-SEIP 2019*, pp. 121–130, 2019.

[34] F. Baccanico, G. Carrozza, M. Cinque, D. Cotroneo, A. Pecchia, and A. Savignano, "Event Logging in an Industrial Development Process: Practices and Reengineering Challenges," in *2014 IEEE International Symposium on Software Reliability Engineering Workshops*, no. i. IEEE, nov 2014, pp. 10–13. [Online]. Available: https://ieeexplore.ieee.org/document/6983789

[35] A. Pecchia, M. Cinque, G. Carrozza, and D. Cotroneo, "Industry Practices and Event Logging: Assessment of a Critical Software Development Process," *Proceedings - International Conference on Software Engineering*, vol. 2, pp. 169–178, 2015.

[36] G. Rong, Q. Zhang, X. Liu, and S. Gu, "A Systematic Review of Logging Practice in Software Engineering," *Proceedings - Asia-Pacific Software Engineering Conference, APSEC*, vol. 2017-Decem, pp. 534–539, 2018.

[37] N. Gurumdimma, A. Jhumka, M. Liakata, E. Chuah, and J. Browne, "CRUDE: Combining Resource Usage Data and Error Logs for Accurate Error Detection in Large-Scale Distributed Systems," *Proceedings of the IEEE Symposium on Reliable Distributed Systems*, pp. 51–60, 2016.

[38] J. Dwyer and T. M. Truta, "Finding anomalies in windows event logs using standard deviation," *Proceedings of the 9th IEEE International Conference on Collaborative Com-*

puting: Networking, Applications and Worksharing, COLLABORATECOM 2013, pp. 563–570, 2013.

[39] D. Evangelin Geetha, T. V. Suresh Kumar, and K. Rajani Kanth, "Predicting performance of software systems during feasibility study of software project management," *2007 6th International Conference on Information, Communications and Signal Processing, ICICS*, pp. 1–5, 2007.

[40] W. Song, X. Xia, H. A. Jacobsen, P. Zhang, and H. Hu, "Efficient Alignment Between Event Logs and Process Models," *IEEE Transactions on Services Computing*, vol. 10, no. 1, pp. 136–149, 2017.

[41] A. C. Pathan and M. A. Potey, "Detection of malicious transaction in database using log mining approach," *Proceedings - International Conference on Electronic Systems, Signal Processing, and Computing Technologies, ICESC 2014*, pp. 262–265, 2014.

[42] B. Kitchenham and S. Charters, "Guidelines for performing Systematic Literature Reviews in Software Engineering (Software Engineering Group, Department of Computer Science, Keele . . . ," Keele University, Tech. Rep. January, 2007.

[43] J. Zhu, P. He, Q. Fu, H. Zhang, M. R. Lyu, and D. Zhang, "Learning to log: Helping developers make informed logging decisions," *Proceedings - International Conference on Software Engineering*, vol. 1, pp. 415–425, 2015.

[44] M. O. Kherbouche, N. Laga, and P. A. Masse, "Towards a better assessment of event logs quality," *2016 IEEE Symposium Series on Computational Intelligence, SSCI 2016*, 2017.

[45] Z. Stojanov, J. Stojanov, D. Dobrilovic, and N. Petrov, "Trends in software maintenance tasks distribution among programmers: A study in a micro software company," *SISY 2017 - IEEE 15th International Symposium on Intelligent Systems and Informatics, Proceedings*, pp. 23–27, 2017.

[46] S. Al-Fedaghi and F. Mahdi, "Events Classification in Log Audit," *International journal of Network Security & Its Applications*, vol. 2, no. 2, pp. 58–73, 2010.

[47] M. J. Jans, M. G. Alles, and M. A. Vasarhelyi, "Process Mining of Event Logs in Auditing: Opportunities and Challenges," *SSRN Electronic Journal*, no. August 2020, 2012.

[48] W. Van Der Aalst, T. Weijters, and L. Maruster, "Workflow mining: Discovering process models from event logs," *IEEE Transactions on Knowledge and Data Engineering*, vol. 16, no. 9, pp. 1128–1142, 2004.

[49] T. Jia, Y. Li, C. Zhang, W. Xia, J. Jiang, and Y. Liu, "Machine Deserves Better Logging: A Log Enhancement Approach for Automatic Fault Diagnosis," *Proceedings - 29th IEEE International Symposium on Software Reliability Engineering Workshops, ISSREW 2018*, pp. 106–111, 2018.

[50] Y. A. Bekeneva, "Algorithm for Generating Event Logs Based on Data from Heterogeneous Sources," in *2020 IEEE Conference of Russian Young Researchers in Electrical and Electronic Engineering (EIConRus)*. IEEE, jan 2020, pp. 233–236. [Online]. Available: https://ieeexplore.ieee.org/document/9039350/

[51] G. Rong, Y. Xu, S. Gu, H. Zhang, and D. Shao, "Can You Capture Information As You Intend To? A Case Study on Logging Practice in Industry," in *Proceedings - 2020 IEEE International Conference on Software Maintenance and Evolution, ICSME 2020*. Institute of Electrical and Electronics Engineers Inc., sep 2020, pp. 12–22.

[52] K. Slaninová, "User behavioural patterns and reduced user profiles extracted from log files," *International Conference on Intelligent Systems Design and Applications, ISDA*, pp. 289–294, 2014.

[53] P. Dhanalakshmi, K. Ramani, and B. E. Reddy, "The Research of Preprocessing and Pattern Discovery Techniques on Web Log Files," *Proceedings - 6th International Advanced Computing Conference, IACC 2016*, pp. 139–145, 2016.

[54] G. Kocsis and P. Ekler, "Analyzing the resource requirements of usage statistics gathering on online newspapers," *CINTI 2012 - 13th IEEE International Symposium on Computational Intelligence and Informatics, Proceedings*, pp. 213–218, 2012.

[55] M. Waqar and D. Rafiei, "Tracking User Activities and Marketplace Dynamics in Classified Ads," *Proceedings - 2016 IEEE/WIC/ACM International Conference on Web Intelligence, WI 2016*, pp. 522–525, 2017.

[56] G. Paliouras, C. Papatheodorou, V. Karkaletsis, C. Spyropoulos, and P. Tzitziras, "From Web usage statistics to Web usage analysis," *Proceedings of the IEEE International Conference on Systems, Man and Cybernetics*, vol. 2, pp. 159–164, 1999.

[57] A. Hasiloglu and A. Bali, "Central audit logging mechanism in personal data web services," *6th International Symposium on Digital Forensic and Security, ISDFS 2018 - Proceeding*, vol. 2018-Janua, pp. 1–3, 2018.

[58] H. Wang and J. Yang, "Research and application of web development based on ASP.NET 2.0+Ajax," *2008 3rd IEEE Conference on Industrial Electronics and Applications, ICIEA 2008*, pp. 857–860, 2008.

[59] P. R. Anish, B. Balasubramaniam, J. Cleland-Huang, R. Wieringa, M. Daneva, and S. Ghaisas, "Identifying Architecturally Significant Functional Requirements," *Proceedings - 5th International Workshop on the Twin Peaks of Requirements and Architecture, TwinPeaks 2015*, pp. 3–8, 2015.

[60] IBM, "HTTP requests - IBM Documentation," 2021. [Online]. Available: https://www.ibm.com/docs/en/cics-ts/5.3?topic=protocol-http-requests

[61] Mozilla, "Getting Started - Developer guides — MDN," 2022. [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/Guide/AJAX/Getting{_}Started

[62] M. Jailia, A. Kumar, M. Agarwal, and I. Sinha, "Behavior of MVC (Model View Controller) based Web Application developed in PHP and .NET framework," in *2016 International Conference on ICT in Business Industry & Government (ICTBIG)*. IEEE, 2016, pp. 1–5. [Online]. Available: http://ieeexplore.ieee.org/document/7892651/

[63] M. X. Gu and K. Tang, "Comparative analysis of WebForms MVC and MVP architecture," *2010 2nd Conference on Environmental Science and Information Application Technology, ESIAT 2010*, vol. 2, pp. 391–394, 2010.

[64] IBM, “HTTP responses - IBM Documentation,” 2021. [Online]. Available: https://www.ibm.com/docs/en/cics-ts/5.3?topic=protocol-http-responses

[65] Mozilla, “Request header - MDN Web Docs Glossary: Definitions of Web-related terms — MDN,” 2022. [Online]. Available: https://developer.mozilla.org/en-US/docs/Glossary/Request{_}header

[66] API.jQuery, “jQuery.ajax() — jQuery API Documentation,” 2022. [Online]. Available: https://api.jquery.com/jquery.ajax/

[67] JavaScript.Info, “Bubbling and capturing.” [Online]. Available: https://javascript.info/bubbling-and-capturing{#}capturing

[68] R. Harkness, M. Crook, and D. Povey, “Programming Review of Visual Basic.NET for the Laboratory Automation Industry,” *Journal of Laboratory Automation*, vol. 12, no. 1, pp. 25–32, 2007.

Appendix A

Logging practice in software engineering

Providing a guide for software engineers and developers to implement a suitable logging implementation in their software systems has to prove to be a vital tool in both industrial use and progress of academia [31]. Guoping Rong et al. made a study to review these logging practices published papers to improve the performance and efficiency of logging implementation. From his study he made selection criteria to include (as in Table A.1) and exclude (as in Table A.2) academic papers about logging practices [31, 36].

The Rong's selection criteria obtained numerous research papers of logging practices applied in the industry by either creating a new logging mechanism or optimising existing logging mechanisms. By reviewing 41 identified papers he found that many practitioners and researchers recognise the importance of logging practice in software engineering. There is a lack of guidance to provide software engineers or developers to create or improve their efficient logging mechanisms [31, 43].

Table A.1: *G. Rong's inclusion selection criteria [31]*

Identification	**Criteria**
I1.	Publications that investigate the methodology for logging practice.
I2.	Publications that investigate the tools, frameworks, systems which support logging practice.
I3.	Publications that propose a standard for logging practice.
I4.	Publications that are peer-reviewed (conference paper, journal article).
I5.	Publications that are primary studies on logging practice.

Table A.2: *G. Rong's exclusion selection criteria [31]*

Identification	**Criteria**
E1.	Publications that investigate log analysis.
E2.	Publications that investigate the usage of logs.
E3.	Publications that investigate the technologies on logging user behaviours.
E4.	Publications that are not written in English.
E5.	Additionally, short papers, demo or industry publications are excluded.

In Figure A.1 shows the distribution of the 41 published papers obtained for Rong's research relating to logging practices. Event logging has an increasingly important role in modern software systems, therefore the research focus on logging practices in software engineering have been on a rise between 1990 and 2017.

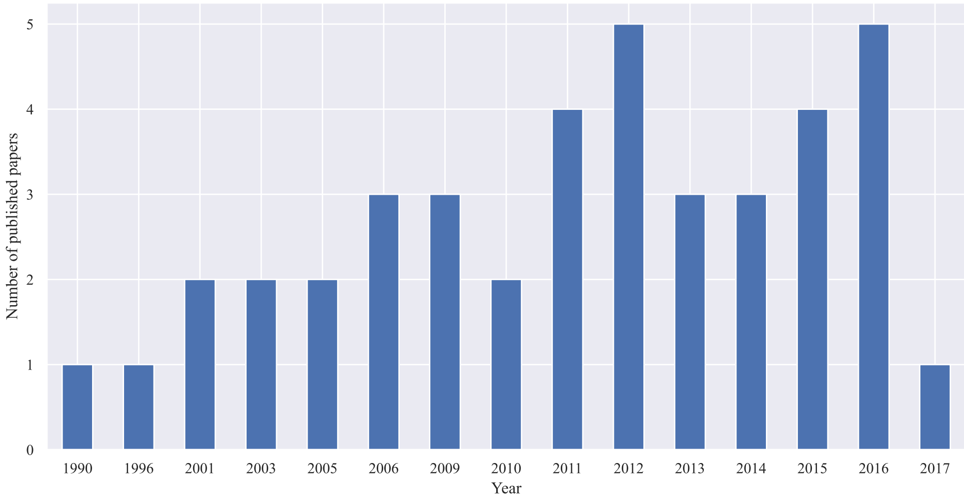

Figure A.1: *The distribution of the papers' published years [31]*