A Systematic Literature Review of Machine Learning Techniques for Software Maintainability Prediction

Hadeel Alsolai
1Computer Science and Information system
Princess Nourah Bint Abdulrahman University
Riyadh, Saudi Arabia
2Computer and Information Sciences
University of Strathclyde
Glasgow, United Kingdom
Hadeel.alsolai@strath.ac.uk

Marc Roper
Computer and Information Sciences
University of Strathclyde
Glasgow, United Kingdom
Marc.roper@strath.ac.uk

Abstract

Context: Software maintainability is one of the fundamental quality attributes of software engineering. The accurate prediction of software maintainability is a significant challenge for the effective management of the software maintenance process. **Objective:** The major aim of this paper is to present a systematic review of studies related to the prediction of maintainability of object-oriented software systems using machine learning techniques. This review identifies and investigates a number of research questions to comprehensively summarize, analyse and discuss various viewpoints concerning software maintainability measurements, metrics, datasets, evaluation measures, individual models and ensemble models. **Method:** The review uses the standard systematic literature review method applied to the most common computer science digital database libraries from January 1991 to July 2018. **Results:** We survey 56 relevant studies in 35 journals and 21 conference proceedings. The results indicate that there is relatively little activity in the area of software maintainability prediction compared with other software quality attributes. CHANGE maintenance effort and the maintainability index were the most commonly used software measurements (dependent variables) employed in the selected primary studies, and most made use of class-level product metrics as the independent variables. Several private datasets were used in the selected studies, and there is a growing demand to publish datasets publicly. Most studies focused on regression problems and performed k-fold cross-validation. Individual prediction models were employed in the majority of studies, while ensemble models relatively rarely. **Conclusion:** Based on the findings obtained in this systematic literature review, ensemble models demonstrated increased accuracy prediction over individual models, and have been shown to be useful models in predicting software maintainability. However, their application is relatively rare and there is a need to apply these, and other, models to an extensive variety of datasets with the aim of improving the accuracy and consistency of results.

KEYWORDS: Systematic literature review, software maintainability prediction, machine learning, metric, dataset.

1 Introduction

Software quality is an essential ingredient for software success. Quality is affected mainly by its attributes, which are divided into two groups: external attributes and internal attributes. Internal attributes, such as class cohesion, are measured directly from the software, while external attributes, such as maintainability, need to be measured indirectly, and their prediction often relies on internal attributes [1]. Software maintainability is one of the fundamental external quality attributes and is recognised as a research area of primary concern in software engineering.

Prediction is a core part of estimation, which is a crucial aspect of project planning [2] and involves the determination of a number of factors including duration, staff, size, costs, and effort [3]. Prediction depends mainly on historical internal and external quality attributes from completed projects. The correlation between internal attributes, "independent variables", and external attributes, "dependent variables", is a recognised part of software maintainability prediction [1]. Software maintainability prediction models have been investigated to assist organisations to utilise cost, allocate resource as well as obtain an accurate management plan and effective maintenance process [4]. However, prediction of software maintainability is a challenging task and requires accurate prediction models [3].

Object-Oriented (OO) software systems are now widely developed by and deployed within organisations and object-orientation has emerged as the dominant software development paradigm. OO programming languages, such as Java, C++, C# and software development tools, such as Unified Modeling Language (UML) are also widely used in these organisations [5], and consequently they share significant concerns regarding the effective maintenance of these OO systems.

This paper reports on a systematic literature review (SLR) of relevant journals and conference proceedings papers that focus on the topic of software maintainability prediction. This review investigates a set of research questions (RQs) to comprehensively summarize, analyse and discuss various viewpoints: software maintainability measurements, metrics, datasets, evaluation measures, standalone models and ensemble models. Our search was focused on the most common computer science digital database libraries between 1991 until 2018 and we survey 56 relevant studies in 35 journals and 21 conferences proceedings.

The primary objective of this review is to investigate the current state of software maintainability prediction to and discover the progress made, limitations and challenges, along with future research requirements. To the best of our knowledge, this is the first SLR of software maintainability prediction for OO systems that comprehensively evaluates a wide variety of important journal and conference proceedings with respect to specific defined research question. Our review differs from the previous review studies [6-11] because it includes a higher number of relevant journal and conference proceedings in the software maintainability field. Furthermore, we also apply a different analysis on the selected primary studies and provide more additional detailed analysis of each paper. Previous review studies focused on non-object-oriented systems [9-11], or considered a limited number of studies [6], or concentrated on a single aspect such as the measurement of software maintainability, or the models or the metrics used [7, 8]. In contrast, this study is classifies the concept of software maintainability according to three dimensions: the measurement of maintainability (dependent variable), the metrics considered (independent variables) and the models employed. This study

has applied the research method for conducting a SLR proposed by Kitchenham [12], and has analysed comprehensively each selected study. Therefore, we have confidence our SLR is both novel and hope that software engineering researchers and practitioners will find it to be a useful resource. The key contributions of this paper are:

- We present a SLR of the work in the field of software maintainability prediction using machine learning techniques.
- We analyse and compare 56 relevant studies according to several research questions.
- We determine gaps and identify further research opportunities in this field.

The structure of this paper is organised as follows: Section 2 introduces the research method used to conduct this literature review. Section 3 provides the results of the SLR. Section 4 discusses our major research questions (RQs). Section 5 concludes this study with limitations and research gaps and formulate directions for further study.

2 Method

The SLR is a commonly and widely applied method in the software engineering field [12]. The review presented here aims to identify, evaluate and interpret all available research relevant to predicting software maintainability using machine learning models. This SLR is based on the quality reporting guidelines as proposed by Kitchenham for performing a SLR in software engineering [12], and also takes on board subsequent lessons learned and advice [13]. Three primary stages are established and adjusted to include appropriate steps, namely, planning, conducting and reporting the review. The planning stage involves the following steps: determining the needs for a systematic review, which was discussed in the introduction; evolving an appropriate review protocol to eliminate the possibility of researcher bias. The conducting stage involves the following steps: formulating RQs to focus on the central issues of this review; developing the search process to conduct search activities; identifying selection criteria to select appropriate studies; examining quality assessment to evaluate selected studies in terms of quality; applying data extraction to document the information obtained from the studies; performing data synthesis to accumulate and summarise the results. The final reporting stage involves only one step: presenting results and discussions to answer each research question. This process is illustrated in Figure 1.

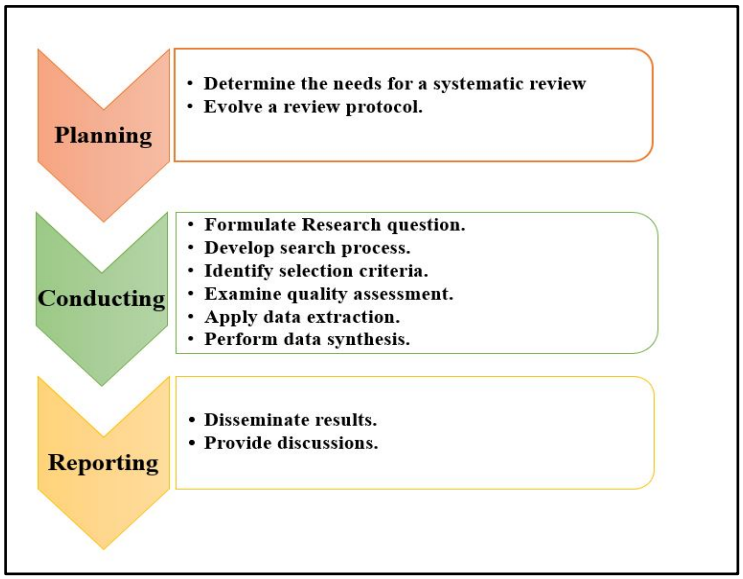

Figure 1: The framework of Systematic Literature Review.

2.1 Review Protocol

The review protocol aims to direct the implementation of the review and minimise the possibility of researcher bias. The critical elements of our review protocol include RQs, the search process, inclusion and exclusion criteria, quality assessment, data extraction and finally data synthesis. Furthermore, the review protocol was iteratively developed and evaluated during the conducting and reporting stages. Details of the protocol are explained in the following sections (2.2-2.6).

2.2 Research Questions

The RQs were introduced to specify the research boundaries. They were formulated with the assistance of the (PICOC) criteria [12] which recognize RQs from four viewpoints as follows:

- **Population:** OO system, software system, application software, software project.
- **Intervention:** Software maintainability prediction, predicting software maintenance effort, change proneness, techniques, methods, models, process and product metrics, dataset.
- **Comparison:** N/A.
- **Outcomes:** Accuracy prediction of software maintainability, building good prediction models.
- **Context:** empirical or experimental studies in academia and industry, large and small size of the datasets.

The primary objective of this SLR is to collect and analyse appropriate evidence to answer our RQs. Our motivation is to answer a set of seven RQs to obtain insights into significant aspects of our research direction, including advancing our knowledge of software maintainability prediction for OO systems and identifying the limitations of research so as to define further research directions. The RQs and their motivation are documented in Table 1 below.

Table 1: Research question.

ID	Research question	Main motivation
RQ1	What are the definitions of software maintainability?	Identify different software maintainability definitions.
RQ1.1	How can the software maintainability be measured (dependent variable)?	Recognize the software maintainability measurements.
RQ2	What type of OO metrics have been proposed to measure software maintainability?	Identify OO proposed metrics that are commonly being used in software maintainability.
RQ2.1	What are the metrics used (independent variable)?	Determine various OO metrics.
RQ2.2	What is the level (e.g. class level, method level) of these metrics?	Identify the level of OO metrics.
RQ3	What software maintainability datasets in the literature have been used to build prediction models?	Determine various datasets commonly being used in the the software maintainability domain.
RQ3.1	What are the types of software maintainability datasets (e.g. public datasets, private datasets)?	Recognize the type of these datasets.
RQ3.2	What tools are used to extract metrics from open source projects?	Identify different tools to extract OO metrics.
RQ3.3	What software programming languages are used to write system code?	Determine various software programming languages commonly being used to collect OO metrics.
RQ3.4	What are the number of classes in the software system?	Identify the number of classes in the software system
RQ4	What are the evaluation measures used to assess the performance of software maintainability prediction models?	Explore evaluation measures commonly being used in each software maintainability datasets.
RQ4.1	What approach (e.g. cross-validation, holdout) is used to evaluate the performance of software maintainability prediction models?	Identify different validation approaches applied on software maintainability prediction models.
RQ5	What type of machine learning problem (e.g. classification, regression) software maintainability fall into?	Identify the type of machine learning problem.
RQ5.1	What are the categories of machine learning problem?	Determine various categories of machine learning problem.
RQ6	What are the individual prediction models (e.g. neural network, linear regression) used to predict software maintainability?	Investigate the individual prediction models commonly being used in software maintainability.
RQ6.1	What are the best performing individual prediction models?	Identify the best performing individual prediction models in each study.
RQ7	What ensemble prediction models (e.g. bagging, boosting) are used to predict software maintainability?	Investigate the ensemble prediction models commonly being used in software maintainability.
RQ7.1	What type of ensemble prediction models were performed to predict software maintainability?	Determine different type of ensemble prediction models.
RQ7.2	Do the ensemble models outperform the individual prediction models?	Investigate whether ensemble models represent an improvement over the performance over the individual prediction models.

2.3 Search Process

The search process must be focused in a way that allows the identified RQs to be accurately investigated and includes four steps: choosing the digital libraries, identifying additional search sources, selecting the interval time of the published articles, and defining search keywords. The search was applied on five of the most popular and largest computer science online digital libraries that publish peer-reviewed articles:

- IEEE eXplore (ieeexplore.ieee.org)
- ACM Digital Library (dl.acm.org)
- Springer (springerlink.com)
- Elsevier (sciencedirect.com)
- Wiley online library (onlinelibrary.wiley.com)

Furthermore, manual research was applied to include relevant journal and conference proceedings in the software maintainability field. We selected these journals and conferences particularly since they involve empirical studies or literature reviews and they are well-established and highly relevant software engineering publications. The selected journals and conferences are presented in Table 2. The search was limited to articles published in the interval from 1991 to 2018. We restricted the search in this time interval since machine learning started to be applied to problems of this nature in the 1990s [14] and investigations into software maintenance began in earnest in1985 [6]. Furthermore, research into software maintainability expanded dramatically with the usage of OO systems in 1990s [15] and no studies relevant to the identified RQs were found to exist before these dates.

Table 2: Selected journals and conference proceedings.

Source	Acronym	Number of Studies	Published by	Impact factor on 5 years	Quarter category
IEEE Transactions on Software Engineering	TSE	8	IEEE	3.92	Q 1
Empirical Software Engineering	EMSE	20	Springer	3.49	Q 1
Information and Software Technology	IST	11	Elsevier	2.76	Q 1
Journal of Systems and Software	JSS	14	Elsevier	2.40	Q 1
IEEE Software	IEEE SW	3	IEEE	2.50	Q 1
Soft Computing	SC	5	Elsevier	2.20	Q 2
Software Quality Journal	SQJ	6	Springer	1.90	Q 2
Journal of Software Maintenance and Evolution: Research and Practice	JSME	6	Wiley	1.21	Q 2
IET Software	IST	2	IEEE	0.97	Q 3

International Journal of System Assurance Engineering and Management	IJSAEM	4	Springer	0.94	Q3
ACM SIGSOFT Software Engineering	ASSE	3	ACM	0.45	Q4
Conferences				**H-index**	
International Conference on Software Maintenance and Evolution	ICSME	4	IEEE	29	
International Conference on Software Engineering	ICSE	7	IEEE	68	
International Conference on Predictive Models and Data Analytics in Software Engineering	PROMISE	1	ACM	NA	

We created a list of search strings by integrating appropriate synonyms and alternative terms with the Boolean operator (AND has the effect of narrowing and limiting the search, while OR serves to broaden and expand the search) and the truncation symbol (*) which is used to identify words with a particular beginning (for example predict* will match with predict, prediction predicting and predicted) [16].

The following search terms were formulated in this SLR: (software maintainability OR maintenance effort) AND (predict* OR forecast* OR estimate*) AND (machine learning OR data mining OR artificial intelligence OR application OR Bayesian network OR neural network OR Regression OR support vector machine) AND (method OR model OR technique) AND (software maintenance metric OR software maintenance measure*).

The role of machine learning techniques has emerged as a recommended technique in several research fields, and these have often proven to be better than other techniques (e.g. human evaluation or statistical methods) [17]. However, we selected some studies that do not use machine learning techniques because these studies answer some of our RQs. Nevertheless, this paper focuses on a systematic summarisation of machine learning techniques used in software maintainability prediction and collects the empirical evidence from employing these techniques.

The endnote system was used to store and organize search results and a spreadsheet was used to create a table of data extracted from each selected paper. The initial search applied the search terms on each selected database as well as the journal and conference proceedings to include the full document. This procedure returned thousands of irrelevant studies, so it was decided to limit the search on the document title and content type (a conference or journal publication). Several duplicate papers were found in these databases which were subsequently removed. Additional studies were determined by referring to the references of identified relevant studies. After collecting studies from the primary search, we selected the relevant studies by scanning the title and abstract. Further investigation was performed to determine appropriate studies by reading the full text. The candidate studies were selected if they meet our criteria in section 2.4. Finally, the selected studies were identified after applying the quality assessment criteria. The progress of the search process is presented in Figure 2 and shows the number of articles identified at each stage of the selection and filtering process. The steps

were iterated until final agreement was reached. The SLR was completed at the end of July 2018 and 56 suitable studies were finally identified.

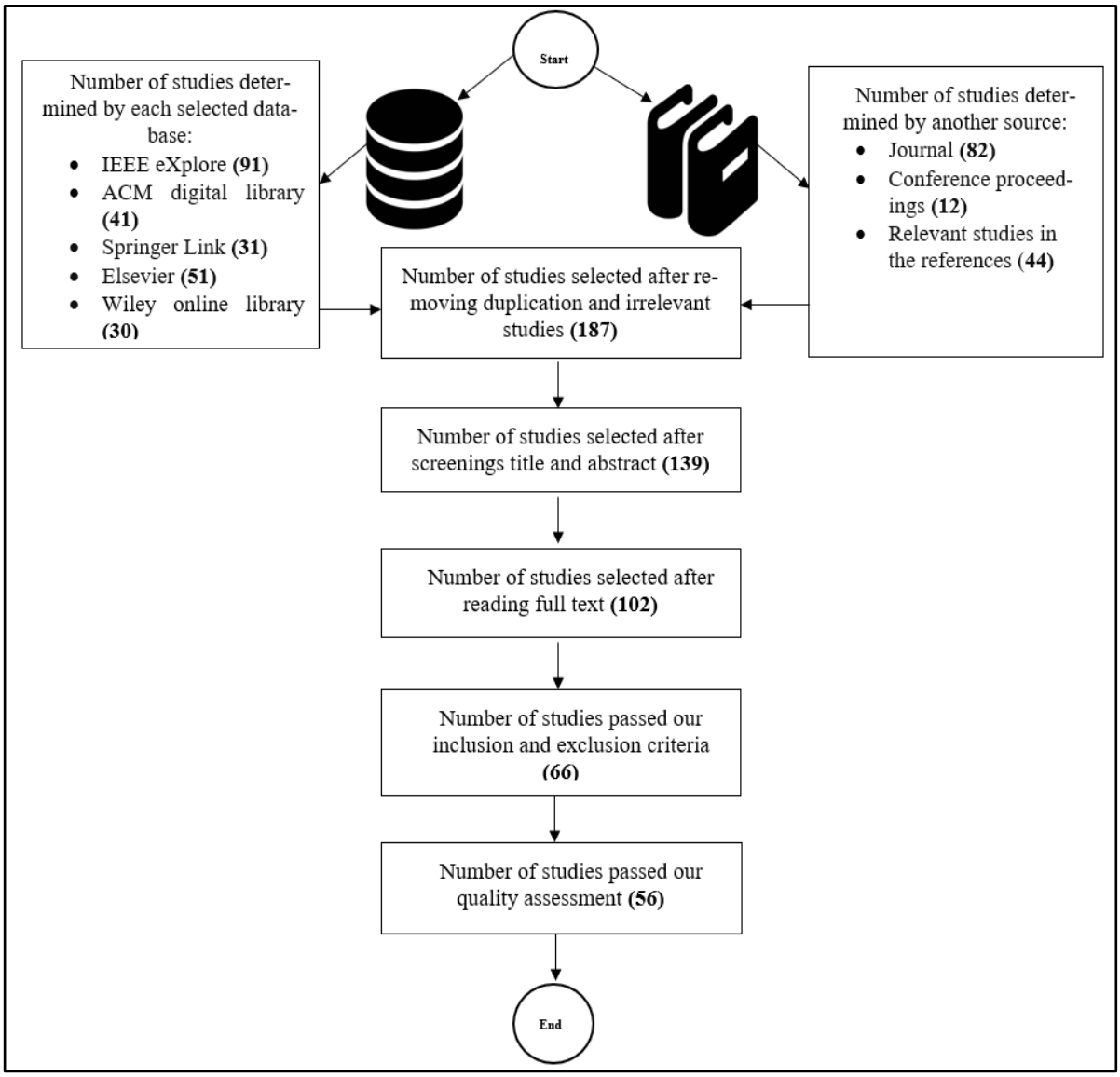

Figure 2: Process of primary studies selection.

2.4 Inclusion and exclusion criteria

The results from the previous steps yielded several irrelevant studies, so we defined inclusion and exclusion criteria in order to filer these out. The inclusion and exclusion criteria used in this SLR are outlined below:

1. Inclusion Criteria:
    - Studies focus on software maintainability prediction and answer any of our RQs.
    - Studies are applied on OO systems.

- Studies consider machine learning techniques.
- Studies are written in the English language.
- Studies are published in either a journal or conference proceedings.
- Studies are peer reviewed articles.
- Studies report on comparisons between model predictions.
- Studies are the most recent and comprehensive (in the case where studies were repeated).

2. Exclusion Criteria
    - Studies do not focus on software maintainability or answer our RQs.
    - Studies were applied on non- object-oriented systems, such as service-oriented, Aspect Oriented Software, web applications or functional systems.
    - Studies consider specific aspects of software maintainability, such as code smells, defects or fault proneness.
    - Studies are not written in English language.
    - Studies do not include the full text.
    - Studies fall outside our timeframe i.e. from 1991 to 2018.
    - Conference papers in the case where studies were published as both conference and journal versions.

Based on the criteria above, sixty-six studies were finally selected. Twenty-seven irrelevant studies were rejected that did not answer our RQs, five studies were rejected that focused on non-object-oriented systems, and four conference papers [18-21] were rejected because more recent journal versions of the work have been published.

2.5 Quality Assessment

The quality assessment stage is performed to evaluate each study identified in the previous step. The quality assessment follows the defined quality checklist as proposed by Kitchenham [12]. The main objective of the quality assessment is to evaluate studies and select studies that answer our RQs and to support more detailed analysis of inclusion and exclusion criteria. The quality assessment questions (QA) are specified below:

- QA1: Does the study define a main research objective or problem?
- QA2: Does the study define software maintainability?
- QA3: Does the study determine the type of software maintainability measurement (dependent variable)?
- QA4: Does the study employ OO software metrics (independent variables)?
- QA5: Does the study indicate the source of the datasets?
- QA6: Does the study use a suitable tool for the extraction of the datasets?

- QA7: Does the study identify the programing language of the systems being analysed?
- QA8: Does the study identify the number of classes in software system?
- QA9: Does the study make the dataset publicly available?
- QA10: Does the study use appropriate evaluation measures?
- QA11: Does the study use suitable cross validation techniques?
- QA12: Does the study justify the prediction techniques?
- QA13: Does the study apply prediction models and identify the best performing model?
- QA14: Does the study present the results and findings clearly?
- QA15: Does the study identify research limitations or challenges?

The scoring procedure of the quality assessment questions is constructed as a following:

- 1 represents Yes.
- 0.5 represents Partly.
- 0 represents No.

The scores rank the papers into four categories: excellent ($13.5 \leq$ score ≤ 15), good ($9.5 \leq$ score ≤ 13), fair ($5 \leq$ score ≤ 9), and fail ($0 \leq$ score ≤ 4.5). From applying the above quality assessment criteria, ten studies fail in our quality assessment. Finally, fifty-six primary studies were selected to conduct this SLR.

2.6 Data Extraction

The data extraction step is performed to extract data from each selected primary study with the aim of collecting data that answers our research question. Seven main properties were classified in Table 3 with respect to our research question requirements.

Table 3: Data Extraction Properties with their research question.

Properties	Research question
Software maintainability measurement	RQ1, RQ1.1
Software maintainability metrics.	RQ2, RQ2.1, RQ2.2
Software maintainability datasets.	RQ3, RQ3.1, RQ3.2, RQ3.3, RQ3.4
Software maintainability evaluation measures	RQ4, RQ 4.1
Machine learning problem to predict software maintainability.	RQ5, RQ5.1
Software maintainability individual models.	RQ6, RQ6.1
Software maintainability ensemble models.	RQ7, RQ7.1, RQ7.2

2.7 Data Synthesis

Data synthesis is applied to extract both quantitative data and qualitative data that forms a body of evidence from the selected studies that address issues related to our research question. The results are presented in the form of tables, pie charts, clustered bar charts, scatter charts and bar charts. These visualisations enable us to conduct a comparative analysis between selected studies and improve the quality of presentation.

3 Results

This section summarises the results obtained from selected primary studies and includes details about the search results, a visualisation of publication years and sources, and following on from this an overview of the quality assessment outcomes.

3.1 Selected primary studies

In this SLR, fifty-six primary studies were selected to compare and evaluate the studies in the software maintainability prediction domain, and are summarised in Table 17 **(**Appendix A)**.** This table provides a brief description of each selected study and contains the following attributes: study ID, reference, the title of the publication, the authors of the articles (first author and co-author), the publication year, place published, publication name and publication type (journal or conference).

3.2 Publications years

The publication years of selected primary studies are between the year 1991 and 2018 and Figure 3 shows the numbers of studies published during these years. After 1993 Li and Henry provided the QUES and UIMS datasets as an appendix to their paper, which motivated researchers to investigate prediction techniques on this dataset. Moreover, there is an indication of increased publications after 2005 when the PROMISE repository was launched [22], and researchers started to make more use of public datasets. However, in this year most of the datasets in the PROMISE repository were for software defect prediction and none for software maintainability.

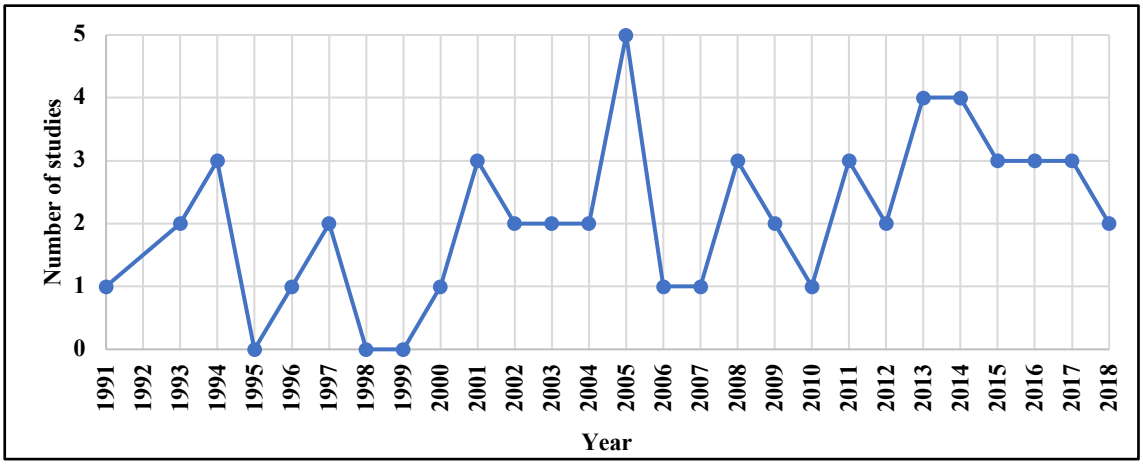

Figure 3: Number of selected studies over the years.

3.3 Publication sources

Of the 56 primary studies selected 35 appeared in journals and 21 in conferences (see Figure 4).

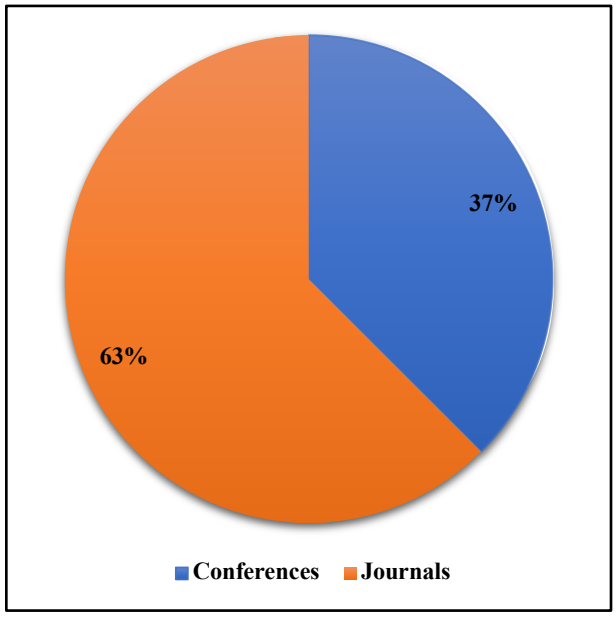

Figure 4: Distribution of publication type.

The most popular journal for papers associated with software maintainability prediction is the Journal of Systems and Software, followed by Information and Software Technology, then IEEE Transactions on Software Engineering. Figure 5 illustrates the number of selected primary studies in each journal.

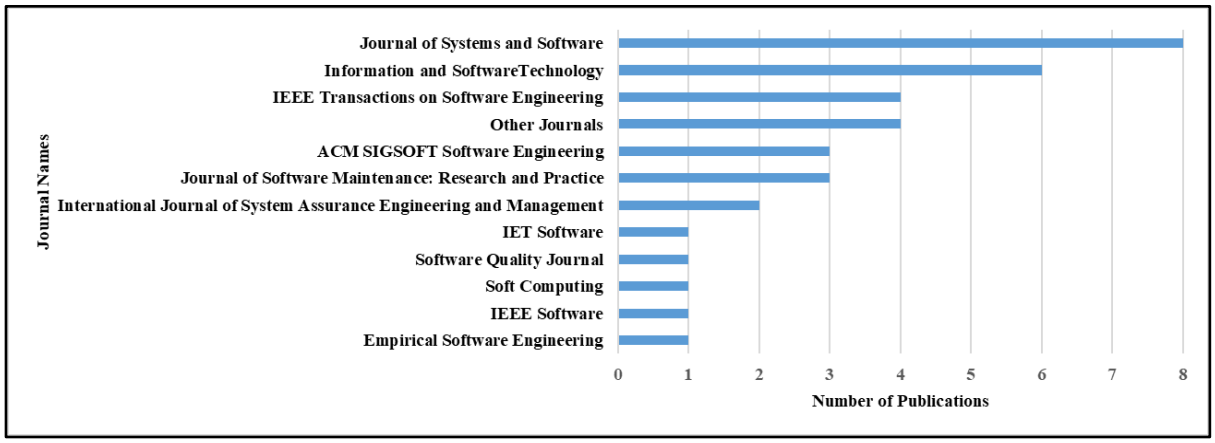

Figure 5: The number of studies in each journal.

Figure 6 shows the number of selected primary studies grouped by place of publication (i.e. digital library database). It can be seen that the most selected primary studies are chosen from the IEEE digital library, followed by Elsevier.

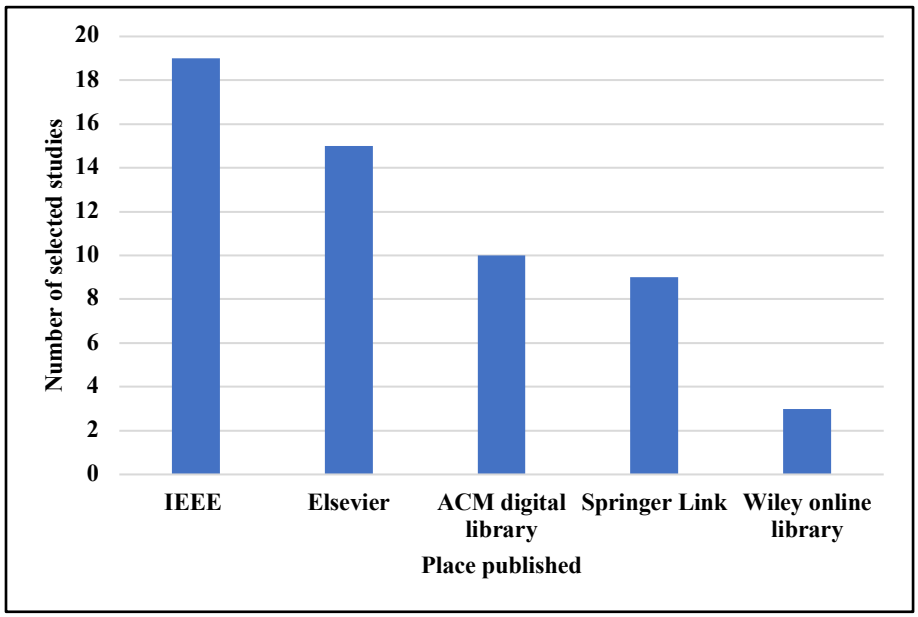

Figure 6: The number of selected studies in each digital library database

3.4 Quality Assessment result

We evaluated the selected primary studies according to the quality assessment questions described in section 2.5 and present the results of this analysis in Table 18 **(**Appendix A**).** This table shows that twelve of the selected studies obtained an excellent rating, thirty-three a good rating, followed by eleven which obtained a fair rating. However, we excluded ten studies that recorded a poor rating.

4 Discussion

This section aims to break down the analysis of the results based upon the RQs identified in Section 2.2. Each of the questions is considered in turn and the findings from all selected papers are considered to identify problems, solutions and to analyse the experimental results. Then, a comparison of the whole topic is conducted to comprehensively understand the topic and determine any limitations.

The first two questions consider software maintainability measurements (RQ1) and software maintainability metrics (RQ2) and the key differences between these are listed in Table 4 below.

Table 4: The differences between software maintainability measurements and metrics.

Software maintainability measurements	Software maintainability metrics
External attribute	Internal attribute
Dependent variable	Independent variable

Measured indirectly	Measured directly
Examines the software in an environment	Examines only the software
Difficult to extract	Easily extracted
Measured after or during execution	Measured without excution
Visible to the developer	Visible to the users

4.1 Software maintainability measurement

In this section, we discuss the findings in relation to the following RQs:

RQ1: What are the definitions of software maintainability? and **RQ1.1**: How can the software maintainability be measured (dependent variable)?

As software maintainability is not something that can be measured directly, our motivation was to gain insight into the different software maintainability definitions and measures that are being employed.

4.1.1 Software Maintainability Definitions

Software maintainability is defined in the IEEE Standard Glossary of Software Engineering Terminology [23] as "the ease with which a software system or component can be modified to correct faults, improve performance or other attributes, or adapt to a changed environment". This definition implies that software maintainability depends on various aspects of software modification (i.e. correction, improvement, adaptation or prevention). Moreover, maintainability is one substantial attribute proposed ([24]) of the set in the software quality model that involves: maintainability, efficiency, suitability, usability, reliability, security, portability and compatibility. However, several selected studies (e.g. S2, S16, S18) revealed that software maintainability is considered as one of the most challenging measurements of the software quality attributes, because there are several measurements and not all of them can be used to predict future activities. Some types of software maintainability (e.g. CHANGE metric, maintainability index and change proneness) can be used as an indirect measure (i.e. dependent variable) based on an extensive variety of metrics (i.e. independent variables) and machine learning prediction models can be applied to make a prediction. On the other hand, other types of software maintainability can be measured directly by observation during the maintenance process and record factors such as time, effort, or numbers of modules investigated. To explore these issues, we present different types of software maintenance measurements and which type can be used as dependent variables to make a prediction and capture an element of maintainability.

4.1.2 Software Maintainability Measurement

The key challenge with software maintainability measurement is that maintainability cannot be measured directly. Therefore, predictive models are based on indirect measures. Measuring software maintainability relies on a set of metrics that may be combined in a software maintainability function. The following equation is a general software maintainability function that performs to collect metrics from A_1 to A_n [25]:

$$M = F(A_1, A_2, \ldots, A_n) \quad (1)$$

The most obvious difficulty from the above function is to identify the appropriate metrics that can be measured from the source code directly. Various ways can be used to measure software maintainability, because there are different proxy definitons of maintainability. However, there is no any commonly agreed approach [25].

The selected primary studies for this SLR included nine types of software maintainability measurements (see Table 5). For each type of measurement, we present more details of the definition, a general form of the equation, value and interpretation of the value. Among these software maintainability measurements shown in Table 5, maintainability is measured by counting the number of changes made in the code. Li and Henry [26] define change metrics that capture the element of maintainability by counting the number of changes made in the source code, where the change could be insertion or deletion. This measurement is performed by several studies as shown in Table 5. Another measure of the software maintainability is based upon corrective maintenance. Lucia [3] calculated maintainability effort using four metrics that count the size and the number of tasks in the system under maintenance and are combined to produce the actual effort metric (dependent variable). Adaptive maintenance effort is used by three studies in Table 5 and is based on the effort (time) involved in adding, understanding and deleting in the source code in the system. The study by Oman and Hagemeister [27] proposes the maintainability index, a composite metric calculated from software complexity and quality metrics. Various selected primary studies have used this measurement as presented in Table 5. The change proneness maintainability measure is employed by two selected primary studies [28, 29] as a part of their experiment, and takes the form of a Boolean variable to indicate if a change is made during maintenance, while the independent variables are the Chidamber and Kemerer metrics [30]. Three selected primary studies compute time effort to perform maintenance tasks, while only one selected primary study calculated the cost of maintenance tasks based on the time consumed to execute these tasks. The main goal of these studies was to determine maintenance time or cost directly rather than construct a new formula or compare the current system with other systems. Some studies describe software maintainability rather than providing a formula. This description may be classified depending on software maintainability attributes or components. Finally, four selected primary studies used other software maintainability measurements. The explanation of the metrics used in the maintainability equation in Table 5 appears in section 4.2.

Table 5: Summary for software maintainability measurement.

Study ID	Type of maintenance	Maintainability defination	Maintainability equation	Measurement value	Maintainability Interpretation
S2, S20, S25, S26, S30, S32, S35, S36, S38, S39, S42, S43, S44, S45, S46, S47, S49, S50, S51, S52	Change maintenance effort	CHANGE, which is dependent metric, is computed according to the number of lines changed in a class where insertion or deletion are counted as 1, and modification of the contents is counted as 2	CHANGE = F (WMC, DIT, NOC, RFC, LCOM, MPC, DAC, NOM, SIZE1, SIZE2)	Integer	Maintainability is interpreted as being inversely proportional to the number of changes made: a high value of change indicates low maintainability, while a low number represents high maintainability.
S18	Corrective maintenance	Corrective maintenance effort computes the effort	Corrective maintenance effort= b1 NA +	Time (person-hours)	A high number of corrective maintenance effort indicates low maintainability, while a low

		spent on each phase of the corrective maintenance process (analyse, implement, produce, define, design).	b2 NB + b3 NC + b4 Size		number of corrective maintenance effort represents high maintainability.
S12, S19, S22	Adaptive maintenance effort	Adaptive maintenance effort computes the effort spent on each phase of the adaptive maintenance process (adding, understanding, deleting, modification, change).	Maintainability Effam= Eff_{add}+ Eff_{und}+ Eff_{del}.	Time (hours or months)	A high number of adaptive maintenance effort indicates low maintainability, while a low number of adaptive maintenance effort represents high maintainability.
S5, S6, S8, S21, S28, S29, S33, S48, S54, S55	Maintenance evaluation by maintainability index	The maintainability index, which is dependent metric, is a single value of a composite metric that compute the function of four metrics: lines of code, cyclomatic complexity, percentage line of comments and Halsted volume.	Maintainability Index $= 171 - 5.2\times \ln(HV) - 0.23* CC - 16.2\times \ln LOC + 50\times \sin\sqrt{2.4 \times COM}$	Decimal number between 0 and 1.	A number above 0.85 represents high maintainability, between 0.85 and 0.65 indicates medium maintainability, below 0.65 represents low maintainability

S47, S56	Maintenance evaluation by change proneness	Change proneness, which is dependent metric, is a Boolean variable indicating a change (addition, deletion or modification) has been made on a class.	Change Proneness = F (WMC, LCOM, CBO, DIT, RFC, NOC)
			IF (class change)
			Change proneness= TRUE
			ELSE
			Change proneness= FALSE
				Boolean Variable TRUE or FALSE	Maintainability change proneness is TRUE if the change occurs in class or FALSE if the change does not occur.

S7, S17, S37	Maintenance time	Compute the time to perform maintenance tasks (including understanding, developing, and modifying the program)	NA	Time (hours)	The greater the amount of time spent, the lower the maintainability.
S9	Maintenance cost	Compute three types of the costs: testing (MMT), modifying (MMM) and understanding (MMU)	Maintenance Cost = MM_T + MM_M + MM_U	Time (person-hours)	Maintainability is directly related to maintenance cost.
S10, S11, S14, S15, S27, S31	Maintenance categorisation according to maintenance attributes	Categorise maintenance according to maintenance attributes: changeability, stability, analysability,	Each study used a different equation	NA	NA

		maintainability and testability			
S1, S13, S16, S53	Maintenance categorisation according to maintenance components	Categorise maintenance according to maintenance components: corrective, adaptive and perfective	Each study used a different equation	NA	NA
S3, S24, S34, S40	Other Measurements	NA	Each study used a different equation	NA	NA

It is apparent from Table 5 that there are several types of software maintenance measurements, these types can be divided into indirect measures and direct measures:

(1) Indirect measures: There are three types that can be used as dependent variables to capture the element of maintainability, which are the CHANGE metric (S2, S20, S25, S26, S30, S32, S35, S36, S38, S39, S42, S43, S44, S45, S46, S47, S49, S50, S51 and S52), the maintainability index (S5, S6, S8, S21, S28, S29, S33, S48, S54 and S55) and change proneness (S47 and S56) .

(2) Direct measures: the remaining types of software maintenance measurements are considered direct measures that are measured software during the maintenance process. These types include corrective maintenance in S18, adaptive maintenance effort in S12, S19 and S22, maintenance time in S7, S17 and S37 and maintenance cost in S9.

Moreover, Table 5 illustrates that relatively few software maintainability measurements are present in the current literature, and this finding is directly in line with previous studies [31].

Figure 7 illustrates the number of selected studies employing the different maintainability measures. From this figure, it can be seen that by far the most popular software maintainability measurement is change maintenance effort that used by twenty selected primary studies which may be attributable to the availability of QUES and UIMS datasets that use this measurement as the dependent variable. Maintainability index is recognised as the second most common measurement, being used by ten studies.

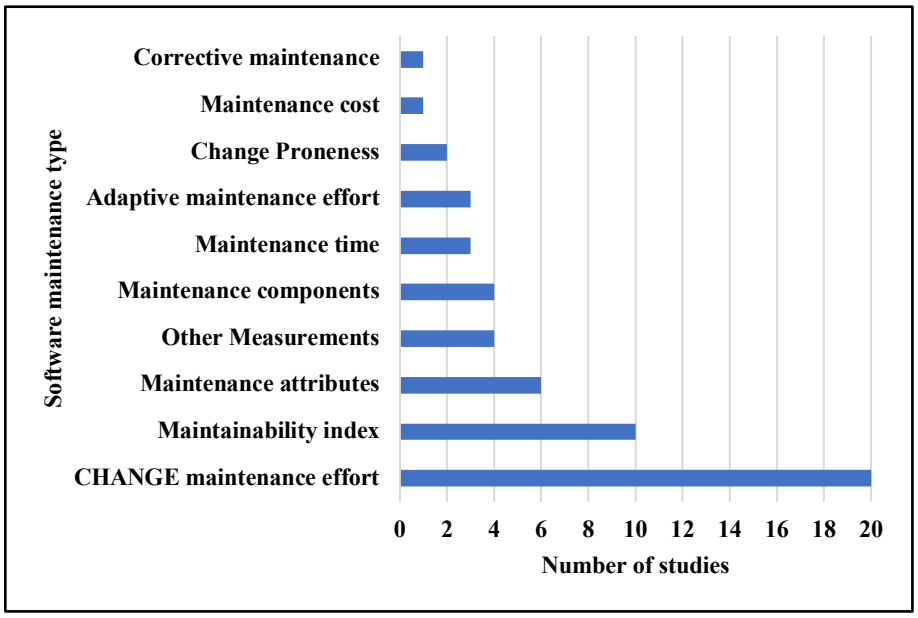

Figure 7: The number of selected studies in each software maintenance type.

4.2 Software maintainability metrics

In this section we address the following research question:

RQ2: What type of OO metrics have been proposed to measure software maintainability, **RQ2.1**: What metrics are used (independent variable)? and **RQ2.2**: What is the level of these metrics?

Software metrics play the most significant role in building a predictive model of software maintainability. OO software metrics are divided into two broad categories: product metrics to measure software maintainability that are based on the quality of software product (e.g. number of lines in source code) and process metrics to measure software maintenance that are based on the quality of software processes (e.g. number of hours to change code) [26].

In addition, software metrics can be categorized into those based on internal attributes that directly measure features of object-oriented software such as inheritance or class cohesion, and external attributes that indirectly measure features of the software environment such as the Change metric that capture elements of maintainability from sets of internal attributes [1].

Furthermore, metrics can be categorised into different levels: method, class, and application. Method level is used mainly with traditional metrics that involve complexity and size metrics, such as Halstead and McCabe [30]. Class level is used commonly and include those proposed by Chidamber and Kemerer [30] and Li and Henry [26] . Chidamber and Kemerer presented a theoretical work to define a set of software metrics for OO systems based on sound measurement theory. They defined six metrics as a base suite for OO designs, namely, DIT, WMC, NOC, CBO, RFC and LCOM (these abbreviations are expanded in Table 6). Li and Henry constructed a model to predict maintenance effort based on 11 object-oriented metrics, five metrics of which came from the Chidamber and Kemerer set (they excluded CBO) and six more proposed by themselves: MPC, ADT, NOM, LOC, SIZE1, SIZE2 (also expanded in Table 6) and the CHANGE metric, which is the dependent variable and based on the number

of changes made in the source code. Application level metrics are extracted from the application as a whole e.g. error-prone subprograms [32] or total number of modules in each application [33].

There were too many different metrics used in selected primary studies, and it is so difficult to describe all these metrics. Therefore, we decided to present the description of metrics used in software maintenance type (Table 5). Table 6 presents the description of the metrics used to predict maintainability in our selected primary studies, grouped according to type of maintenance.

Table 6: Summary of metrics used in different maintenance types.

Type of maintenance	Study ID	Metrics	Description
Change maintenance effort	S2, S20, S25, S26, S30, S32, S35, S36, S38, S39, S42, S43, S44, S45, S46, S47, S49, S50, S51, S52	**Li and Henry metrics**	
		DIT	Depth of inheritance tree
		NOC	Number of children
		MPC	Message-passing coupling
		RFC	Response for a class
		LCOM	Lack of cohesion in methods
		DAC	Data abstraction coupling
		WMC	Weighted methods per class
		NOM	Number of methods
		SIZE1	Lines of code
		SIZE2	Number of properties
Corrective maintenance	S18	NA	Number of tasks requiring software modification
		NB	Number of tasks requiring fixing of data misalignment
		NC	Number of other tasks
		N	Number of the total tasks (N=NA+NB+NC)
		SIZE	Size of the system to be maintained
Adaptive maintenance effort	S12, S19, S22	Eff_{add}	Effort for addition
		Eff_{und}	Effort for understanding
		Eff_{del}	Effort for deletion
Maintenance evaluation by Maintainability Index	S5, S6, S8, S21, S28, S29, S33, S48, S54, S55	HV	Halstead volume metric
		CC	Cyclomatic Complexity metric
		LOC	Counted as a line of code

		COM	A percentage of comment lines
Maintenance evaluation by Change Proneness	S47, S56	**Chidamber and Kemerer metrics**	
		DIT	Depth of Inheritance Tree
		WMC	Weighted Methods Per Class
		NOC	Number of Children
		CBO	Coupling between object classes
		RFC	Response for a Class
		LCOM	Lack of Cohesion in Methods
Maintenance time	S7, S17, S37	IL	Interaction Level
		IS	Interface Size
		OAC	Operation Argument Complexity
		ID	Inheritance Depth
Maintenance cost	S9	MM_T	Man-months to understanding
		MM_M	Man-months to modifying
		MM_U	Man-months to testing

What stands out in Table 6 is the high number of selected primary studies that used the Li and Henry metrics (20 studies). These studies reported evidence of a strong relationship between OO metrics and software maintainability. However, some studies performing feature selection techniques have not clearly specified the best OO metrics for software maintainability prediction. Moreover, the total number of selected primary studies that used the maintainability index metrics, which are widely accepted in industry [34], is half that of the Li and Henry metrics. It would seem likely that the high number of studies using the Li and Henry metrics is due to the availability of QUES and UIMS datasets that include these metrics, while studies using the maintainability index metrics may be due to the availability of tools to extract and measure these metrics.

Table 7 provides a summary of the metrics used to predict maintainability in our selected primary studies, grouped according to type (product/process) along with an indication of the level at which the measurement is captured.

Table 7: Summary of software maintainability metrics.

Metrics type	Metrics level	Study ID
Product metrics	Class level	S2, S4, S7, S10, S11, S14, S20, S25, S26, S27, S30, S32, S36, S38, S39, S40, S42, S43, S46, S47, S49, S50, S52, S56

	Application level	S3
	Class level, method level	S5, S6, S8, S12, S16, S21, S28, S29, S33, S41, S44, S48, S54, S55
	Class level and Application level	S45, S51
Process metrics	Application level	S13, S17, S18, S22, S24, S31, S34, S37
	Application level, class level,	S15, S19
Product and process metrics	Application level	S9, S35

From Table 7, it can be seen that the majority of the selected primary studies used class level product metrics (related to the fact that most of the studies are attempting to predict classes changed in the source code). The table also shows that a large number of selected primary studies used process level metrics to predict application level maintainability, with product metrics only featuring in a small number of studies for this category of change.

Figure 8 provides a visualisation of the data in Table 7, and aggregates the total number of selected primary studies using both metric type and metric level, and clearly illustrates the popularity of employing product metrics for class-level predictions, and process metrics for application level predictions.

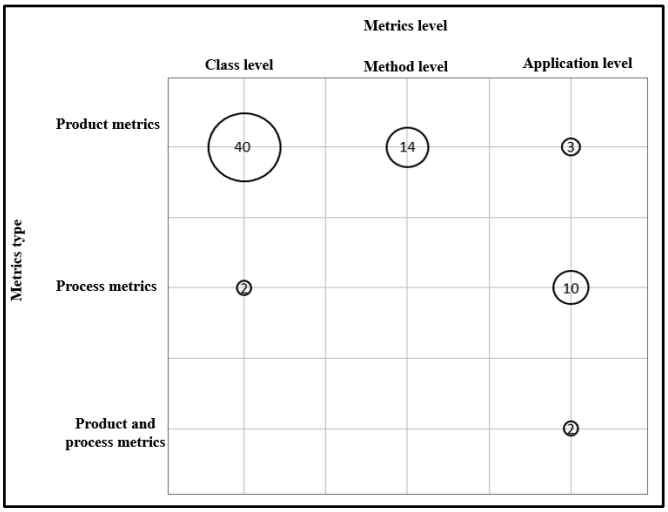

Figure 8: The number of studies using metrics type and metrics level.

Figure 9 presents the distribution of metrics of selected primary studies. 78% of the studies used product metrics and 16% of the studies used process metrics. Moreover, 4% of the studies integrated both product and process metrics. The evidence from this result suggests that product metrics are the majority of metrics used in software maintainability.

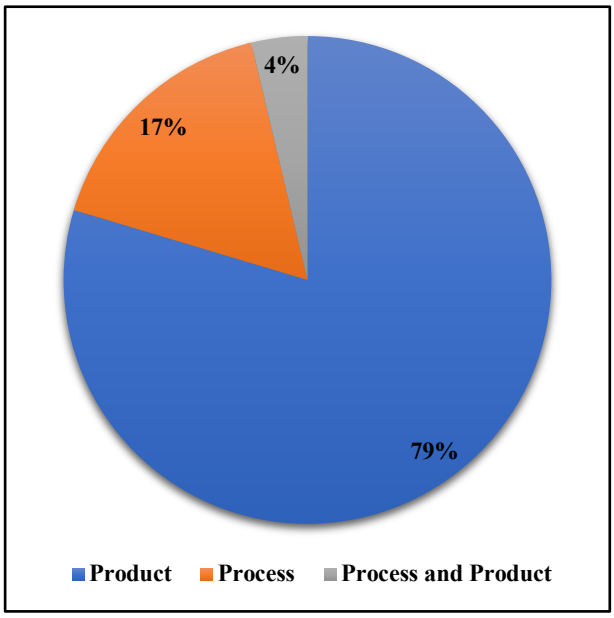

Figure 9: The distribution of metrics.

4.3 Software maintainability datasets

This section seeks to answer the following questions:

RQ3:What are the software maintainability datasets in the literature that were used to build prediction models? **RQ3.1:** What is the type of software maintainability datasets? **RQ3.2:** What tools are used to extract metrics from open source projects? **RQ3.3:** What programming languages are used to collect metrics? **RQ3.4:** What is the number of classes in the software system?

A dataset is a combination of related sets of the data that may be used to perform machine learning models, and it is considered the foundation of building prediction models. For the model building the dataset is divided into a training set, which is used as input to train model, and a testing set, which is used as input to evaluate the model [35].

4.3.1 Datasets used

The datasets used in the selected primary studies may be divided into three main types:

- **Public dataset:** The dataset is available as appendix or table in published paper or in a publicly accessible repository, such as the Promise repositories in S23. In 2018, Hegedűsa et al. in S55 provided their datasets via the PROMISE repository: one of the first regarded as suitable for software maintainability prediction [36]. These datasets were built from extracted OO metrics from seven open-source systems and include a calculated maintainability index. Their contribution provides encouragement to the researcher community to develop more research in software maintainability prediction.
- **Partial dataset:** The dataset is not available, but has been extracted from open source software project repositories such as GitHub [37] or SourceForge [38].

- **Private dataset:** The dataset is not available and has been extracted from a private software system.

Table 8 illustrates a summary of different types of the datasets used in the selected primary studies.

Table 8: Summary of different types of dataset

Public datasets				
Study ID	**Datasets name**	**Dataset Source**	**Dataset size**	**Dataset link**
S2, S20, S25, S26, S30, S36, S38, S46, S47, S49, S52	QUES	Commercial software products (Quality Evaluation System)	71 classes	Provided as an appendix in [26].
	UIMS	Commercial software products (User Interface System)	39 classes	
S11	UML class diagram	Bank Information Systems	27 classes	Provided as an appendix in [39].
S13	Bank	Bank Information Systems	55 classes	Proposed as a table in [33]
S34	Controlled experiment	Academic course	24 classes	Proposed as an appendix in [40].
S53	Spring, Edition, RxJava, Restlet, Hadoop, Camel, Kotlin, Elasticsearch, Hbase Drools, OrientDB	Open source system in GitHub [37]	1151 classes	https://zenodo.org/record/835534#.W123H9IzY2x
S55	Titan, mcMMO, junit, oryx mct, antlr4, mapdb	Open source system in GitHub [37]	10,844 classes	https://zenodo.org/record/581651#.W129Y9IzY2w
Partial dataset				
Study ID		**Dataset Source**		
S27, S29, S33, S39, S41, S47, S50, S54		Open source system in sourceforge [38]		
S21		Industrial software		
S28, S35, S40, S44, S45, S48, S51, S56		Other open source project		
Private dataset				
Study ID		**Dataset Source**		
S1, S3, S4, S5, S6, S7, S8, S9, S10, S12, S16, S17, S18, S19, S22, S24, S31, S32, S37, S42, S43		Private projects		

What can be clearly seen in Table 8 is most selected primary studies make use of two public datasets: User Interface Management System (UIMS) and Quality Evaluation System (QUES) proposed by Li and Henry [26]. These datasets are derived from systems written in Classic-Ada as the object-oriented programming language. A further notable finding is that most of the selected primary studies have been conducted using private datasets, as opposed to public

or partial datasets, which makes many empirical studies of software maintainability prediction not repeatable.

Figure 10 illustrates the number of selected primary studies in each dataset type. From the chart below which clearly illustrates the difference between the number of studies using private datasets compared with other types of datasets. However, there are new public datasets proposed in S53 and S55 that may encourage researchers to apply their models to predict software maintainability.

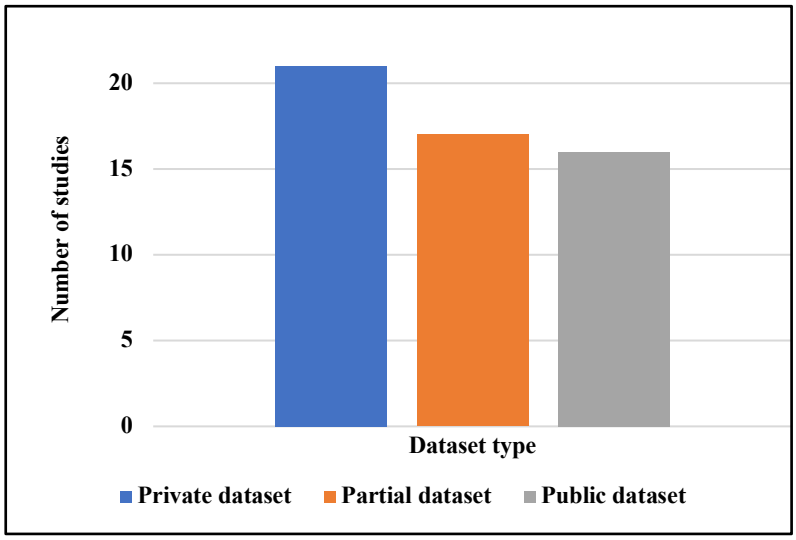

Figure 10:The number of studies used each type of the dataset.

4.3.2 Dataset size

The dataset size may be classified into three main groups: small, where the number of classes less than 100, medium, where the number of classes less than 500, and large, where the number of classes is more than 500. Figure 11 provides the distribution of the dataset size of selected primary studies by counting the number of studies using each size of the dataset. This result shows that selected primary studies were mostly performed on the larger sized datasets, either medium or large, which improves the validity of prediction models.

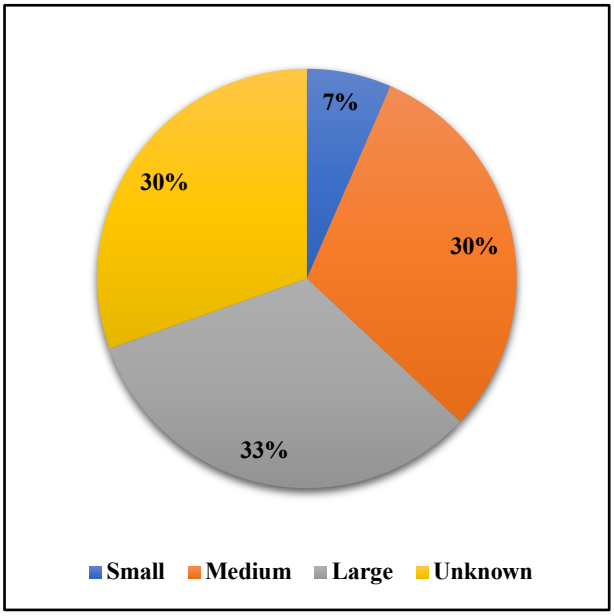

Figure 11: The distribution of the dataset size.

4.3.3 Programming language

Most of the datasets were extracted from systems written in Java, Classic-Ada, C#, or C++. Figure 12 presents the distribution of the dataset language of selected primary studies. Java is the most popular language used in the studies which may be due to the availability of open-source systems written in Java.

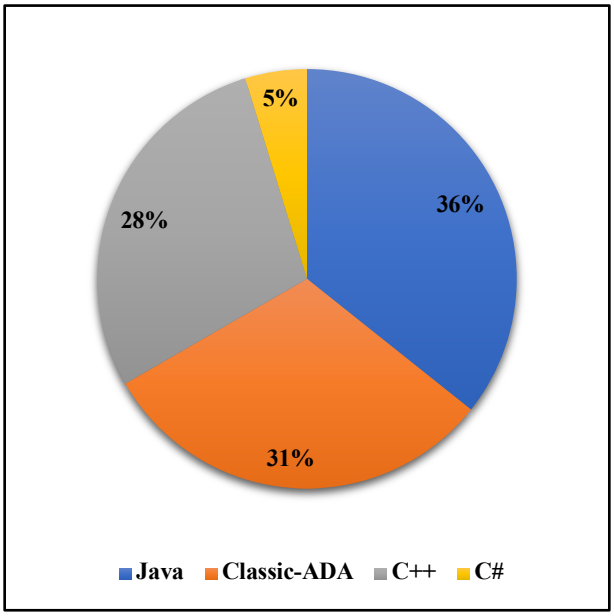

Figure 12: The distribution of programing language.

4.3.4 Tools used for data extraction

The variables in the datasets were extracted from software source by using specific tools. Table 9 presents the tools used for extracting the independent variables of the selected primary studies. The dependent variables were typically collected from comparing the first and the last versions of source software (either manually or by using a tool). Table 10 shows the tools used to identify the changes that have taken place (although several of the studies did not mention the tool used to collect the data).

Table 9: The tool extraction independent metrics.

Programming language	Study ID	Tool name
C	S2, S20, S25, S26, S30, S36, S38, S46, S47, S49, S52	Classic-Ada Metric Analyzer
	S6	HPMAS prior to perfective maintenance modification
C++	S4, S15	An automated data collection tool
	S12	log file
	S21	Krakatau Metrics Professional
C and C++	S5, S8, S11, S13 S17, S31, S34, S37	Survey questionnaire
	S35	Resource Standard Metrics
	S45	Logiscope software static analysis tool.
C++ and Java	S27, S41, S44	CCCC
Java and C#	S27, S41	OOMeter
C#	S44, S41	Visual Studio
C, C++ and Java	S10	Reconfigurable Automated Metrics for Object-Oriented Software
Java	S16	KLOCwork tool
	S27, S41, S43, S46, S48, S50, S56	Chidamber and Kemerer Java Metric (CKJM) tool
	S51	ObjectAid UML Explorer
		JHawk tool
		Classycle
	S54	COINS tool
	S55	SourceMeter static code analysis too

	S56	Defect Collection and Reporting System (DCRS) tool
	S27, S41	Analyst4j
		Dependency Finder
	S28, S51, S41	JDepend
	S27, S28, S41	Understand for Java
	S41	Java Coding Standard Checker

As can be seen from Table 9, there is a broader range of tools available for Java which may be a reflection of the popularity of the language as illustrated in Figure 12. Extraction tools that work with C# are used less frequently. Some of the extraction tools can work with more than one language.

Table 10: The tools for dependent metrics extraction.

Study ID	Tool name	Software maintainability type
S11	Fuzzy Prototypical Knowledge Discovery	Categorize maintainability into easy, medium or difficult maintain
S16	Distributed Software Development (DSD)	Categorize maintenance activities as perfective, adaptive or corrective
S39, S51	Version Control System (VCS)	Track source code changes
S43	Eclipse compare plug-in	Compare changes
S45	Static code analysis tools	Calculate the code change history
S50	Beyond Compare	Compare changes between two versions of software
S55	QualityGate SourceAudit	Calculate relative maintainability index (RMI), similarly to the commonly maintainability index measurement

4.4 Evaluation Measures

This section addresses the following questions:

RQ4: What are the evaluation measures used to assess the performance of software maintainability prediction models? **RQ4.1:** What are the validation approaches used to evaluate the performance of software maintainability prediction models?

Various evaluation measures have been used in software maintainability prediction to evaluate and compare the accuracy of model performance. The appropriate evaluation measure is usually based on problem type: regression, classification or clustering. The evaluation

measures identified in the selected studies are reported in Table 11 which also provides the definition and equation for each measure.

Table 11: Summary of evaluation measures used.

Evaluation measures for Regression problems				
Study ID	**Name**	**Acronym**	**Definition**	**Equation**
S18, S25, S26, S29, S34, S38, S44, S47, S54	Magnitude of relative error	MRE	The absolute difference between the actual value and predicted values divided by the actual value	MRE = |actual value – predicted value| / actual value
S25, S26, S30, S34, S38, S49, S38, S44, S47, S54	Mean magnitude of relative error	MMRE	It is the mean of MRE.	$\text{MMRE} = 1/\text{n} \sum_{i=1}^{i=n} MRE$
S18, S25, S26, S30, S34, S38, S42, S44, S47, S54	Pred(q)	PRED	It calculates the proportion of instances in the dataset where the MRE is less than or equal a specified value (q). The q is defined value, k is the number of states where MRE is less than or equal to q, and n is the whole number of views in the dataset.	Pred(q) = k/n
S20, S33, S38	Mean square error	MSE	It measures the average of the squares of the differences between the actual and predicted values.	$\text{MSE} = 1/\text{n} \sum_{i=1}^{i=n} (Y_i - \acute{Y}_i)^2$
S26, S29	Absolute relative error	ARE	The Ab. Res. is the absolute value of residual, which is the difference between the actual value and predicted	Ab.Res = |actual value – predicted value|
S32, S42, S46, S52	Mean Absolute Relative Error	MARE	It is the mean of the ARE	$\text{MARE} = 1/\text{n} \sum_{i=1}^{i=n} ARE$
S20, S36, S46, S49, S50, S52	Mean absolute error	MAE	It is the average of absolute values of the difference between $X'i$ and $X\,i$, where $X'i$ is the predicted value and $X\,i$ is the actual value	$\text{MAE} = 1/\text{n} \sum_{i=1}^{i=n} (\lvert X'i - X\,i \rvert)$
S46, S50	Root Mean Square Error	RMSE	It is the square root between the square of predicted value and the actual divided by number of observations in the dataset, where $X'i$ is the predicted value and $X\,i$ is the actual value	$\text{RMSE}= \frac{\sqrt{(X'i - X\,i)^2}}{n}$
S46, S52	Standard Error of the Mean	SEM	It is the standard deviation divided by root of the number of observations in the dataset.	$\text{SEM} = \text{SD}/\sqrt{n}$
S2, S5, S8, S12, S16, S17, S18, S19, S20, S21, S24, S29, S31,	R square	R^2	It presents the proportion of the variance of the dependent variable that is explained by the model.	R-squared = 1- (Explained variance / Total variance), where explained variation is the sum of squares of the residuals (i.e. actuals-predicted) and total variation is the

S32, S33, S34, S36, S48, S51				residuals with respect to the average (i.e. actuals – average) [41].
Evaluation measures for Classification problems				
Study ID	**Name**	**Acronym**	**Definition**	**Equation**
S37, S45, S53, S56	Accuracy		It is the number of true predictions over all types of predictions.	Accuracy = (TP + TN) / (TP + TN + FP + FN)
S37, S40, S43, S53, S56	Recall		It is a proportion of actual positives that are correctly determined.	Recall = TP / TP + FN
S37, S40, S43, S53, S56	Precision		It is capability of a model to correctly identify relevant instances.	Precision = TP / TP + FP
S37, S39, S56	F-Measure		It integrates precision and recall/sensitivity in one equation.	F-Measure = 2 * Precision * Recall / Precision + Recall
S56	Specificity		It is a proportion of actual negative that are rightly determined.	Specificity = TN/ TN+ FP
S39, S40, S43, S47	Area under curve of ROC curve	AUC	It is a plot of two parameters: the true positive rate and false positive rate.	
Evaluation measures for Clustering problems				
Study ID	**Name**		**Definition**	**Equation**
S11, S31	Mean cluster		It calculates how close the clustering is to the preidentified classes by average of all cross-cluster pairs, where $a_i = (a_1+a_2+ \ldots..+a_n)$	Mean= $1/n \sum_{i=1}^{i=n} (a\ i)$

**TN: True negatives, FP: False Positive, FN: False Negative, TP: True Positive.

From Table 11, key findings emerge: the most popular prediction accuracy measures used for regression problems after R-squared are those based on the magnitude of relative error (MRE, and MMRE) [42] and (PRED) [5]. However, prior studies have pointed out that several evaluation measures (e.g. MMRE) have bias and instability issues in their results [43, 44]. To resolve these issues, a baseline or benchmark is recommended to compare and evaluate the performance of the prediction models [43, 44]. For classification problems, the most commonly used evaluation measures are recall and precision, followed by accuracy.

Figure 13 shows the number evaluation measures in each machine learning problem used by selected primary studies. R-squared is the most frequently used evaluation measure for regression problems (19 studies), followed by MMRE and PRED (10 studies). For the classification problem Recall and Precision were applied by 5 studies. In the clustering problem, only one evaluation measure was employed.

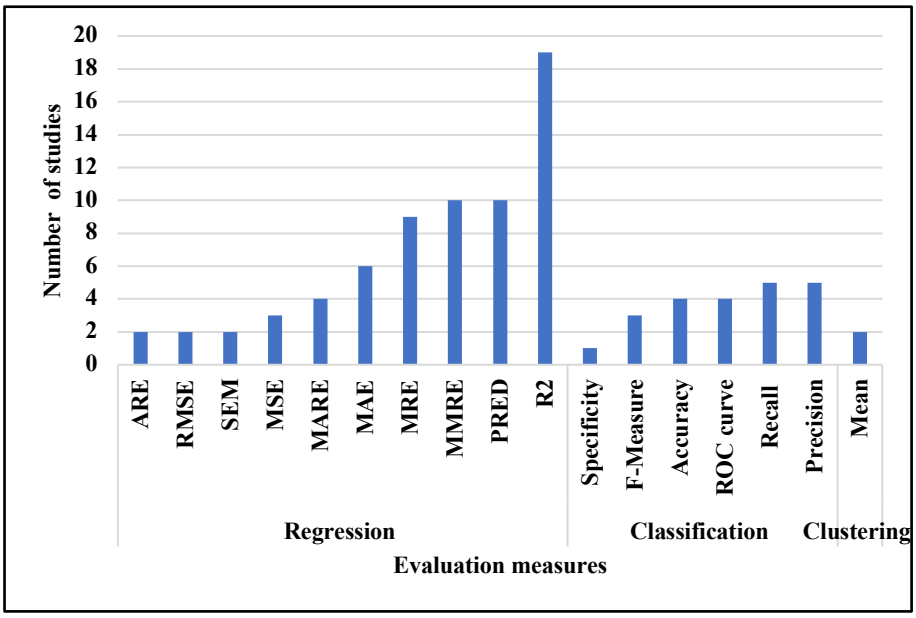

Figure 13: The distribution of evaluation measures used by selected primary studies.

Validation is used to evaluate the performance of a trained model on previously unseen data [35]. There are three major validation types used in the selected primary studies: k-fold cross-validation, leave-one-out and holdout. Table 12 illustrates the validation types used by the selected primary studies. A fact that worth mentioning is that this table contains fewer than half of the selected primary studies, the remainder of the studies measured software maintainability directly without applying prediction models. Therefore, they did not need to validate the model's performance.

Table 12: Summary of validation types.

Study ID	Validation Types	Definition
S20, S37, S39, S40, S45, S46, S47, S49, S52, S53, S54, S56	k-fold cross-validation	The dataset is divided randomly into K folds (or partitions) of the same size, where one-fold is used to test the model and the remaining *k*-1are used as training data. The process is repeated *k* times to select a new different fold at each iteration. The overall performance is based on the average over the *k* test folds.
S18, S26, S29, S30	Leave-one-out	It is a logical extreme version of k-fold cross validation where k is equal to the size of the data set. One observation of the total dataset is removed, and the model is constructed with the remaining dataset and tested in the removed observation. It then repeats the process by deleting a new different observation and so on for the whole data set.

S4, S17, S19, S25, S32, S36, S38, S42, S44	Holdout	It partitions dataset into two sets, where one portion is used for training the model and another portion for testing.

Figure 14 presents the number of selected primary studies employing each validation type. The results show that k-fold cross validation is the most commonly used validation type, followed by holdout.

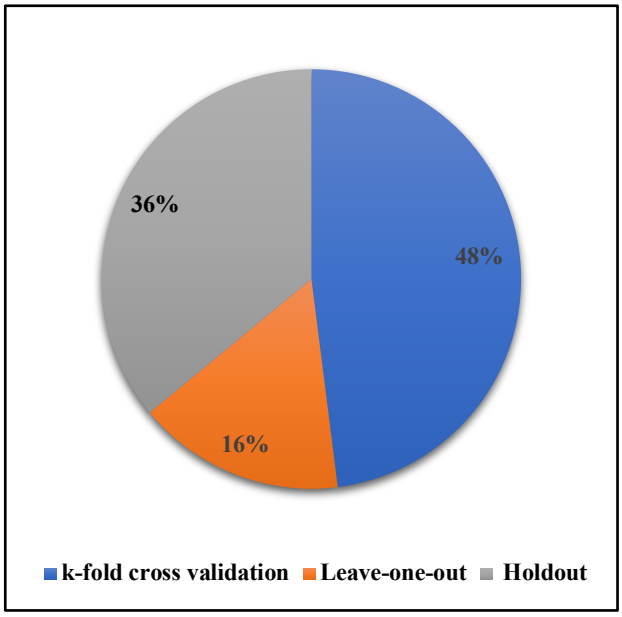

Figure 14: The distribution of valadition types.

4.5 Machine Learning Problem Category

This section addresses following RQs: **RQ5:** What type of machine learning was performed in the software maintainability domain? **RQ5.1:** What are the categories of machine learning problem explored?

Machine learning approaches may be divided into three main groups: supervised, unsupervised and semi-supervised. Supervised learning encompasses two machine learning problem types: regression and classification, while unsupervised learning comprises clustering and association. The semi-supervised is a combination of supervised and unsupervised. The regression problem predicts a continuous number, where the classification predicts a category of two or more types. Clustering groups data together based upon attributes and distance measures, whereas association detects rules that explain a large amount of the data. Moreover, supervised learning builds machine learning models to predict output data from the input data, where the input data has a label. Unsupervised learning builds models from unlabelled input data. Semi-supervised learning builds models to predict output data from input data, where some data items have a label [35]. Table 13 shows the summary of machine learning problems used by the selected primary studies. As mentioned in the explanation of Table 12, most of the

selected primary studies used direct measures to evaluate software during the maintenance process without performing any machine learning problems.

Table 13: Summary of machine learning problems.

Study ID	Machine learning approach	Machine learning problem type
S20, S25, S26, S29, S30, S38, S42, S44, S45, S46, S47, S48, S49, S50, S52, S54	Supervised	Regression
S37, S39, S40, S43, S47, S53, S56		Classification
S11, S31, S32	Unsupervised	Clustering

As can be seen from Figure 15 below, regression problems feature far more frequently than other machine learning problems when predicting software maintainability.

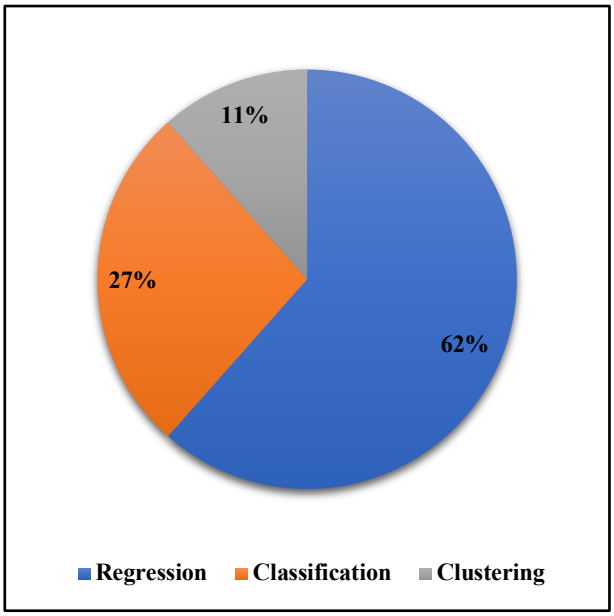

Figure 15: The frequency of machine learning problem types.

4.6 Individual Prediction Models

This section explores different individual prediction models used in the selected primary studies of software maintainability prediction and identifies the superior model in each study. This section addresses the following questions: **RQ6:** What are the individual prediction models used to predict software maintainability? **RQ6.1:** Which are the best performing individual prediction models?

Table 14 presents a summary of the studies that have used individual models to predict software maintainability, along with the best accuracy prediction model over the most evaluation measures. The selection of the best model relies on the evaluation measures that

compare and evaluate the prediction accuracy of the individual prediction models used in each study. According to these measures, the best model is selected as a recommendation model in each selected study. Again as indicated in Table 12 and Table 13, Table 14 includes a subset of selected primary studies, since only these studies used indirect measures to predict software maintainability and apply machine learning models.

Table 14: Summary of individual prediction models used with the best model in each study

Study ID	Individual prediction models	The best accuracy prediction model
S11	FDP	FDP
S20	WNN and GRNN	GRNN
S25	BN, RT, MLR (Backward elimination) and MLR (Stepwise selection)	BN
S26	MARS, MLR, SVR, ANN and RT	MARS
S29	LR	LR
S30	TreeNet, MARS, MLR, SVR, ANN, and RT	TreeNet
S31	K-means clustering	K-means cluster
S32	SVM	SVM
S36	MLP, WNN and GRNN	MLP
S37	LR, DT, CART, BFTree and LEGAL-Tree	DT
S38	FL, BN and MARS	FL
S39	Multivariate model	Multivariate model
S42	FF3LBPN, GRNN and GMDH	GMDH
S44	BPN, KN, FFNN, GRNN	KN
S45	SVR, MLP, IBk, M5Rules, KStar, GP, AR, REPTree, M5P, RF, RSS, IR, PLS classifier, GS, DS, PR, CR, RBD, LMS, LR, Decision table, LWL, RBFNN	SVR
S46	Neuro-GA Approach	Neuro-GA Approach
S48	MLR based on stepwise selection and backward elimination methods	Stepwise selection
S49	FLANN, GA, PSO, CSA, FLANN-GA, FLANN-PSO, FLANN-CSA	FLANN-GA
S50	GA, Decision Table, RBFNN, BN and SMO	GA
S52	Neuro-Fuzzy approach	Neuro-Fuzzy approach
S54	MLR, MLP, SVR, M5P and RT.	SVR

** FDP: Fuzzy Deformable Prototypes, WNN: Ward neural network, GRNN: General regression neural network, BN: Bayesian network, RT: Regression tree, MLR: multiple linear regression, MARS: Multiple adaptive regression splines, SVR: Support vector regression, ANN: Artificial Neural Network, LR: Linear regression, TreeNet: Multiple additive regression trees, SVM: Support vector machine, MLP: Multilayer Perceptron, GRNN: general regression neural network, LR: logistic regression, DT: decision tree, CART: Classification and Regression Trees, FL: fuzzy logic-based, FF3LBPN:Forward 3-Layer Back Propagation Network, , GMDH: Group Method of Data Handling, BPN: Back Propagation Network, KN: Kohonen Network, FFNN: Feed Forward Neural Network, GP: Gaussian processes, AR: Additive

regression, RF: Random forest, RSS: Random subspace, IR: Isotonic regression, GS: Grid search, DS: Decision stump, PR: Pace regression, CR: Conjunctive rule, RBD: Regression by discretization, LMS: LeastMedSq, LWL: locally weighted learning, FLANN: functional link artificial neural network, GA: Genetic algorithm, PSO: Particle swarm optimization, CSA: clonal selection algorithm, RBFNN: Radial Basis Function Neural Network, SMO: Sequential Minimal Optimization.

We reported in Table 14 only the best model performance regardless of which evaluation measurements were used. As shown in Table 14, several selected primary studies have different recommended individual models to predict software maintainability. However, two studies have reported that SVR outperforms other selected models.

The results for an accurate prediction model are recognized if they meet the criteria of pred(.30) $\geq$ 0.70 [45] or pred(.25) $\geq$ 0.75 or/and MMRE $\leq$ 0.25 [46]. Even though the suggested criteria are based on relatively old references [45, 46], recent studies have employed these criteria to evaluate prediction accuracy in the software engineering domain [47-49]. Also, several selected primary studies have used these criteria, such as S25, S26, S30, S38 and S54. However, S35 suggested that it is a challenging task to meet these criteria.

Table 15 presents the performance of the MMRE value for some selected primary studies. However, several studies in Table 14 did not use MMRE or had not performed a regression problem, so we could not evaluate their performance against the criteria. The result of this table indicates that FLANN-GA in S49 is the only model that meets the criteria of MMRE, while FL model in S38 and Neuro-Fuzzy approach in S52 are close to meeting the criteria to build an accurate effort prediction model.

Table 15: Performance of MMRE value for some selected primary studies.

Dataset name	Study ID	The best accuracy prediction model	MMRE
QUES	S25	Stepwise selection	0.39
	S26	MARS	0.32
	S30	MARS	0.32
	S38	FL model	**0.27**
	S46	Neuro-GA Approach	0.37
	S49	FLANN-GA	0.32
	S52	Neuro-Fuzzy approach	0.33
UIMS	S25	Bayesian network	0.97
	S26	SVR	1.86
	S30	TreeNet	1.57
	S38	FL model	0.53
	S46	Neuro-GA Approach	0.31
	S49	FLANN-GA	**0.24**
	S52	Neuro-Fuzzy approach	**0.28**

Five open source Java software systems	S44	KN	Mean MMRE: 0.44 for model 1. Mean MMRE: 0.32 for model 2.
Twenty-six open source Java software systems	S54	SVR	Mean MMRE: 0.91

The number of individual prediction models that has been used in selected primary studies is illustrated in Figure 16 (fifty-three in total). From the fifty-three models shown in Figure 16, the five most frequently employed individual models are MLR, SVR, GRNN, RT and FLANN. MLR is the most frequently used individual model for software maintainability (six studies).

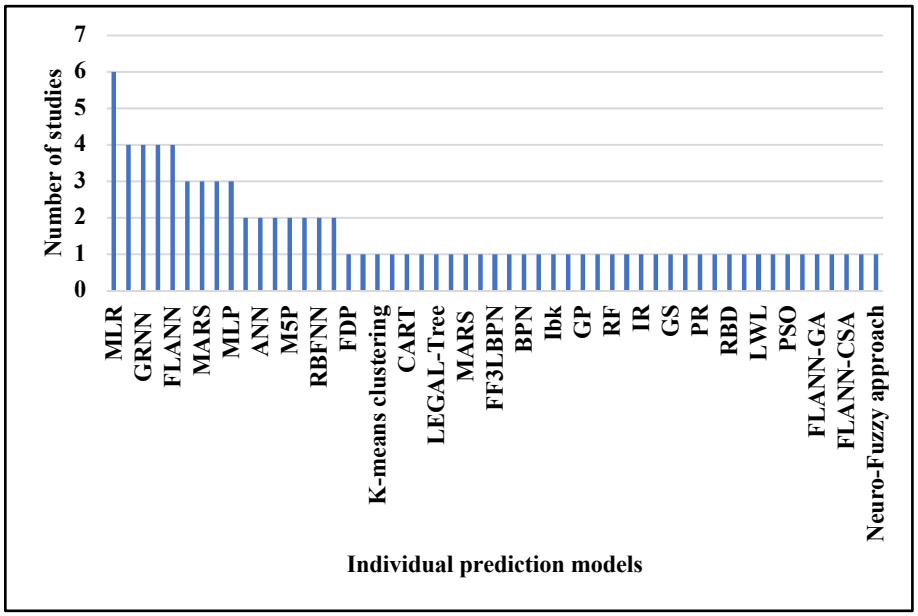

Figure 16: Individual prediction models used in selected primary studies.

4.7 Ensemble Prediction Models

This section investigates various ensemble prediction models used in the studies and compares the ensemble model performance against individual models. The aims of this section are to answer the following questions: **RQ7:** What type of ensemble prediction models were employed to predict software maintainability? And **RQ7.1:** Are the ensemble models able to improve over the performance of the individual prediction models?

An ensemble model is a combination several individual models designed to improve on the accuracy prediction of individual models. They can be classified into two major types: homogenous that uses the same type of individual models, and heterogenous that uses different types of individual models [50]. Moreover, the ensemble models can be categorized into linear ensembles, which combine the outputs of the base model in a linear manner (e.g. weighted averaging, averaging) and nonlinear ensembles, which combine the outputs of the base model

in a nonlinear manner (e.g decision tree forest) [51]. Five main selected primary studies employed ensemble models to predict software maintainability. These studies emphasized the positive impact of ensemble methods in predicting software maintainability compared with individual prediction models. Table 16 summarises the ensemble prediction models used.

Table 16: Summary of the ensemble prediction models.

Study ID	Ensemble Type	Ensemble name	Base models	Combination rules	The best model	Does the ensemble model improve the performance of the base model?
S40	Heterogeneous	SMEM-MCC	ISMEM, DT, BPN, SMO	NA	Ensemble model (SMEM-MCC)	Yes
S43	Homogeneous	RF	Naïve Bayes, Bayes Network, Logistic, Multilayer Perceptron	Averaging	Ensemble model (RF)	Yes
S47	Heterogeneous	Linear ensemble	MLP, RBF, SVM, M5P	Averaging, weighted averaging and best in training	Ensemble model	Yes
	Homogeneous	Bagging and AdaBoost	MLP, RBF, SVM, DT	Averaging		Yes
	Heterogeneous	Linear ensemble and Non-linear	SVM, MLP, logistic regression, genetic programming, K-means	Best in training, majority voting and decision tree forest		Yes
S53	Homogeneous	RF and AdaBoost	J48	Averaging	Ensemble model (AdaBoost)	Yes
S56	Homogeneous	MVEC, WVEC, HIEC, WVHIEC	Seven individual Particle Swarm Optimization (PSO)	Majority voting, weighted voting and hard instance	Ensemble model (HIEC and WVHIEC)	Yes

** SMEM-MCC: Software Maintainability Evaluation Model based on Multiple Classifiers Combination, RF: Random forest, Bagging: Bootstrap aggregating, AdaBoost: Adaptive Boosting, MVEC: Majority Voting Ensemble Classifier, WVEC: Weighted Voting Ensemble Classifier, HIEC: Hard Instance Ensemble Classifier, WVHIEC: Weighted Voting Hard Instance Ensemble Classifier.

As shown in Table 16 above, different types of ensemble models have been compared with individual prediction models. The ensemble prediction models in selected primary studies (i.e. S40, S43, S47, S53 and S56) improved the performance of individual models and increased their accuracy prediction over individual prediction models. Even though the ensemble models reported a superior result over individual prediction models, a limited number of selected primary studies applied the ensemble models to predict software maintainability (which may be due to the fact that ensemble models are relatively new [52]). Figure 17 presents the distribution of prediction models in software maintainability. This figure provides evidence of

the restricted ensemble models used in the selected primary studies compared with individual prediction models.

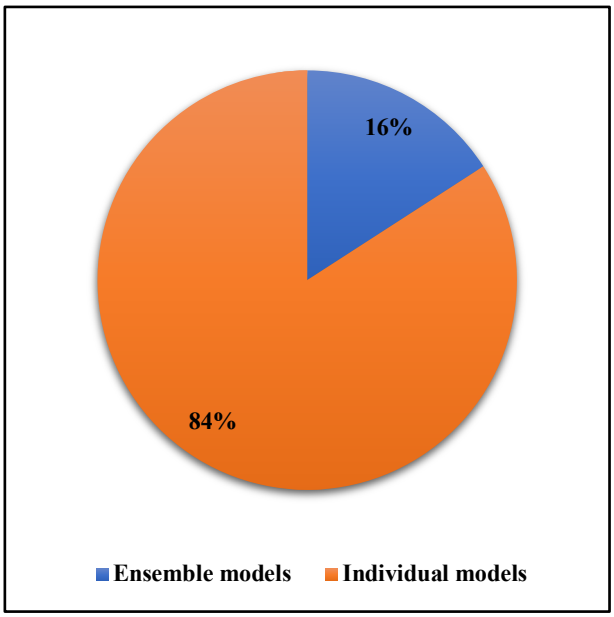

Figure 17: The distribution of prediction models.

The number of ensemble prediction models used in selected primary studies is shown in Figure 18 (ten in total). The number of homogeneous ensemble models used exceeds the heterogeneous ones. Furthermore, the linear ensemble is the most frequently used heterogeneous model, and RF and AdaBoost are the most frequently used homogeneous models.

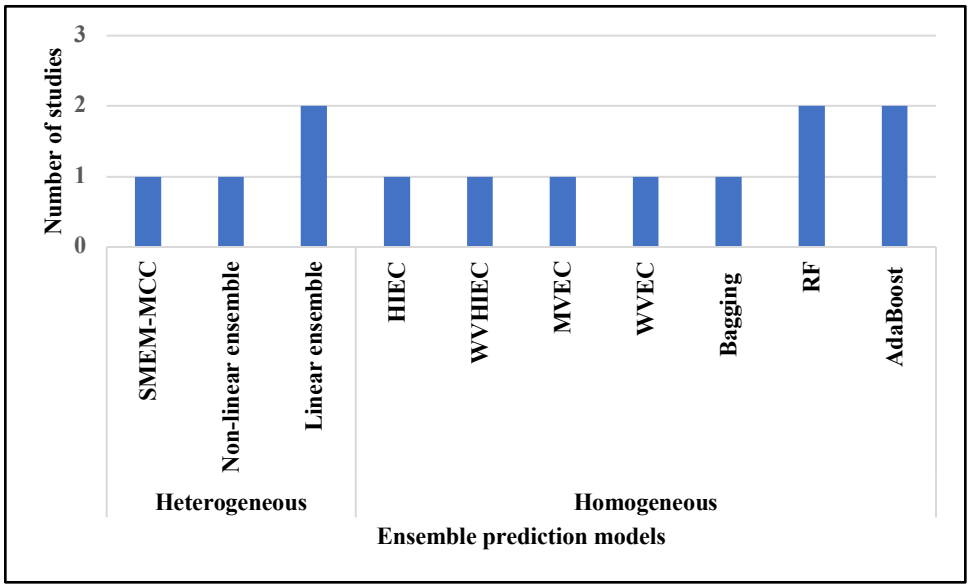

Figure 18: Ensemble prediction models used in selected primary studies.

5 Conclusion

This literature review aimed to identify and analyse the measurements, metrics, datasets, evaluation measures, machine learning problems, individual prediction models and ensemble prediction models employed in the field of software maintainability prediction. An extensive search was conducted in five online digital libraries to select peer-reviewed articles that publish in either journals or conferences. Fifty-six studies have been selected between 1991 and 2018, and seven main questions have been answered from each selected primary study. This literature review has been performed as a SLR to evaluate all relevant research evidence and identify the available studies in the field of software maintainability prediction, with the purpose to answer certain RQs.

The main answer for each research question in this literature review is reported in Figure 19 (**Appendix B**). This figure provides details and highlights the most important answers. This visualisation enables us to break down the complex problem of software maintainability prediction into several solutions according to our RQs. As a result, this mind map integrates a whole literature review into one organised picture to respond to our RQs as the following:

- There are ten software maintainability measurements used in selected primary studies: CHANGE maintenance effort, corrective maintenance, adaptive maintenance effort, maintenance evaluation by maintainability index, maintenance evaluation by change proneness, maintenance time, maintenance cost, maintenance attributes, maintenance components and other measurements.
- There are three main types of software maintainability metrics used in selected primary studies: product metrics, process metrics, and product metrics combined with process metrics.
- There are three main types of software maintainability datasets used in selected primary studies: public datasets, partial datasets and private datasets.
- The software maintainability evaluation measures are grouped by machine learning problem: regression problem-based models include MRE, MMRE, PRED, MSE, ARE, MARE, MAE, RMSE, SEM and R2, classification problems include Accuracy, Recall, Precision, F-Measure, Specificity and ROC curve, and cluster problems include the mean.
- Machine learning models to predict software maintainability involve two main categories: supervised learning for regression and classification, and unsupervised for clustering.
- Fifty-three individual models were constructed by selected primary studies.
- Ten ensemble models were created by selected primary studies (seven homogeneous ensemble models and three heterogeneous ensemble models).

Most notably, this is the first study to our knowledge that provides a SLR in software maintainability prediction. Therefore, we have confidence our proposed study will be a novel and hopefully valuable contribution to the area of software maintainability prediction. The findings obtained in this study can be used by the researchers to provide an overall overview of this area. The main findings derived from this SLR are:

- Relatively few studies have been conducted in software maintainability prediction compared with other software quality attributes such as defect prediction or fault prediction.
- Different types of software maintainability measurements were investigated that roughly equate to ten types. The change maintenance effort was the most common measurement used by 20 studies, followed by the maintainability index, which is used by ten studies. Studies of software maintainability metrics were generally categorized into three types: product metrics, process metrics and product and process metrics. These types were applied on different levels: class level, method level and application level. Most studies used product metrics along with class level measurements (forty studies), followed by product metrics with method level measurements (fourteen studies), and finally process metrics with application level measures (ten studies). Overall, the total distribution of software maintainability metrics is as follows. 79% of studies used product metrics, 17% of studies used process metrics, while only 4% of studies used product and process metrics.
- The Li and Henry metrics are utilised by twenty studies, and these studies confirmed the evidence of the power relationship between OO metrics and software maintainability. Ten studies used the maintainability index metrics.
- Studies of software maintainability datasets were broadly divided into three types: public, private and partial. The most commonly used was private datasets (more than twenty studies).
- UIMS and QUES datasets were the most frequently used (eleven studies). Regarding the size of the datasets, the result reveal that most studies used either medium or large sized datasets, which improves the validity of prediction models.
- 36% of the datasets were collected from Java systems, most likely as a consequence of the availability of open-source systems written in Java.
- Even though there were several extraction tools used to collect metrics, most of these tools were designed for Java. However, some of these tools can work with more than one programming language.
- The evaluation measures are selected usually based on problem type: regression, classification or clustering. R-squared is the most repeatedly used in regression problem by nineteen studies, followed by MMRE and PRED (ten studies). Recall and Precision were applied many times in the classification problem. In the clustering problem, only one evaluation measure was employed.
- Three primary validation types used in studies are k-fold cross-validation (48%), leave-one-out (36%) and holdout.
- The most popular machine learning problems were regression problems (62% of the total selected primary studies). 27% of the studies were related to classification problems, and only 11% of these studies were related to clustering problems.
- Some individual models achieved predictions that were close to meeting the criteria of an accurate prediction model, which are pred(.30) $\geq$ 0.70 [45] or pred(.25) $\geq$ 0.75 or/and

MMRE ≤ 0.25 [46], namely FL model in S38 and Neuro-Fuzzy approach in S52. FLANN-GA in S49 is the only model that meets the MMRE criteria.

- MLR, FLANN, RT, GRNN and SVR were the most frequently employed individual models in software maintainability prediction, with more than four studies using these models.
- Ensemble models were used by five (16%) of the selected primary studies (i.e. S40, S43, S47, S53 and S56), compared to more than three-quarters of the studies which used individual models. However, all the ensemble models used in studies yield improved accuracy over the individual models.
- Ten ensemble prediction models were applied to predict software maintainability (seven homogeneous ensemble models and three heterogeneous ensemble models).
- The linear ensemble was the most frequently used heterogeneous model, while RF and AdaBoost were the most frequently used homogeneous models.

The findings of this SLR revealed the following guidelines for the researchers in software maintainability prediction for future work:

- There is a need for more investigations to be carried out in the area of software maintainability prediction using machine learning techniques, as only 26 studies from the selected primary one using machine learning techniques.
- Only one model meets the model accuracy criteria mentioned earlier (MMRE ≤ 0.25), so further studies are needed to empirically explore and aim to improve the performance of machine learning techniques.
- A limited number of studies explored the capability of ensemble models, even though these have been shown to improve the performance of the base model.
- The QUES and UIMS datasets are publicly available and used by most of the selected primary studies (ten studies) but are small data sets and quite old. There is a need to a larger number of more recent and more substantial datasets to become publicly available to encourage an increase in the number of experimental studies of machine learning techniques.

Acknowledgments

This research was funded by the Deanship of Scientific Research at Princess Nourah bint Abdulrahman University through the Fast-track Research Funding Program.

Appendix A

See Table 17 and Table 18.

Table 17: Selected primary studies.

Study ID	Ref	Topic	Author	Year	Place published	Publication name	Type

S1	[53]	Cyclomatic Complexity Density and Software Maintenance Productivity	Gill and Kemerer	1991	IEEE	IEEE Transactions on Software Engineering	Journal
S2	[26]	Object-Oriented Metrics that Predict Maintainability	Li and Henry	1993	Elsevier	Journal of Systems and Software	Journal
S3	[32]	Measuring and Assessing Maintainability at the End of High Level Design	Briand et al.	1993	IEEE	Conference on Software Maintenance	Conference
S4	[30]	A Metrics Suite for Object Oriented Design	Chidamber and Kemerer	1994	IEEE	IEEE Transactions on Software Engineering	Journal
S5	[27]	Construction and Testing of Polynomials Predicting Software Maintainability	Oman and Hagemeister	1994	Elsevier	Journal of Systems and Software	Journal
S6	[54]	Using Metrics to Evaluate Software System Maintainability	Coleman et al.	1994	ACM DL	Computer	Journal
S7	[55]	Evaluating Inheritance Depth on the Maintainability of Object-Oriented Software	Daly et al.	1996	Springer link	Empirical Software Engineering	Journal
S8	[56]	Development and Application of an Automated Source Code Maintainability Index	Welker et al.	1997	Wiley online library	Journal of Software Maintenance: Research and Practice	Journal
S9	[57]	A Method for Estimating Maintenance Cost in a Software Project: A Case Study	Granja-Alvarez and Barranco-García	1997	Wiley online library	Journal of Software Maintenance: Research and Practice	Journal
S10	[58]	Reusability and Maintainability Metrics for Object-Oriented Software	Lee and Chang	2000	ACM DL	38th annual on Southeast regional conference	Conference
S11	[39]	Using Metrics to Predict OO Information Systems Maintainability	Genero et al.	2001	ACM DL	13th International Conference on Advanced Information Systems Engineering	Conference
S12	[59]	Estimation and Prediction Metrics for Adaptive Maintenance Effort of Object-Oriented Systems	Fioravanti and Nesi	2001	IEEE	IEEE Transactions on Software Engineering	Journal
S13	[33]	Using Code Metrics to Predict Maintenance of Legacy Programs: A Case Study	Polo et al.	2001	IEEE	Proceedings IEEE International Conference on Software Maintenance.	Conference
S14	[60]	Metrics for Maintainability of Class Inheritance Hierarchies	Sheldon et al.	2002	Wiley online library	Journal of Software Maintenance: Research and Practice	Journal
S15	[31]	An Integrated Measure of Software Maintainability	Aggarwal et al.	2002	IEEE	Annual Reliability and Maintainability Symposium.	Conference
S16	[61]	Predicting Maintainability with Object-Oriented Metrics - An Empirical Comparison	Dagpinar and Jahnke	2003	IEEE	10th Working Conference on Reverse Engineering.	Conference
S17	[62]	Predicting Maintenance Performance Using Object-Oriented Design Complexity Metrics	Bandi et al.	2003	IEEE	IEEE Transactions on Software Engineering	Journal
S18	[3]	Assessing Effort Estimation Models for Corrective Maintenance Through Empirical Studies	De Luciaa et al.	2004	Elsevier	Information and Software Technology	Journal

S19	[63]	A Metrics-Based Software Maintenance Effort Model	Hayes et al.	2004	IEEE	Eighth European Conference on Software Maintenance and Reengineering	Conference
S20	[64]	Application of Neural Networks for Software Quality Prediction using Object-Oriented Metrics	Thwin and Quah	2005	Elsevier	Journal of Systems and Software	Journal
S21	[65]	Modeling Design/Coding Factors That Drive Maintainability of Software Systems	Misra	2005	Springer link	Software Quality Journal	Journal
S22	[66]	An empirical investigation of the impact of the object-oriented paradigm on the maintainability of real-world mission-critical software	Lim et al.	2005	Elsevier	Journal of Systems and Software	Journal
S23	[67]	The Promise of Public Software Engineering Data Repositories	Cukic	2005	IEEE	IEEE Software	Journal
S24	[68]	Maintainability Prediction: A Regression Analysis of Measures of Evolving Systems	Hayes and Zhao	2005	IEEE	IEEE International Conference on Software Maintenance	Conference
S25	[5]	An Application of Bayesian Network for Predicting Object-Oriented Software Maintainability	Koten and Gray	2006	Elsevier	Information and Software Technology	Journal
S26	[69]	Predicting Object-Oriented Software Maintainability Using Multivariate Adaptive Regression Splines	Yuming Zhou and Hareton Leung	2007	Elsevier	Journal of Systems and Software	Journal
S27	[70]	Comparing software metrics tools	Lincke et al.	2008	ACM DL	International Symposium on Software Testing and Analysis	Conference
S28	[34]	A Comparative Study ofMI Tools: Defining the Roadmap to MI Tools Standardization	Sarwar et al.	2008	IEEE	IEEE International Multitopic Conference	Conference
S29	[71]	Predicting the Maintainability of Open Source Software Using Design Metrics	Yuming and Baowen	2008	Springer link	Wuhan University Journal of Natural Sciences	Journal
S30	[4]	Application of TreeNet in Predicting Object-Oriented Software Maintainability: A Comparative Study	Elish and Elish	2009	IEEE	Software Maintenance and Reengineering	Conference
S31	[72]	An empirical analysis of the impact of software development problem factors on software maintainability	Chen and Huang	2009	Elsevier	Journal of Systems and Software	Journal
S32	[73]	Applications of Support Vector Mathine and Unsupervised Learning for Predicting Maintainability using Object-Oriented Metrics	Jin and Liu	2010	IEEE	Second International Conference on MultiMedia and Information Technology	Conference
S33	[74]	Determination of Maintainability Index for Object Oriented Systems	Kaur and Singh	2011	ACM DL	ACM SIGSOFT Software Engineering Notes	Journal
S34	[40]	A Controlled Experiment in Assessing and Estimating Software Maintenance Tasks	Nguyen et al.	2011	Elsevier	Information and Software Technology	Journal
S35	[75]	Assessing Programming Language Impact on Development and Maintenance: A Study on C and C++	Bhattacharya and Neamtiu	2011	ACM DL	33rd International Conference on Software Engineering	Conference

S36	[76]	Maintainability Prediction of Object-Oriented Software System by Multilayer Perceptron model	Dubey et al.	2012	ACM DL	ACM SIGSOFT Software Engineering Notes	Journal
S37	[77]	Predicting Software Maintenance Effort through Evolutionary-Based Decision Trees	Basgalupp et al.	2012	ACM DL	27th Annual ACM Symposium on Applied Computing	Conference
S38	[1]	Machine learning approaches for predicting software maintainability: A Fuzzy-Based Transparent Model	Ahmed and Al-Jamimi	2013	IEEE	IET Software	Journal
S39	[78]	Object-Oriented Class Maintainability Prediction Using Internal Quality Attributes	Al Dallal	2013	Elsevier	Information and Software Technology	Journal
S40	[79]	A New Software Maintainability Evaluation Model Based on Multiple Classifiers Combination	Ye et al.	2013	IEEE	International Conference on Quality, Reliability, Risk, Maintenance, and Safety Engineering	Conference
S41	[80]	Tool for Generating Code Metrics for C# Source Code using Abstract Syntax Tree Technique	Singh et al.	2013	ACM DL	ACM SIGSOFT Software EngineeringNotes	Journal
S42	[81]	Application of Group Method of Data Handling model for software maintainability prediction using object-oriented systems	Malhotra and Chug	2014	Springer link	International Journal of System Assurance Engineering and Management	Journal
S43	[82]	Software Maintainability Prediction by Data Mining of Software Code Metrics	Kaur et al.	2014	IEEE	International Conference on Data Mining and Intelligent Computing	Conference
S44	[83]	A Metric Suite for Predicting Software Maintainability in Data Intensive Applications	Malhotra and Chug	2014	Springer link	Transactions on Engineering Technologies	Conference
S45	[84]	SMPLearner: Learning to Predict Software Maintainability	Zhang et al.	2014	Springer link	Automated Software Engineering	Journal
S46	[85]	Validating the Effectiveness of Object-Oriented Metrics for Predicting Maintainability	Kumara et al.	2015	Elsevier	Procedia Computer Science	Conference
S47	[28]	Three Empirical Studies on Predicting Software Maintainability Using Ensemble Methods	Elish et al.	2015	Springer link	Soft Computing	Journal
S48	[86]	A Proposed New Model for Maintainability Index of Open Source Software	Kaur et al.	2015	IEEE	International Conference on Reliability, Infocom Technologies and Optimization	Conference
S49	[87]	Hybrid Functional Link Artificial Neural Network Approach for Predicting Maintainability of Object-Oriented Software	Kumar and Rath	2016	Elsevier	Journal of Systems and Software	Journal
S50	[88]	An Empirical Investigation of Evolutionary Algorithm for Software Maintainability Prediction	Jain et al.	2016	IEEE	IEEE Students Conference on Electrical, Electronics and Computer Science	Conference
S51	[89]	Using Indirect Coupling Metrics to Predict Package Maintainability and Testability	Almugrin et al.	2016	Elsevier	Journal of Systems and Software	Journal
S52	[90]	Software Maintainability Prediction Using Hybrid Neural Network and Fuzzy Logic Approach with Parallel Computing Concept	Kumar and Rath	2017	Springer link	International Journal of System Assurance Engineering and Management	Journal

S53	[91]	Boosting Automatic Commit Classification into Maintenance Activities by Utilizing Source Code Changes	Levin and Yehudai	2017	ACM DL	13th International Conference on Predictive Models and Data Analytics in Software Engineering	Conference
S54	[15]	Performance of Maintainability Index Prediction Models: A Feature Selection Based Study	Reddy and Ojha	2017	Springer link	Evolving Systems	Journal
S55	[36]	Empirical Evaluation of Software Maintainability Based on a Manually Validated Refactoring Dataset	Péter Hegedűs et al.	2018	Elsevier	Information and Software Technology	Journal
S56	[29]	Particle Swarm Optimization-based Ensemble Learning for Software Change Prediction	Malhotraa and Khanna	2018	Elsevier	Information and Software Technology	Journal

Table 18: Quality assessment result.

Study ID	QA 1	QA 2	QA 3	QA 4	QA 5	QA 6	QA 7	QA 8	QA 9	QA 10	QA 11	QA 12	QA 13	QA 14	QA 15	Total score	Rating
S1	Y	Y	Y	Y	Y	Y	Y	P	N	N	N	P	N	P	Y	9	Fair
S2	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	P	Y	N	13.5	Excellent
S3	Y	Y	Y	Y	Y	Y	P	N	N	N	N	N	N	Y	P	8	Fair
S4	Y	N	N	Y	Y	Y	Y	Y	N	N	N	Y	N	Y	Y	9	Fair
S5	Y	Y	Y	Y	Y	Y	Y	P	N	N	N	P	Y	Y	N	10	Good
S6	Y	Y	Y	Y	Y	Y	Y	Y	N	N	N	N	N	Y	Y	10	Good
S7	Y	Y	Y	Y	Y	Y	Y	Y	N	Y	N	P	N	Y	Y	11.5	Good
S8	Y	Y	Y	Y	Y	Y	N	P	N	Y	N	P	P	Y	Y	10.5	Good
S9	Y	Y	Y	Y	P	N	Y	P	N	N	N	P	N	Y	P	8	Fair
S10	Y	Y	Y	Y	Y	Y	Y	N	N	N	N	N	N	Y	Y	9	Fair
S11	Y	Y	Y	Y	Y	Y	Y	P	Y	N	N	Y	Y	Y	Y	12.5	Good
S12	Y	Y	Y	Y	Y	Y	Y	Y	N	Y	N	Y	P	Y	Y	12.5	Good
S13	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	N	P	P	Y	Y	13	Good
S14	Y	Y	Y	Y	N	N	N	N	N	N	N	N	N	Y	Y	6	Fair
S15	Y	Y	Y	Y	N	Y	N	N	N	N	N	P	P	Y	P	7.5	Fair
S16	Y	Y	Y	Y	Y	Y	Y	Y	N	Y	N	P	P	Y	P	11.5	Good
S17	Y	Y	Y	Y	Y	Y	N	Y	N	Y	Y	Y	N	Y	Y	12	Good
S18	Y	Y	Y	Y	Y	Y	P	P	N	Y	Y	Y	P	Y	Y	12.5	Good
S19	Y	Y	Y	Y	Y	N	N	N	N	Y	Y	P	P	Y	Y	10	Good
S20	P	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	P	N	12.5	Good
S21	Y	Y	Y	Y	Y	Y	Y	Y	P	Y	N	P	P	Y	P	13	Good
S22	Y	Y	Y	Y	Y	N	Y	N	N	N	N	N	N	Y	Y	8	Fair
S23	Y	N	N	Y	Y	Y	Y	Y	Y	N	N	N	N	Y	N	8	Fair
S24	P	Y	Y	Y	Y	N	N	N	N	Y	N	P	P	Y	P	7.5	Fair
S25	P	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	N	13.5	Excellent
S26	P	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	14.5	Excellent
S27	Y	Y	Y	Y	Y	Y	Y	Y	P	N	N	N	N	Y	Y	10.5	Good
S28	Y	Y	Y	Y	N	Y	N	Y	P	N	N	N	N	Y	Y	8.5	Fair
S29	Y	Y	Y	Y	Y	N	Y	Y	P	Y	Y	Y	P	Y	N	12	Good
S30	P	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	P	14	Excellent
S31	Y	Y	Y	Y	Y	Y	N	Y	N	Y	N	Y	Y	Y	Y	12	Good
S32	Y	Y	Y	Y	Y	N	Y	Y	N	Y	Y	Y	P	P	N	11	Good
S33	Y	Y	Y	Y	Y	N	Y	Y	P	Y	N	Y	P	Y	N	11	Good
S34	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	N	Y	Y	Y	Y	14	Excellent
S35	Y	N	P	Y	Y	Y	Y	Y	P	Y	N	N	N	Y	Y	10	Good
S36	N	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	P	P	13	Good
S37	Y	Y	Y	Y	Y	Y	N	Y	N	Y	Y	Y	Y	Y	N	12	Good
S38	N	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	N	P	12.5	Good
S39	Y	Y	Y	Y	Y	Y	Y	Y	P	Y	Y	P	P	Y	Y	13.5	Excellent
S40	N	Y	Y	Y	P	N	Y	Y	P	Y	Y	Y	Y	P	N	10.5	Good
S41	P	N	N	Y	Y	Y	Y	Y	P	N	N	N	N	N	P	6	Fair
S42	Y	N	Y	Y	Y	N	Y	Y	N	Y	Y	Y	Y	Y	Y	12	Good
S43	Y	Y	Y	Y	N	Y	Y	Y	N	Y	N	Y	Y	Y	P	11.5	Good
S44	Y	Y	Y	Y	Y	Y	Y	Y	P	Y	Y	Y	Y	Y	Y	14.5	Excellent
S45	Y	Y	Y	Y	Y	Y	Y	Y	P	Y	Y	Y	Y	Y	Y	14.5	Excellent
S46	N	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	P	P	P	12.5	Good
S47	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	15	Good
S48	P	Y	Y	Y	Y	Y	Y	Y	P	Y	N	Y	Y	P	Y	12.5	Good
S49	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	15	Excellent
S50	N	Y	Y	Y	Y	Y	Y	Y	P	Y	N	Y	Y	P	N	11	Good
S51	Y	Y	Y	Y	Y	Y	Y	N	P	Y	N	N	N	Y	Y	10.5	Good
S52	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	P	Y	P	14	Excellent
S53	P	Y	Y	N	Y	P	Y	Y	Y	Y	Y	Y	P	Y	P	12	Good
S54	Y	Y	Y	Y	Y	Y	Y	Y	P	Y	Y	Y	Y	Y	Y	13.5	Excellent
S55	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	N	P	N	Y	Y	12.5	Good
S56	Y	Y	Y	Y	Y	Y	Y	N	P	Y	Y	Y	Y	Y	Y	13.5	Good

Appendix B

See Figure 19.

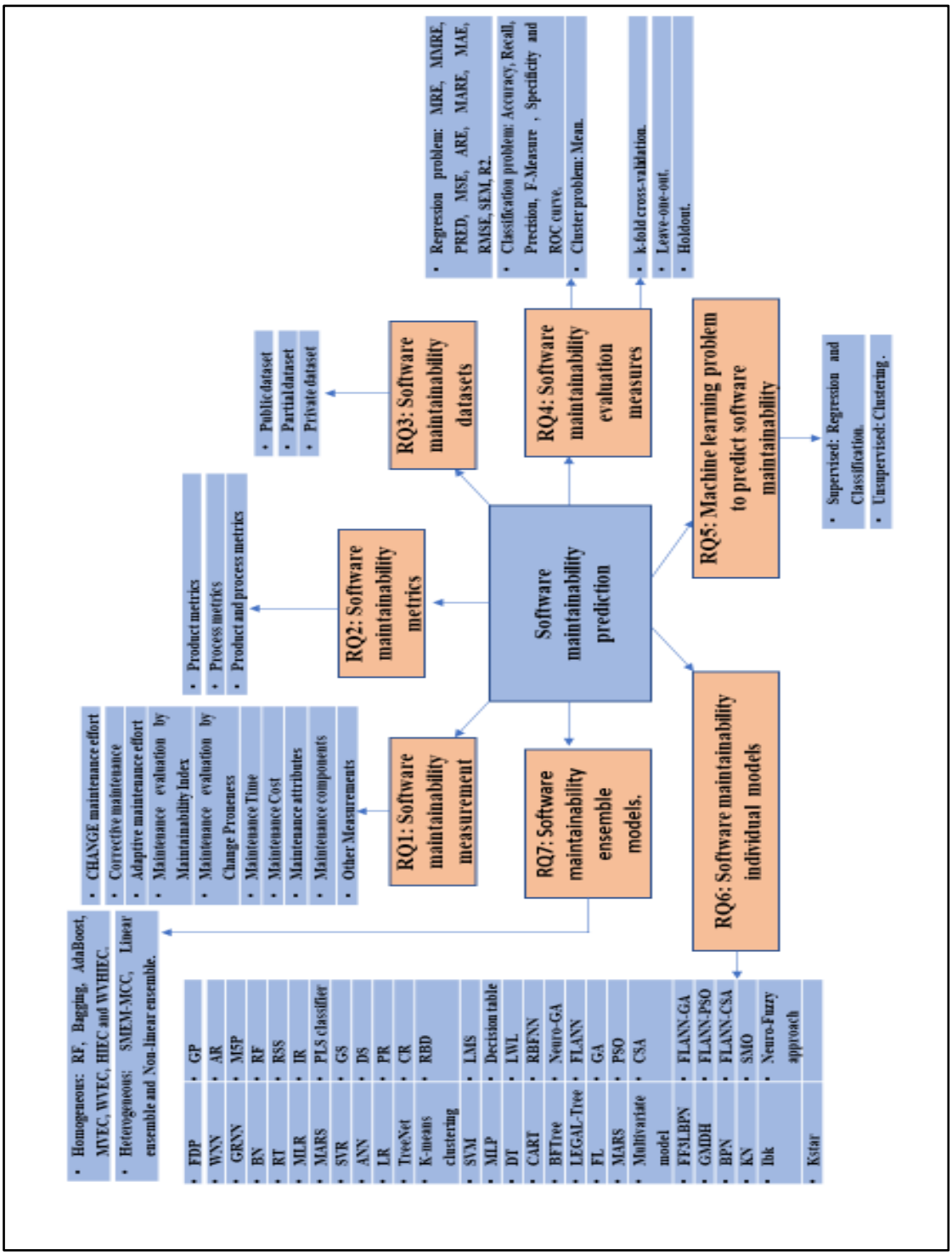

Figure 19: The details of mind map in software maintainability prediction.

References

[1] M.A. Ahmed, H.A. Al-Jamimi, Machine learning approaches for predicting software maintainability: a fuzzy-based transparent model, IET Software, 7 (2013) 317-326.

[2] M. Jorgensen, M. Shepperd, A systematic review of software development cost estimation studies, IEEE Transactions on software engineering, 33 (2007).

[3] A. De Lucia, E. Pompella, S. Stefanucci, Assessing effort estimation models for corrective maintenance through empirical studies, Information and Software Technology, 47 (2005) 3-15.

[4] M.O. Elish, K.O. Elish, Application of TreeNet in Predicting Object-Oriented Software Maintainability: A Comparative Study, in: 13th European Conference on Software Maintenance and Reengineering, 2009, pp. 69-78.

[5] C. van Koten, A.R. Gray, An application of Bayesian network for predicting object-oriented software maintainability, Information and Software Technology, 48 (2006) 59-67.

[6] M. Riaz, E. Mendes, E. Tempero, A systematic review of software maintainability prediction and metrics, in: 2009 3rd International Symposium on Empirical Software Engineering and Measurement, 2009, pp. 367-377.

[7] A. Shafiabady, M.N.r. Mahrin, M. Samadi, Investigation of software maintainability prediction models, in: Advanced Communication Technology (ICACT), 2016 18th International Conference on, IEEE, 2016, pp. 783-786.

[8] R. Kumar, N. Dhanda, Maintainability Quantification of Object Oriented Design: A Revisit, International Journal, 4 (2014).

[9] G. Tiwari, A. Sharma, Maintainability Techniques for Software Development Approaches–A Systematic Survey, Special Issue of International Journal of Computer Applications (0975-8887), ICNICT, (2012).

[10] R. Burrows, A. Garcia, F. Taïani, Coupling metrics for aspect-oriented programming: A systematic review of maintainability studies, in: International Conference on Evaluation of Novel Approaches to Software Engineering, Springer, 2008, pp. 277-290.

[11] M. Riaz, Maintainability prediction of relational database-driven applications: a systematic review, (2012).

[12] B. Kitchenham, Procedures for performing systematic reviews, Keele, UK, Keele University, 33 (2004) 1-26.

[13] P. Brereton, B.A. Kitchenham, D. Budgen, M. Turner, M. Khalil, Lessons from applying the systematic literature review process within the software engineering domain, Journal of Systems and Software, 80 (2007) 571-583.

[14] R. Malhotra, A systematic review of machine learning techniques for software fault prediction, Applied Soft Computing, 27 (2015) 504-518.

[15] B.R. Reddy, A. Ojha, Performance of Maintainability Index prediction models: a feature selection based study, Evolving Systems, (2017) 1-26.

[16] A. Liberati, D.G. Altman, J. Tetzlaff, C. Mulrow, P.C. Gøtzsche, J.P.A. Ioannidis, M. Clarke, P.J. Devereaux, J. Kleijnen, D. Moher, The PRISMA statement for reporting systematic reviews and meta-analyses of studies that evaluate healthcare interventions: explanation and elaboration, BMJ, 339 (2009).

[17] D. Zhang, J.J. Tsai, Machine learning and software engineering, Software Quality Journal, 11 (2003) 87-119.

[18] P. Oman, J. Hagemeister, Metrics for assessing a software system's maintainability, in: Software Maintenance, IEEE, 1992, pp. 337-344.

[19] I. Kádár, P. Hegedus, R. Ferenc, T. Gyimóthy, A Code Refactoring Dataset and Its Assessment Regarding Software Maintainability, in: 23rd International Conference on Software Analysis, Evolution, and Reengineering 2016, pp. 599-603.

[20] S.R. Chidamber, C.F. Kemerer, Towards a metrics suite for object oriented design, in: Conference proceedings on Object-oriented programming systems, languages, and applications, ACM, 1991, pp. 197-211.

[21] H. Aljamaan, M.O. Elish, I. Ahmad, An Ensemble of Computational Intelligence Models for Software Maintenance Effort Prediction, in, Springer Berlin Heidelberg, 2013, pp. 592-603.

[22] T. Menzies, Krishna, R., Pryor, D., The Promise Repository of Empirical Software Engineering Data, in, 2016.

[23] IEEE Std 610.12-1990: IEEE Standard Glossary of Software Engineering Terminology, IEEE, 1990.

[24] O.i.d. normalisation, Systems and Software Engineering: Systems and Software Quality Requirements and Evaluation (SQuaRE): System and Software Quality Models, ISO/IEC, 2011.

[25] R. Land, Measurements of software maintainability, in: Proceedings of the 4th ARTES Graduate Student Conference, 2002, pp. 1-7.

[26] W. Li, S. Henry, Object-oriented metrics that predict maintainability, The Journal of Systems & Software, 23 (1993) 111-122.

[27] P. Oman, J. Hagemeister, Construction and testing of polynomials predicting software maintainability, Journal of Systems and Software, 24 (1994) 251-266.

[28] M.O. Elish, H. Aljamaan, I. Ahmad, Three empirical studies on predicting software maintainability using ensemble methods, Soft Computing, 19 (2015) 2511-2524.

[29] R. Malhotra, M. Khanna, Particle swarm optimization-based ensemble learning for software change prediction, Information and Software Technology, 102 (2018) 65-84.

[30] S.R. Chidamber, C.F. Kemerer, A metrics suite for object oriented design, IEEE Transactions on software engineering, 20 (1994) 476-493.

[31] K.K. Aggarwal, Y. Singh, J.K. Chhabra, An integrated measure of software maintainability, in: Annual Reliability and Maintainability Symposium. 2002 Proceedings (Cat. No.02CH37318), 2002, pp. 235-241.

[32] L.C. Briand, S. Morasca, V.R. Basili, Measuring and assessing maintainability at the end of high level design, in: 1993 Conference on Software Maintenance, 1993, pp. 88-87.

[33] M. Polo, M. Piattini, F. Ruiz, Using code metrics to predict maintenance of legacy programs: a case study, in: Proceedings IEEE International Conference on Software Maintenance. ICSM 2001, 2001, pp. 202-208.

[34] M.I. Sarwar, W. Tanveer, I. Sarwar, W. Mahmood, A comparative study of MI tools: Defining the Roadmap to MI tools standardization, in: 2008 IEEE International Multitopic Conference, 2008, pp. 379-385.

[35] C. Sammut, G.I. Webb, Encyclopedia of machine learning, Springer Science & Business Media, 2011.

[36] P. Hegedűs, I. Kádár, R. Ferenc, T. Gyimóthy, Empirical evaluation of software maintainability based on a manually validated refactoring dataset, Information and Software Technology, 95 (2018) 313-327.

[37] Github - the largest open source community in the world, in.

[38] sourceforge, in.

[39] M. Genero, J. Olivas, M. Piattini, F. Romero, Using metrics to predict OO information systems maintainability, in: International Conference on Advanced Information Systems Engineering, Springer, 2001, pp. 388-401.

[40] V. Nguyen, B. Boehm, P. Danphitsanuphan, A controlled experiment in assessing and estimating software maintenance tasks, Information and Software Technology, 53 (2011) 682-691.

[41] Coefficient of determination, in.

[42] S.D. Conte, H.E. Dunsmore, Y.E. Shen, Software engineering metrics and models, Benjamin-Cummings Publishing Co., Inc., 1986.

[43] M. Shepperd, S. MacDonell, Evaluating prediction systems in software project estimation, Information and Software Technology, 54 (2012) 820-827.

[44] T. Menzies, M. Shepperd, Special issue on repeatable results in software engineering prediction, in, Springer, 2012.

[45] S.G. MacDonell, Establishing relationships between specification size and software process effort in CASE environments, Information and Software Technology, 39 (1997) 35-45.

[46] B.A. Kitchenham, L.M. Pickard, S.G. MacDonell, M.J. Shepperd, What accuracy statistics really measure [software estimation], IEE Proceedings - Software, 148 (2001) 81-85.

[47] S. Basri, N. Kama, H.M. Sarkan, S. Adli, F. Haneem, An algorithmic-based change effort estimation model for software development, in: 2016 23rd Asia-Pacific Software Engineering Conference (APSEC), IEEE, 2016, pp. 177-184.

[48] S.K. Sehra, Y.S. Brar, N. Kaur, G. Kaur, Optimization of COCOMO Parameters using TLBO Algorithm, International Journal of Computational Intelligence Research, 13 (2017) 525-535.

[49] A. Srivastava, S. Singh, S.Q. Abbas, Performance Measure of the Proposed Cost Estimation Model: Advance Use Case Point Method, in: Soft Computing: Theories and Applications, Springer, 2019, pp. 223-233.

[50] M.O. Elish, T. Helmy, M.I. Hussain, Empirical study of homogeneous and heterogeneous ensemble models for software development effort estimation, Mathematical Problems in Engineering, 2013 (2013).

[51] N. Raj Kiran, V. Ravi, Software reliability prediction by soft computing techniques, Journal of Systems and Software, 81 (2008) 576-583.

[52] D. Opitz, R. Maclin, Popular ensemble methods: An empirical study, Journal of artificial intelligence research, 11 (1999) 169-198.

[53] G.K. Gill, C.F. Kemerer, Cyclomatic complexity density and software maintenance productivity, IEEE Transactions on Software Engineering, 17 (1991) 1284-1288.

[54] D. Coleman, D. Ash, B. Lowther, P. Oman, Using metrics to evaluate software system maintainability, Computer, 27 (1994) 44-49.

[55] J. Daly, A. Brooks, J. Miller, M. Roper, M. Wood, Evaluating inheritance depth on the maintainability of object-oriented software, Empirical Software Engineering, 1 (1996) 109-132.

[56] K.D. Welker, P.W. Oman, G.G. Atkinson, Development and application of an automated source code maintainability index, Journal of Software Maintenance: Research and Practice, 9 (1997) 127-159.

[57] J.C. Granja-Alvarez, M.J. Barranco-García, A method for estimating maintenance cost in a software project: a case study, Journal of Software Maintenance: Research and Practice, 9 (1997) 161-175.

[58] Y. Lee, K.H. Chang, Reusability and maintainability metrics for object-oriented software, in: Proceedings of the 38th annual on Southeast regional conference, ACM, 2000, pp. 88-94.

[59] F. Fioravanti, P. Nesi, Estimation and prediction metrics for adaptive maintenance effort of object-oriented systems, IEEE Transactions on Software Engineering, 27 (2001) 1062-1084.

[60] F.T. Sheldon, K. Jerath, H. Chung, Metrics for maintainability of class inheritance hierarchies, Journal of Software Maintenance, 14 (2002) 147-160.

[61] M. Dagpinar, J.H. Jahnke, Predicting maintainability with object-oriented metrics -an empirical comparison, in: 10th Working Conference on Reverse Engineering., 2003, pp. 155-164.

[62] R.K. Bandi, V.K. Vaishnavi, D.E. Turk, Predicting maintenance performance using object-oriented design complexity metrics, IEEE Transactions on Software Engineering, 29 (2003) 77-87.

[63] J.H. Hayes, S.C. Patel, L. Zhao, A metrics-based software maintenance effort model, in: Eighth European Conference on Software Maintenance and Reengineering, 2004, pp. 254-258.

[64] M.M.T. Thwin, T.-S. Quah, Application of neural networks for software quality prediction using object-oriented metrics, Journal of Systems and Software, 76 (2005) 147-156.

[65] S.C. Misra, Modeling Design/Coding Factors That Drive Maintainability of Software Systems, Software Quality Journal, 13 (2005) 297-320.

[66] J.S. Lim, S.R. Jeong, S.R. Schach, An empirical investigation of the impact of the object-oriented paradigm on the maintainability of real-world mission-critical software, Journal of Systems and Software, 77 (2005) 131-138.

[67] B. Cukic, Guest Editor's Introduction: The Promise of Public Software Engineering Data Repositories, IEEE Software, 22 (2005) 20-22.

[68] J.H. Hayes, L. Zhao, Maintainability prediction: a regression analysis of measures of evolving systems, in: 21st IEEE International Conference on Software Maintenance 2005, pp. 601-604.

[69] Y. Zhou, H. Leung, Predicting object-oriented software maintainability using multivariate adaptive regression splines, Journal of Systems and Software, 80 (2007) 1349-1361.

[70] R. Lincke, J. Lundberg, W. Löwe, Comparing software metrics tools, in: Proceedings of the 2008 international symposium on Software testing and analysis, ACM, 2008, pp. 131-142.

[71] Y. Zhou, B. Xu, Predicting the maintainability of open source software using design metrics, Wuhan University Journal of Natural Sciences, 13 (2008) 14-20.

[72] J.-C. Chen, S.-J. Huang, An empirical analysis of the impact of software development problem factors on software maintainability, Journal of Systems and Software, 82 (2009) 981-992.

[73] C. Jin, J.-A. Liu, Applications of support vector mathine and unsupervised learning for predicting maintainability using object-oriented metrics, in: Multimedia and Information Technology (MMIT), 2010 Second International Conference on, IEEE, 2010, pp. 24-27.

[74] K. Kaur, H. Singh, Determination of Maintainability Index for Object Oriented Systems, SIGSOFT Softw. Eng. Notes, 36 (2011) 1-6.

[75] P. Bhattacharya, I. Neamtiu, Assessing programming language impact on development and maintenance: a study on c and c++, in: Proceedings of the 33rd International Conference on Software Engineering, ACM, 2011, pp. 171-180.

[76] S.K. Dubey, A. Rana, Y. Dash, Maintainability prediction of object-oriented software system by multilayer perceptron model, SIGSOFT Softw. Eng. Notes, 37 (2012) 1-4.

[77] M.P. Basgalupp, R.C. Barros, D.D. Ruiz, Predicting software maintenance effort through evolutionary-based decision trees, in: Proceedings of the 27th Annual ACM Symposium on Applied Computing, ACM, 2012, pp. 1209-1214.

[78] J. Al Dallal, Object-oriented class maintainability prediction using internal quality attributes, Information and Software Technology, 55 (2013) 2028-2048.

[79] F. Ye, X. Zhu, Y. Wang, A new software maintainability evaluation model based on multiple classifiers combination, in: 2013 International Conference on Quality, Reliability, Risk, Maintenance, and Safety Engineering (QR2MSE), 2013, pp. 1588-1591.

[80] P. Singh, S. Singh, J. Kaur, Tool for generating code metrics for C# source code using abstract syntax tree technique, SIGSOFT Softw. Eng. Notes, 38 (2013) 1-6.

[81] R. Malhotra, A. Chug, Application of Group Method of Data Handling model for software maintainability prediction using object oriented systems, International Journal of System Assurance Engineering and Management, 5 (2014) 165-173.

[82] A. Kaur, K. Kaur, K. Pathak, Software maintainability prediction by data mining of software code metrics, in: 2014 International Conference on Data Mining and Intelligent Computing (ICDMIC), 2014, pp. 1-6.

[83] R. Malhotra, A. Chug, A Metric Suite for Predicting Software Maintainability in Data Intensive Applications, in, Springer Netherlands, 2014, pp. 161-175.

[84] W. Zhang, L. Huang, V. Ng, J. Ge, SMPLearner: learning to predict software maintainability, Automated Software Engineering, 22 (2015) 111-141.

[85] L. Kumar, D.K. Naik, S.K. Rath, Validating the Effectiveness of Object-Oriented Metrics for Predicting Maintainability, Procedia Computer Science, 57 (2015) 798-806.

[86] A. Kaur, K. Kaur, K. Pathak, A proposed new model for maintainability index of open source software, in: Proceedings of 3rd International Conference on Reliability, Infocom Technologies and Optimization, 2014, pp. 1-6.

[87] L. Kumar, S.K. Rath, Hybrid functional link artificial neural network approach for predicting maintainability of object-oriented software, Journal of Systems and Software, 121 (2016) 170-190.

[88] A. Jain, S. Tarwani, A. Chug, An empirical investigation of evolutionary algorithm for software maintainability prediction, in: 2016 IEEE Students' Conference on Electrical, Electronics and Computer Science (SCEECS), 2016, pp. 1-6.

[89] S. Almugrin, W. Albattah, A. Melton, Using indirect coupling metrics to predict package maintainability and testability, Journal of Systems and Software, 121 (2016) 298-310.

[90] L. Kumar, S.K. Rath, Software maintainability prediction using hybrid neural network and fuzzy logic approach with parallel computing concept, International Journal of System Assurance Engineering and Management, 8 (2017) 1487-1502.

[91] S. Levin, A. Yehudai, Boosting Automatic Commit Classification Into Maintenance Activities By Utilizing Source Code Changes, in: Proceedings of the 13th International Conference on Predictive Models and Data Analytics in Software Engineering, ACM, 2017, pp. 97-106.